
RobustSpring: Benchmarking Robustness to Image Corruptions for Optical Flow, Scene Flow and Stereo

Jenny Schmalfuss*

Victor Oei*

Lukas Mehl

Madlen Bartsch

Shashank Agnihotri

Margret Keuper

Andres Bruhn

A Supplementary Material

A.1 Dataset, Benchmark and Code Repository

A preliminary version of the benchmark website is online at <https://spring-benchmark.org>. The code for the website and our data subsampling procedure to upload method evaluation to the benchmark can be found at <https://github.com/cv-stuttgart/springwebsite>. Please note that instructions for uploading results to the website are at <https://spring-benchmark.org/faq>. The website and subsampling code are licensed under an MIT license.

For the RobustSpring dataset, the raw image data files are permanently stored at <https://doi.org/10.18419/DARUS-5047>. For convenient loading and evaluation for optical flow, scene flow and stereo, we provide the filepath-based RobustSpring datasets on Huggingface that pair the correct frame pairs or quadruples on <https://huggingface.co/datasets/jeschmalfuss/RobustSpring>. The RobustSpring dataset (raw files and the Huggingface dataset) are licensed under CC-BY-4.0, following the license of the Spring [7] dataset.

A.2 Links and Checkpoints for Evaluated Models

We evaluated the original, author provided optical flow, scene flow and stereo methods on the RobustSpring dataset. Tab. A1 reports the repositories and checkpoints for the optical flow, scene flow and stereo models, which were benchmarked on RobustSpring in Tab. 1 and Tab. 2. Further details on training and checkpoints for these models can be found in their original publications.

Ressources and Run Times. We conducted all evaluations on a single NVIDIA RTX A6000 (48 GB) GPU. The evaluation time of models on the RobustSpring data strongly depends on the computational efficiency and requirements of the original models. As a representative example, the optical flow evaluation with RAFT [9] took 20 hours on the full RobustSpring data.

Once the methods are evaluated and the results uploaded to the RobustSpring benchmark website, the robustness evaluation for an optical flow method takes 13 minutes on our server, which accesses Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz, 8 vCPUs. The robustness evaluation for scene flow takes about 26 minutes, and 7 minutes for stereo.

A.3 RobustSpring Image Corruptions

Below, we provide supplementary information on the image corruptions for the RobustSpring dataset. Besides visualizing further benchmark samples and supplying a video that showcases the space- and time-integration of our corruptions, we also give details on their implementation with a focus on RobustSpring-specific consistencies.

*Equal Contribution

Table A1: Repositories and checkpoints used for evaluating methods in RobustSpring. *The Mixed checkpoint MS-RAFT+ is pretrained on Chairs and Things and then fine-tuned on a mix of Sintel, Viper, KITTI 2015, Things, and HD1k.

Method	Repository	Checkpoint
Optical Flow		
RAFT	https://github.com/princeton-vl/RAFT	Sintel
PWCNet	https://github.com/NVlabs/PWC-Net	Sintel
GMFlow	https://github.com/haofeixu/gmflow	Sintel
GMA	https://github.com/zacjiang/GMA	Sintel
FlowNet2	https://github.com/NVIDIA/flowNet2-pytorch	Sintel
FlowFormer	https://github.com/drinkingcoder/FlowFormer-Official	Sintel
MS-RAFT+	https://github.com/cv-stuttgart/MS-RAFT_plus	Mixed*
SPyNet	https://github.com/anuragranj/flowattack (PyTorch implementation) https://github.com/anuragranj/spynet (original implementation)	Sintel
Scene Flow		
M-FUSE	https://github.com/cv-stuttgart/M-FUSE	KITTI 2015
Stereo		
RAFT-Stereo	https://github.com/princeton-vl/RAFT-Stereo/	Scene Flow (s)
ACVNet	https://github.com/gangweiX/ACVNet	Scene Flow (s)
LEAStereo	https://github.com/XuelianCheng/LEAStereo	Scene Flow (s), KITTI (k)
GANet	https://github.com/feihuzhang/GANet	Scene Flow (s), KITTI (k)

32 A.3.1 Additional Corruption Benchmark Samples

33 To complement the benchmark samples in Fig. 3, we show benchmark results on additional corruptions
34 in Fig. A1.

35 A.3.2 Video of Corruptions

36 With the supplementary material, we include a video that visualizes all corruptions applied to the
37 Spring test dataset.

38 A.3.3 Corruption Implementation

39 Below we provide the implementation details and parameters for all corruptions in the RobustSpring
40 dataset. We cluster the corruptions by their main classes. The original (uncorrupted) image is denoted
41 as I , while the corrupted version is \hat{I} . The pixel-aligned depth values are D . In the stereo video
42 setting, the image subscripts t and $t + 1$ denote frames over time, while l and r denote left and right
43 frame, where necessary. All of RobustSpring’s noises are independent of time, stereo and depth,
44 which means they are sampled independently for every single image of the dataset.

45 **Brightness.** The brightness is adapted via

$$\hat{I} = I + c, \quad (1)$$

46 and for time- and stereo-consistent brightness changes in RobustSpring we choose the parameter
47 $c = c_t^l = c_t^r = c_{t+1}^l = c_{t+1}^r = 0.39$.

48 **Contrast.** The equation to adapt contrast is

$$\hat{I} = (I - \text{mean}(I)) \cdot c + \text{mean}(I), \quad (2)$$

49 where we selected $c = c_t^l = c_t^r = c_{t+1}^l = c_{t+1}^r = 0.16$ for time- and stereo-consistent contrast
50 adaptations.

51 **Saturation.** For those adaptations the RGB image is transformed to HSV, and the saturation
52 component S is adapted via

$$\hat{S} = S \cdot \alpha + \beta, \quad (3)$$

53 with $\alpha = \alpha_t^l = \alpha_t^r = \alpha_{t+1}^l = \alpha_{t+1}^r = 2.3$ and $\beta_t^l = \beta_t^r = \beta_{t+1}^l = \beta_{t+1}^r = 0.01$ for time- and
54 stereo-consistent saturation changes.

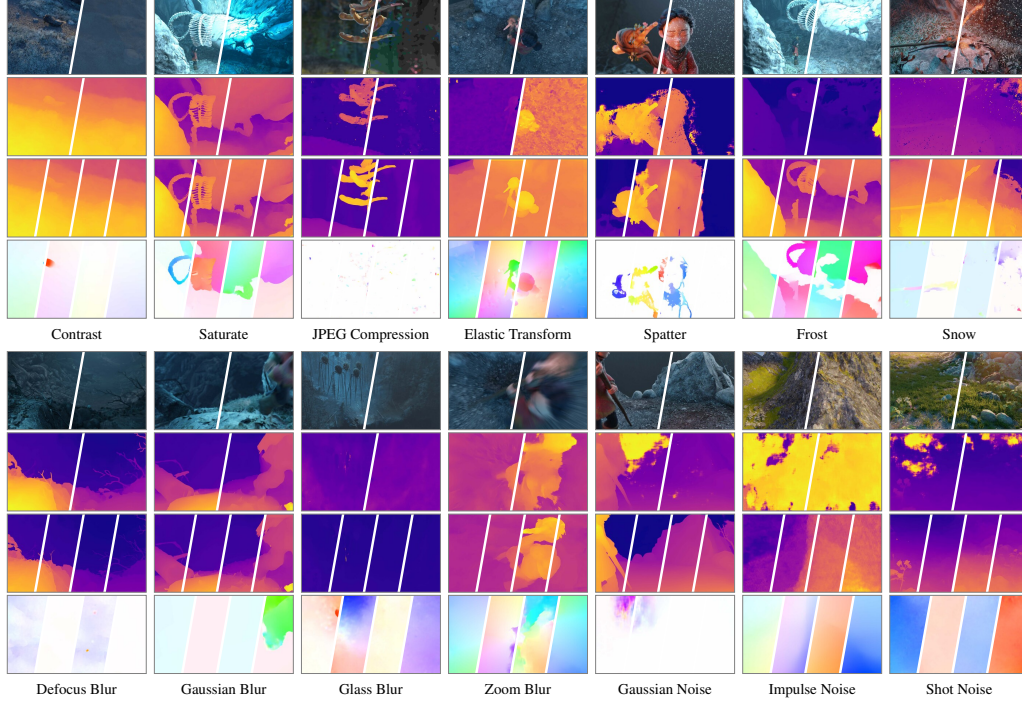


Figure A1: RobustSpring example frames, complementing Fig. 3. The first row shows clean and corrupted images. The second row shows the left and right disparity maps predicted with LEA Stereo [1]. The third row shows the target disparities for forward left, backward left, forward right, and backward right directions from M-FUSE [6]. The fourth row shows optical flow estimates for forward left, backward left, forward right, and backward right from RAFT [9]. All disparities and flows are computed on the corrupted dataset.

55 **Defocus Blur.** The defocus blur convolves the image with a circular mean filter C^{mean}

$$\hat{I} = I * C_r^{mean}, \quad (4)$$

56 where we choose the radius $r = r_t^l = r_t^r = r_{t+1}^l = r_{t+1}^r = 6$ for time- and stereo-consistent blurring.

57 **Gaussian Blur.** Gaussian blur convolves the image with a Gaussian C^{gauss}

$$\hat{I} = I * C_\sigma^{Gauss}, \quad (5)$$

58 where we choose the standard deviation $\sigma = \sigma_t^l = \sigma_t^r = \sigma_{t+1}^l = \sigma_{t+1}^r = 4$ for time- and stereo-
59 consistent blurring.

60 **Glass Blur.** This is a Gauss-blurred image, whose pixels are afterwards shuffled via the shuffling
61 $S(I, i, r)$ over several iterations i within a neighborhood of radius r

$$\hat{I} = S(I * C_\sigma^{Gauss}, i, r), \quad (6)$$

62 where two sets of time-consistent parameters are picked for the different stereo cameras: $\sigma_t^l =$
63 $\sigma_{t+1}^l = 1.2$, $\sigma_t^r = \sigma_{t+1}^r = 1.2$, $i_t^l = i_{t+1}^l = 1$, $i_t^r = i_{t+1}^r = 1$, $r_t^l = r_{t+1}^l = 3$ and $r_t^r = r_{t+1}^r = 3$.

64 **Motion Blur.** Motion blur is implemented by averaging the intensities of pixels along the motion
65 trajectory determined by the optical flow. Let $\mathbf{v}(x, y) = (v_x(x, y), v_y(x, y))$ be the optical flow
66 vector at pixel (x, y) , and let

$$N = \max \left(1, \left\lceil 10 \cdot \max_{(x,y)} \|\mathbf{v}(x, y)\| \right\rceil \right) \quad (7)$$

67 be the number of samples along the motion path. Then, the blurred pixel is computed as

$$\hat{I}(x, y) = \frac{1}{N+1} \sum_{k=0}^N I\left(x + \frac{k}{N} v_x(x, y), y + \frac{k}{N} v_y(x, y)\right). \quad (8)$$

68 Here, the scaling factor 10 controls the extent of the blur relative to the magnitude of the motion.

69 **Zoom Blur.** Zoom blur is created by averaging the original image with a series of zoomed-in versions
 70 of itself. Specifically, let $Z(I, z)$ denote the image I zoomed by a factor z , and let $\{z_i\}$ be a set of
 71 zoom factors ranging from 1 to approximately 1.24 (in increments of 0.02). Then the final image is
 72 computed as

$$\hat{I} = \frac{1}{N+1} \left(I + \sum_{i=1}^N Z(I, z_i) \right), \quad (9)$$

73 where N is the number of zoom factors. This formulation averages the original image with its
 74 progressively zoomed versions, resulting in a smooth zoom blur effect.

75 **Gaussian Noise.** This noise adds a random value from a Normal distribution to every pixel in the
 76 original image, where $\mathcal{N}_I(\mu, \sigma^2)$ is a I -shaped array of random numbers that are drawn from the
 77 Normal distribution with mean μ and variance σ^2 :

$$\hat{I} = I + \alpha \cdot \mathcal{N}_I(0, 1). \quad (10)$$

78 The scaling $\alpha = 0.115$ is selected for all images in RobustSpring, but $\mathcal{N}_I(0, 1)$ is sampled anew for
 79 every image.

80 **Impulse Noise.** Here, for a fixed fraction of pixels p , their values are replaced by the values 0 or 255.
 81 For RobustSpring, $p = 0.075$.

82 **Speckle Noise.** Like Gaussian noise, Speckle noise also builds on random values from a Normal
 83 distribution, but adds these values after additionally scaling with I :

$$\hat{I} = I + I \cdot \alpha \cdot \mathcal{N}_I(0, 1). \quad (11)$$

84 For RobustSpring, the parameter is $\alpha = 0.45$.

85 **Shot Noise.** Shot noise uses values drawn from a Poisson distribution \mathcal{P} per pixel

$$\hat{I} = \frac{\mathcal{P}(I \cdot c)}{c}, \quad (12)$$

86 where $c = 23$ for RobustSpring.

87 **Pixelation.** This is achieved by downsampling the image to size s , a fraction of its original size, with
 88 a box filter $\text{box}(I, s)$, followed by upsampling $\text{up}(I, s)$ to the size s , which is the original size:

$$\hat{I} = \text{up}(\text{box}(I, I \cdot c), I). \quad (13)$$

89 For RobustSpring, we use the size fraction $c = c_t^l = c_t^r = c_{t+1}^l = c_{t+1}^r = 0.16$ for time- and
 90 stereo-consistent pixelation.

91 **JPEG Compression.** For JPEG compression, the quality q is the only variable parameter

$$\hat{I} = \text{JPEG}(I, q), \quad (14)$$

92 which is selected as $q = q_t^l = q_t^r = q_{t+1}^l = q_{t+1}^r = 6$ for time- and stereo-consistent JPEG
 93 compression.

94 **Elastic Transformation.** The elastic transformation applies an elastic deformation using
 95 `torchvision.transforms.v2` with parameters $\alpha = 110.0$ and $\sigma = 5.0$ to control the deformation
 96 magnitude and smoothness, while preserving the original frame dimensions.

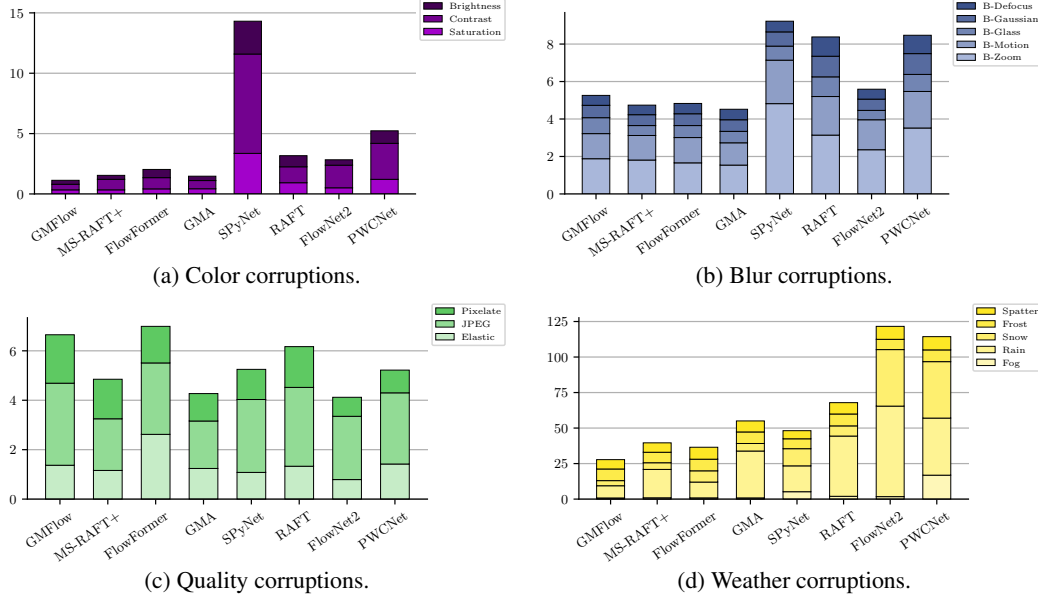


Figure A2: Additional results on accumulated corruption robustness R_{EPE}^c for optical flow models over corruption classes color, blur, quality, and weather. More results are in Fig. 4a and Fig 4b.

97 **Spatter.** The spatter corruption simulates liquid droplets by generating a liquid layer from Gaussian
 98 noise, applying blur and thresholding, and blending it with the original image using a predefined
 99 color.

100 **Frost.** The frost corruption overlays a frost texture onto the image by randomly selecting and resizing
 101 a pre-stored frost image and blending it with the input to create an icy appearance.

102 **Snow and Rain.** The implementation for snow and rain is based on [8], with methodological and
 103 performance improvements. On the methodological side, we replaced additive blending with order-
 104 independent alpha blending (Meshkin’s Method) [5] and included global illumination [3] in the color
 105 rendering. Also, we expanded the monocular two-step motion simulation to multi-step stereo images.
 106 On the performance side, we introduce an efficient parallel particle initialization and improve the
 107 parallel processing performance.

108 **Fog.** Fog is modelled using the Koschmieder model from [10] as

$$\hat{I} = I \cdot e^{-\frac{D \cdot \ln(20)}{d_m}} + l \cdot (1 - e^{-\frac{D \cdot \ln(20)}{d_m}}), \quad (15)$$

109 where d_m is the visibility range and l the luminance of the sky. For RobustSpring, we use $d_m = 45$
 110 and $l = 0.8$. As it directly depends on the depth D , this is depth-consistent and due to its integration
 111 into the 3D scene it is also stereo- and time-consistent.

112 A.4 Additional Experimental Results

113 Below, we expand on the experiments in the main paper and provide supplementary results for our
 114 major experiments.

115 A.4.1 Corruption Robustness by Corruption Group

116 Figure A2 shows the corruption robustness R_{EPE}^c for each optical flow method across the remaining
 117 four corruption groups in addition to Fig. 4a and Fig 4b in the main paper. It underlines the varying
 118 degrees of robustness of the evaluated methods against specific types of corruption.

119 A.4.2 Accuracy vs. Median Corruption Robustness

120 In Fig. A3 we show the accuracy-robustness evaluation with the *Median* corruption robustness, to
 121 complement Fig. 4c which uses the average corruption robustness. Even though the robustness

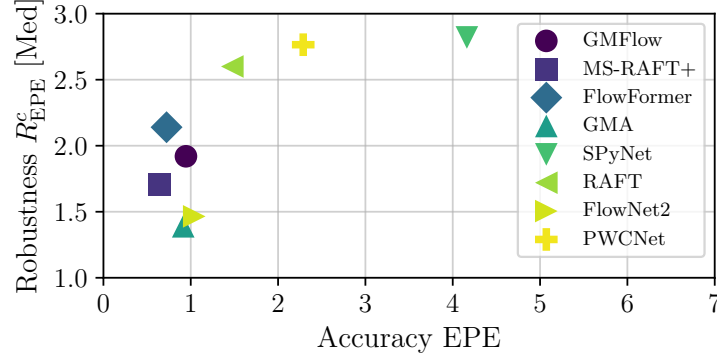


Figure A3: Accuracy vs. robustness of optical flow methods, measured as EPE and median R_{EPE}^c . Small values indicate accurate and robust methods. Fig. 4c shows the average R_{EPE}^c .

Table A2: Pairwise comparison matrix for the Schulze method.

	GMFlow	GMA	MS-RAFT+	FlowFormer	SPyNet	RAFT	PWCNet	FlowNet2
GMFlow	0	9	7	16	10	16	14	12
GMA	11	0	9	19	12	17	14	14
MS-RAFT+	13	11	0	19	12	18	15	14
FlowFormer	4	1	1	0	8	9	13	8
SPyNet	10	8	8	12	0	12	14	5
RAFT	4	3	2	11	8	0	14	8
PWCNet	6	6	5	7	6	6	0	4
FlowNet2	8	6	6	12	15	12	16	0

ranking of methods varies between average and media corruption robustness, c.f. Sec. 4.2 for a discussion on ranking differences, the general trend that corruption robustness and accuracy are weakly correlated remains. However, there is still no clear winner, and an accuracy-robustness tradeoff persists among particularly accurate or robust methods.

A.4.3 Schulze Pairwise Comparison Matrix

The Schulze method is a ranking algorithm used to determine the most preferred candidate based on pairwise comparisons. We include a pairwise comparison matrix in Table A2 for our ranking. The table shows how often the method in row i is better than the method in column j , based on the number of corruptions where method i achieves a lower error than method j . The ranking process consists of the following steps:

1) Constructing the Pairwise Comparison Matrix. For each pair of methods, we count how many times one method achieves a lower EPE than the other across different corruptions. If method A has a lower EPE than method B in a given corruption, the corresponding entry in the matrix is incremented.

2) Computing the Strongest Paths. We define the strength of a path from method A to method B as the number of cases where A outperforms B . The strongest paths between methods are determined by considering indirect paths: if method A is better than method B , and method B is better than method C , then the strength of the indirect path from A to C is considered.

3) Determining the Final Ranking. Method A is ranked higher than method B if the strongest path from A to B is stronger than the strongest path from B to A . This ensures that even if a method loses to another in some comparisons, it can still be ranked higher if it consistently performs well against other methods.

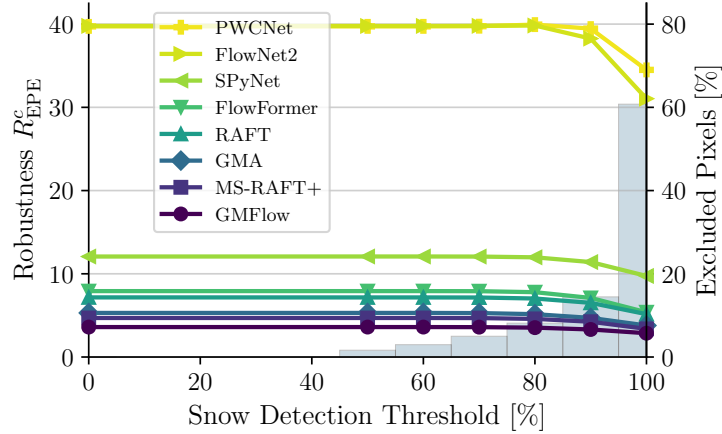


Figure A4: Stability of corruption robustness R_{EPE}^c for snow corruption. Analogous rain results in Fig. 5a.

Table A3: Top 10% noisiest KITTI frames with noise estimation from Immerkaer [4]. Frames from kitti15/dataset/training/image_2.

Frame	000061_10.png	000060_10.png	000065_10.png	000068_10.png	000143_10.png	000069_10.png	000174_10.png	000142_10.png	000107_10.png	000175_10.png	000064_10.png	000063_10.png	000066_10.png	000067_10.png	000062_10.png	000055_10.png	000154_10.png	000158_10.png	000157_10.png	000054_10.png
Noise	3.54	3.22	3.07	2.92	2.80	2.79	2.77	2.74	2.69	2.66	2.62	2.61	2.61	2.60	2.48	2.47	2.46	2.46	2.43	2.41

A.4.4 Corruption Robustness on Snow

Finally, we complement the evaluation of our corruption robustness metric in the presence of rain in Fig. 5a with the corresponding evaluation in the presence of snow in Fig. A4. The results with rain translate to snow with the following minor differences: Because snow has less motion blur than rain, it covers fewer pixels (60% of all pixels vs. 90% for rain). For snow, the score drops a bit more than for rain when object pixels are excluded ($\leq 25\%$ drop vs. $\leq 5\%$ for rain), potentially as a consequence of the increased object opacity for snow particles. Still, the background error ($\geq 75\%$ contribution to corruption robustness) dominates the score, and the robustness ranking for optical flow methods remains stable, whether snow pixels are included in the score calculation or not. Hence, the additional evaluation on snow further substantiates the stability and expressiveness of corruption robustness as an evaluation metric.

Reproducibility. We compute the noise in the KITTI [2] frames according to Immerkaer [4], and select the top 10% noisiest frames. The final frames and their noise levels are listed in Tab. A3.

References

- [1] Xuelian Cheng, Yiran Zhong, Mehrtash Harandi, Yuchao Dai, Xiaojun Chang, Hongdong Li, Tom Drummond, and Zongyuan Ge. Hierarchical neural architecture search for deep stereo matching. *Proc. Conference on Neural Information Processing Systems (NeurIPS)*, 33:22158–22169, 2020.
- [2] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012.
- [3] Shirsendu Sukanta Halder, Jean-François Lalonde, and Raoul de Charette. Physics-based rendering for improving robustness to rain. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10203–10212, 2019.

- 166 [4] John Immerkaer. Fast noise variance estimation. *Computer Vision and Image Understanding (CVIU)*, 64
167 (2):300–302, 1996.
- 168 [5] Morgan McGuire and Louis Bavoil. Weighted blended order-independent transparency. *Journal of*
169 *Computer Graphics Techniques (JCGT)*, 2013.
- 170 [6] Lukas Mehl, Azin Jahedi, Jenny Schmalfluss, and Andrés Bruhn. M-FUSE: multi-frame fusion for scene
171 flow estimation. In *Proc. IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*,
172 pages 2020–2029, 2023.
- 173 [7] Lukas Mehl, Jenny Schmalfluss, Azin Jahedi, Yaroslava Nalivayko, and Andrés Bruhn. Spring: A high-
174 resolution high-detail dataset and benchmark for scene flow, optical flow and stereo. In *Proc. IEEE/CVF*
175 *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4981–4991, 2023.
- 176 [8] Jenny Schmalfluss, Lukas Mehl, and Andrés Bruhn. Distracting downpour: Adversarial weather attacks
177 for motion estimation. In *Proc. IEEE/CVF International Conference on Computer Vision (ICCV)*, pages
178 10106–10116, 2023.
- 179 [9] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. In *Proc. European*
180 *Conference on Computer Vision (ECCV)*, pages 402–419, 2020.
- 181 [10] Thomas Wiesemann and Xiaoyi Jiang. Fog augmentation of road images for performance analysis of
182 traffic sign detection algorithms. In *Proc. International Conference on Advanced Concepts for Intelligent*
183 *Vision Systems (ACVIS)*, 2016.