Location-aware Single Image Reflection Removal
Supplementary Material

In this supplementary material, we provide the detailed architecture of our network and more SIRR results on the benchmark datasets, such as SIR^2 [5], the "Nature" testing datasets from IBCLN [4], the unaligned datasets from ERRNet [6], and the captured real-world testing datasets from Zhang *et al*. [9]. The SIRR results on Internet images are reported as well.

Network Architecture. In Tab. 1 and Tab. 2, we report the detailed parameters of the SE-ResBlock and CBAM-ResBlock used in our network respectively, and the detailed network parameters of our network is reported in Tab. 3. Our network can work well when the width and height of the input images $\mathbf{I}, \hat{\mathbf{T}}_i, \hat{\mathbf{R}}_i, \hat{\mathbf{C}}_i$ are an integer multiple of 8 since the minimal resolution of the features that appear in our network is $1/8$ of input images. Setting the resolution to an integer multiple of 8 can avoid the misalignment solutions caused by convolution and down-sample operations since our network is based on a recurrent structure. Besides, images of resolution up to 1560 in width or height can be input into our network when inferring on an Nvidia Geforce RTX 2080 Ti GPU.

Tab. 4 lists the fine-tuned kernel weights for our multi-scale Laplacian sub-module, and Fig. 1 illustrates the inverse Laplacian maps computed using the fine-tuned kernel weights. It can be seen that the fine-tuned weights remain efficient to suppress low-frequency reflections. Our paper's ablation study also verifies the fine-tuned Laplacian weights can improve the SIRR performance since they are updated with respect to data.

More Iterative Refinement Results. In Fig. 2 and Fig. 3, we show four groups of iterative refinement results of our recurrent network. The improved transmission layers: $\hat{\mathbf{T}}_i$, reflection layers: $\hat{\mathbf{R}}_i$, reflection confidence maps: $\hat{\mathbf{C}}_i$ in iteration i are shown in the two figures respectively.

More Results on Benchmark Datasets. From Fig. 4 to Fig. 7, we show more results of our method on the benchmark datasets. We also compare our method with the state-of-art SIRR methods, such as Zhang *et al*. [9], RMNet [7], ERRNet [6], Kim *et al*. [3], and IBCLN [4].

Results on Internet Images. Fig. 8 shows the SIRR results of our method on the images photographed through glasses collected from the Internet. While these pictures are taken in a variety of scenes, our method can efficiently reconstruct high-quality background transmission layers.

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[2] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7132–7141, 2018.

[3] Soomin Kim, Yuchi Huo, and Sung-Eui Yoon. Single image reflection removal with physically-based training images. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5164–5173, 2020.

[4] Chao Li, Yixiao Yang, Kun He, Stephen Lin, and John E Hopcroft. Single image reflection removal through cascaded refinement. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3565–3574, 2020.

[5] Renjie Wan, Boxin Shi, Ling Yu Duan, Ah Hwee Tan, and Alex C. Kot. Benchmarking single-image reflection removal algorithms. In *Int. Conf. Comput. Vis.*, 2017.

[6] Kaixuan Wei, Jiaolong Yang, Ying Fu, David Wipf, and Hua Huang. Single image reflection removal exploiting misaligned training data and network enhancements. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8178–8187, 2019.

[7] Qiang Wen, Yinjie Tan, Jing Qin, Wenxi Liu, Guoqiang Han, and Shengfeng He. Single image reflection removal beyond linearity. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3771–3779, 2019.

[8] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Eur. Conf. Comput. Vis.*, pages 3–19, 2018.

[9] Xuaner Zhang, Ren Ng, and Qifeng Chen. Single image reflection separation with perceptual losses. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4786–4794, 2018.

Block name	Output size	Filter size or Setting
SE-ResBlock [2]		
Conv_1 + PReLU / ReLU	$H \times W \times C$	$3 \times 3, C$, stride=1
Conv_2	$H \times W \times C$	$3 \times 3, C$, stride=1
SE-Layer		
AdaptiveAvgPool2d + Reshape	C	Pooling output size: 1×1
Linear_1 + PReLU / ReLU	C / reduction	$C \times C/2$ (reduction=2)
Linear_2 + Sigmoid	C	$C/2 \times C$ (reduction=2)
Reshape	$1 \times 1 \times C$	None
Multiplication_1	$H \times W \times C$	Input: output of conv_2
Multiplication_2 + Add	$H \times W \times C$	multiply by res_scale (0.1) Input: feature before Conv_1

Table 1. The architecture and detailed parameters of SE-ResBlock [2]. The dimension of the input feature map is $H \times W \times C$, where H and W are the height and width of the feature map, and C is the channel number. The activation function PReLU is used in RDM & TSM modules, while ReLU is used in the Image feature branch. The variable reduction is set to 2 in this table.

Block name	Output size	Filter size or Setting
CBAM-ResBlock [8]		
Conv_1 + ReLU	$H \times W \times C$	$3 \times 3, C$, stride=1
Conv_2	$H \times W \times C$	$3 \times 3, C$, stride=1
CBAM-Layer		
Channel-attention (Shared MLP)		
AdaptiveAvgPool2d	$1 \times 1 \times C$	Input: the output of Conv_2 Pooling output size: 1×1
Conv_2 + ReLU	$1 \times 1 \times (C/2)$	$1 \times 1, C/2$ (reduction=2), stride=1
Conv_3	$1 \times 1 \times C$	$1 \times 1, C$, stride=1
AdaptiveMaxPool2d	$1 \times 1 \times C$	Input: the output of Conv_2 Pooling output size: 1×1
Conv_2 + ReLU	$1 \times 1 \times (C/2)$	$1 \times 1, C/2$ (reduction=2), stride=1
Conv_3	$1 \times 1 \times C$	$1 \times 1, C$, stride=1
Add + Sigmoid_1	$1 \times 1 \times C$	Input: the two outputs of Conv_3
Spatial-attention		
Mean	$H \times W \times 1$	Input: the output of Conv_2 keep_dim=True
Max	$H \times W \times 1$	Input: the output of Conv_2 keep_dim=True
Concat + Conv_4 + Sigmoid_2	$H \times W \times 1$	Input: the output feature of Mean and Max 7×7, 1, stride=1
Multiplication_1	$H \times W \times C$	Input: the output feature of Sigmoid_1 Input: the feature before Conv_1
Multiplication_2	$H \times W \times C$	Input: the output feature of Multiplication_1 Input: the output feature of Sigmoid_2
Add	$H \times W \times C$	Input: the feature before Conv_1

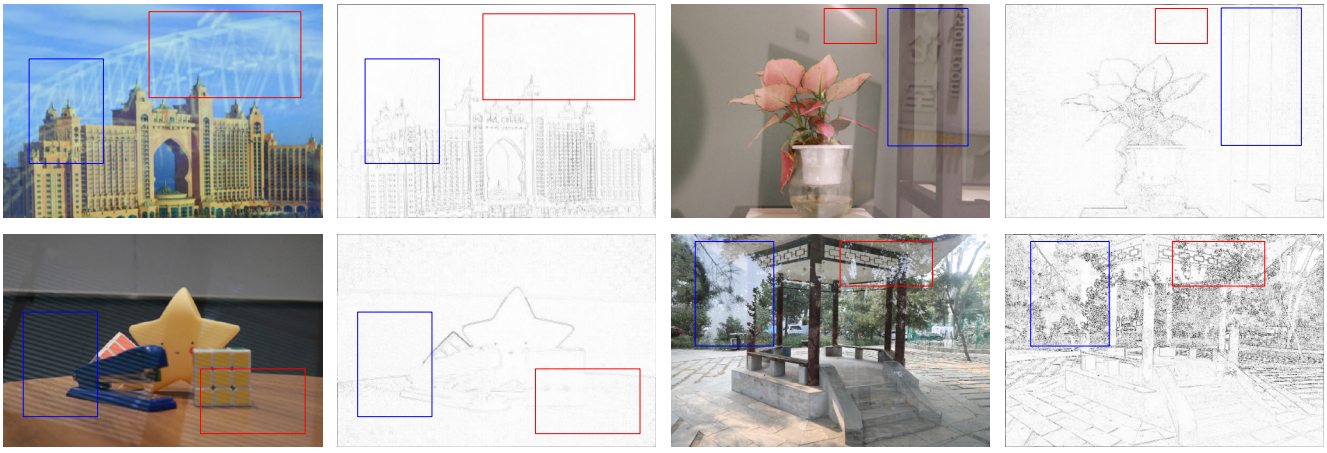
Table 2. The architecture and detailed parameters of CBAM-ResBlock [8]. The dimension of the input feature map is $H \times W \times C$. The variable reduction is set to 2 in this table.

Block name	**Output size**	**Filter size** or **Setting**
Concat	$H \times W \times 6$	Input: the original image $\mathbf{I}$ and the transmission layer $\hat{\mathbf{T}}_{i-1}$
Stage 1:		
Image feature branch:		
Conv_0 + ReLU	$H \times W \times 32$	$3 \times 3, 32$, stride=1
SE-ResBlock (ReLU, reduction=2) $\times 6$, (Tab. 1)		
Reflection detection module (RDM):		
Multi-scale Laplacian submodule (MLSM):		
Laplacian_conv_0 + Concat	$H \times W \times 24$	Input: the Concat results $3 \times 3, 6$, stride=1
Conv_1 + PReLU	$H \times W \times 32$	$3 \times 3, 32$, stride=1
SE-ResBlock (PReLU, reduction=2) $\times 3$, (Tab. 1)		
Conv_2 + ReLU	$H \times W \times 32$	$3 \times 3, 32$, stride=1
Conv_3 + Sigmoid_0	$H \times W \times 1$	$3 \times 3, 1$, stride=1; Output: RCMap $\hat{\mathbf{C}}_i$
Transmission-feature suppression module (TSM):		
SE-ResBlock (PReLU, reduction=2) $\times 3$, (Tab. 1)		
Multiplication	$H \times W \times 32$	Input: $\hat{\mathbf{C}}_i$ from Sigmoid_0
Concat	$H \times W \times 64$	Input: output of Image feature branch
Conv2D LSTM (Input feature size: $H \times W \times 64$, output feature size: $H \times W \times 32$) [1]		
Conv_4 + ReLU	$H \times W \times 32$	$3 \times 3, 32$, stride=1
Conv_5 + ReLU_0	$H \times W \times 3$	$3 \times 3, 3$, stride=1; Output: $\hat{\mathbf{R}}_i$
Stage 2:		
Encoder:		
Concat	$H \times W \times 10$	Input: $\mathbf{I}, \hat{\mathbf{T}}_{i-1}, \hat{\mathbf{R}}_i$ from ReLU_0, $\hat{\mathbf{C}}_i$ from Sigmoid_0
Conv_6 + ReLU	$H \times W \times 64$	$3 \times 3, 64$, stride=1
CBAM-ResBlock (reduction=2) $\times 1$, (Tab. 2)		
Conv_7 + ReLU	$(H/2) \times (W/2) \times 128$	$3 \times 3, 128$, stride=2
Conv_8 + ReLU	$(H/2) \times (W/2) \times 128$	$3 \times 3, 128$, stride=1
CBAM-ResBlock (reduction=4) $\times 2$, (Tab. 2)		
Conv_9 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=2
Conv_10 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1
Conv_11 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1
CBAM-ResBlock (reduction=8) $\times 3$, (Tab. 2)		
diConv_0 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1, dilation=2
diConv_1 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1, dilation=4
diConv_2 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1, dilation=8
diConv_3 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1, dilation=16
Conv_12 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1
Conv_13 + ReLU	$(H/4) \times (W/4) \times 256$	$3 \times 3, 256$, stride=1
Decoder:		
Conv_15 + ReLU	$(H/4) \times (W/4) \times 3$	$3 \times 3, 3$, stride=1; Output: $\hat{\mathbf{T}}_i^{1/4}$
deConv_0 + AvgPool2d + ReLU	$(H/2) \times (W/2) \times 128$	$4 \times 4, 128$, stride=2 Input: the input of Conv_15
Add + Conv_16 + ReLU	$(H/2) \times (W/2) \times 128$	Input: skip connection from the input of Conv_9 $3 \times 3, 128$, stride=1
Conv_17 + ReLU	$(H/2) \times (W/2) \times 3$	$3 \times 3, 3$, stride=1; Output: $\hat{\mathbf{T}}_i^{1/2}$
deConv_1 + AvgPool2d + ReLU	$H \times W \times 64$	$4 \times 4, 64$, stride=2 Input: the input of Conv_17
Add + Conv_18 + ReLU	$H \times W \times 32$	Input: skip connection from the input of Conv_7 $3 \times 3, 32$, stride=1
Conv_19 + ReLU	$H \times W \times 3$	$3 \times 3, 3$, stride=1; Output: $\hat{\mathbf{T}}_i$

Table 3. The architecture and detailed parameters of our network. The size of our network is $10.926M$. The dimension of the input RGB images is denoted as $H \times W \times 3$. Our Laplacian_conv_0 block operates with four scales(original size's 1/1, 1/2, 1/4, 1/8) on the concatenation of images $\{\mathbf{I}, \hat{\mathbf{T}}_{i-1}\}$, and then upsamples their Laplacian maps to the original resolution. The learned kernel weights are described in Tab. 4.

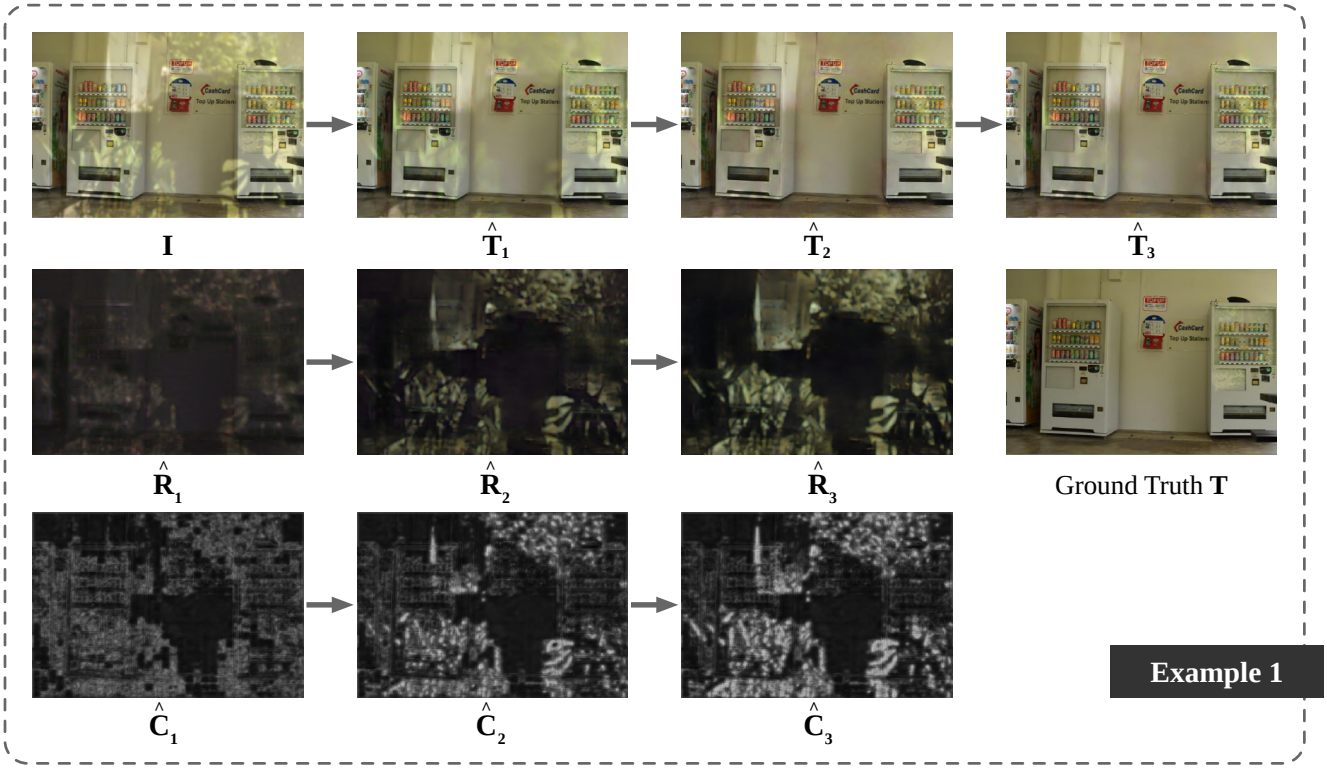
Kernel name	Channels: Num / Index	Kernel weights
Original kernel: k_L		$[0,\ -1,\ 0;\ -1,\ 4,\ -1;\ 0,\ -1,\ 0]$
Fine-tuned Laplacian kernel: $\mathcal{L}ap$	6 / 0	$[1.8113e^{-3},\ -1.0049,\ 6.1352e^{-3};$ $-1.0111,\ 4.0027,\ -1.0060;$ $4.7658e^{-3},\ -1.0013,\ 1.0225e^{-2}]$
	6 / 1	$[9.1783e^{-4},\ -1.0004,\ 6.9816e^{-3};$ $-1.0086,\ 4.0024,\ -9.9966e^{-1};$ $-1.9976e^{-3},\ -1.0023,\ 5.7984e^{-3}]$
	6 / 2	$[-2.4999e^{-3},\ -1.0017,\ 5.3995e^{-3};$ $-1.0121,\ 4.0074,\ -1.0034;$ $2.1355e^{-4},\ -9.9753e^{-1},\ 8.4232e^{-3}]$
	6 / 3	$[2.3978e^{-4},\ -1.0142,\ -4.8809e^{-4};$ $-1.0128,\ 4.0073,\ -1.0161;$ $6.6788e^{-3},\ -1.0092,\ 3.7869e^{-3}]$
	6 / 4	$[2.9756e^{-3},\ -1.0024,\ 1.2700e^{-2};$ $-1.0073,\ 3.9949,\ -1.0033;$ $9.4667e^{-3},\ -1.0018,\ 1.0563e^{-2}]$
	6 / 5	$[3.6218e^{-3},\ -1.0026,\ 8.3107e^{-3};$ $-1.0078,\ 3.9986,\ -1.0030;$ $1.1985e^{-3},\ -1.0036,\ 7.0303e^{-3}]$

Table 4. Fine-tuned Laplacian kernel weights for $\mathcal{L}ap$. The Laplacian kernel weights are shared across scales but fine-tuned separately for R,G,B channels. Since we concatenate the original image and transmission layer as inputs, there are six sets of fine-tuned Laplacian kernel weights. It can be seen that the fine-tuned Laplacian kernel weights at each channel are close to the original Laplacian kernel weights due to the gradient clipping.



Input Inverse Laplacian map Input Inverse Laplacian map

Figure 1. The original input images and their inverse Laplacian maps computed with the fine-tuned Laplacian kernel $\mathcal{L}ap$ in Tab. 4. The inverse Laplacian maps are obtained in the same way as described in the paper. We use the first three channels of $\mathcal{L}ap$ to process the original image $\mathbf{I}$, since the last three channels correspond to the processing of the transmission layer $\hat{\mathbf{T}}_i$. The fine-tuned Laplacian kernel $\mathcal{L}ap$ can also suppress the low-frequency reflection signals.



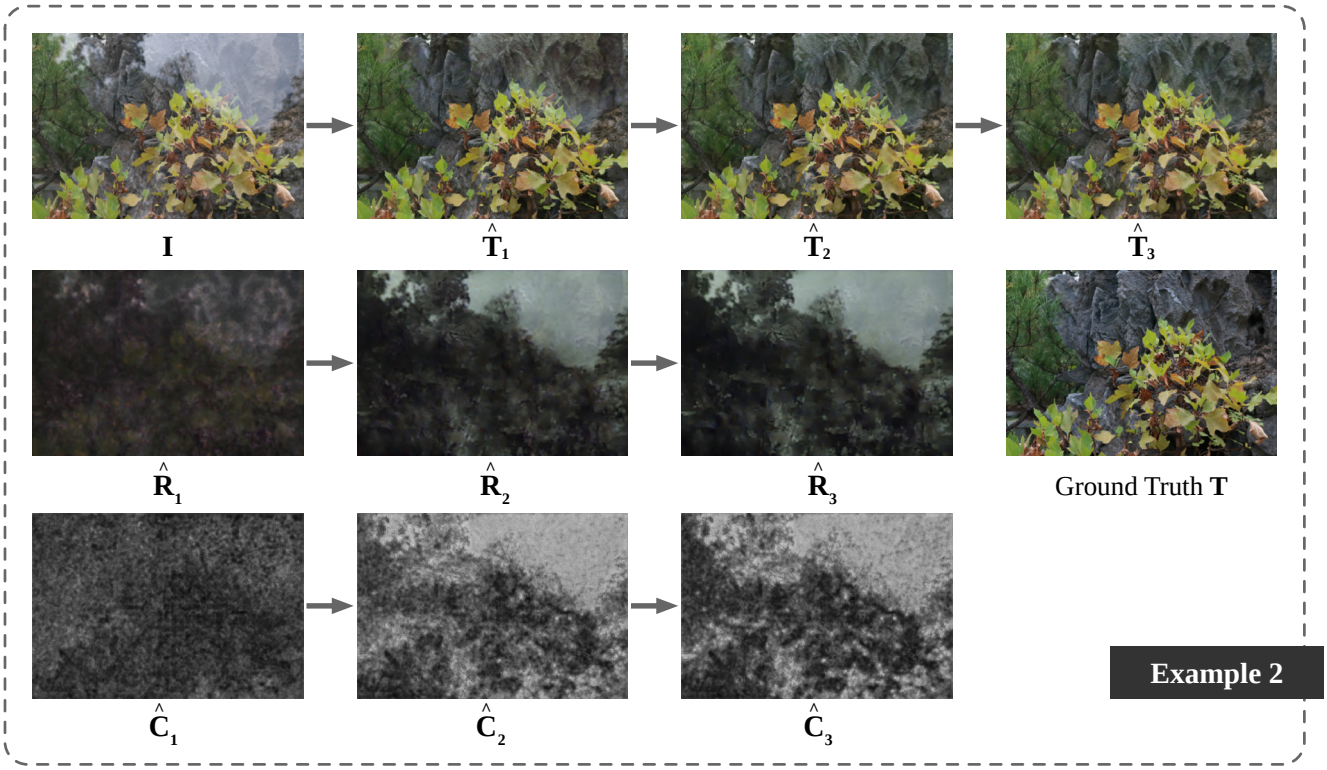


Figure 2. Illustrations of the iterative refinement of transmission layer $\hat{\mathbf{T}}_i$, reflection layer $\hat{\mathbf{R}}_i$ and RCMap $\hat{\mathbf{C}}_i$, Part I. The iteration number is indexed by the subscript i.

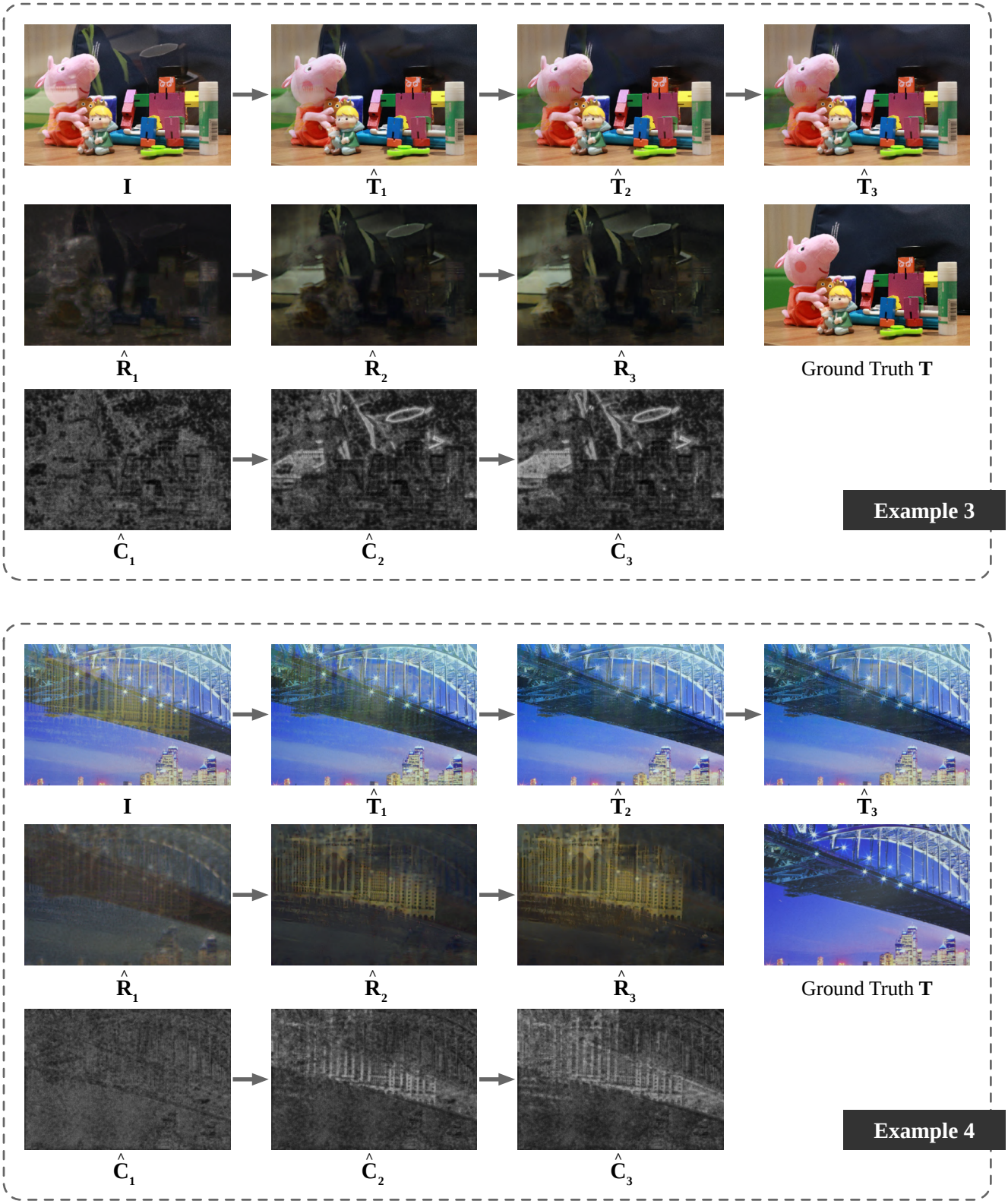


Figure 3. Illustrations of the iterative refinement for transmission layer $\hat{\mathbf{T}}_i$, reflection layer $\hat{\mathbf{R}}_i$ and RCMap $\hat{\mathbf{C}}_i$, Part II. The iteration number is indexed by the subscript i.

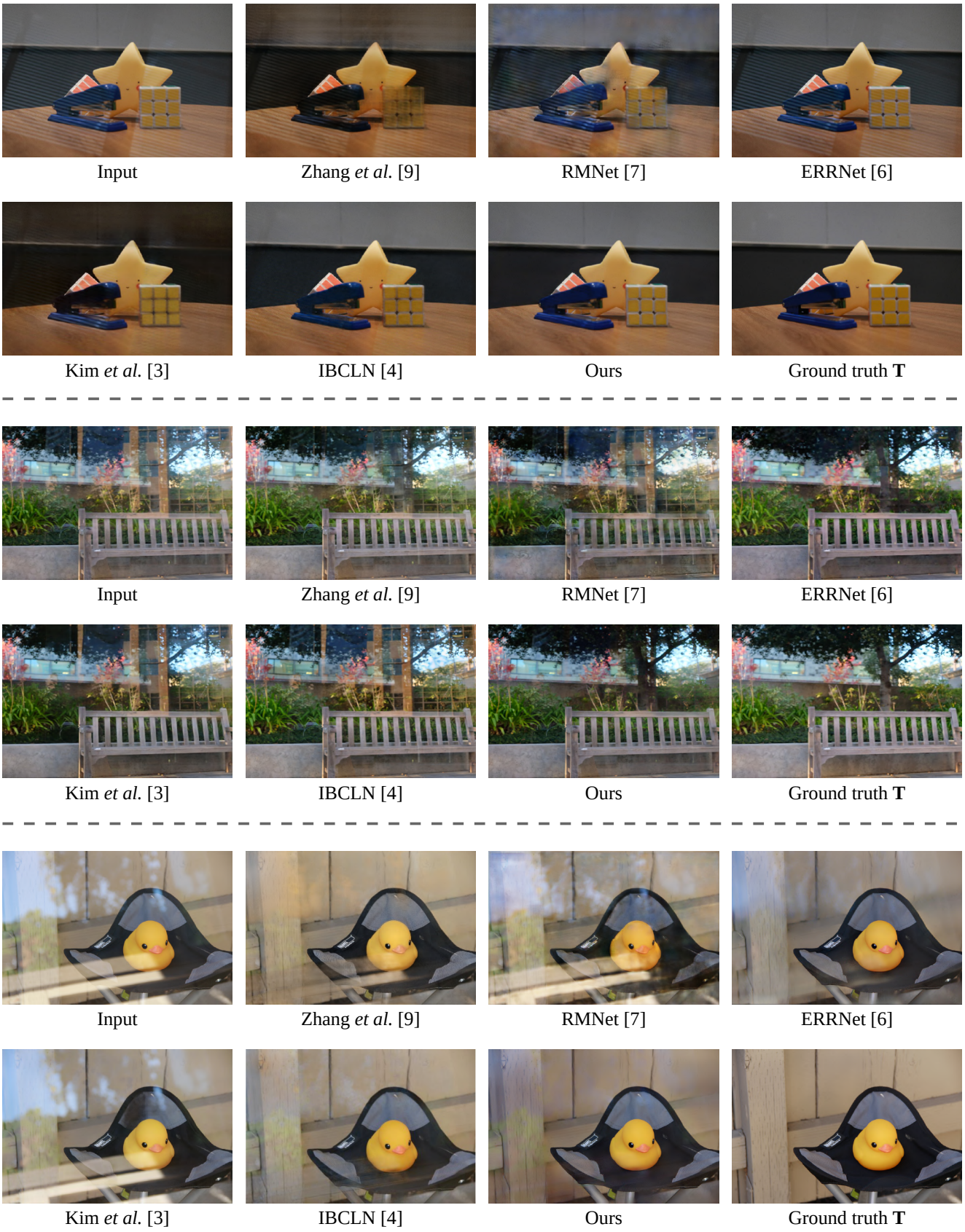


Figure 4. Visual comparisons between our method and five state-of-the-art SIRR methods, Part I.

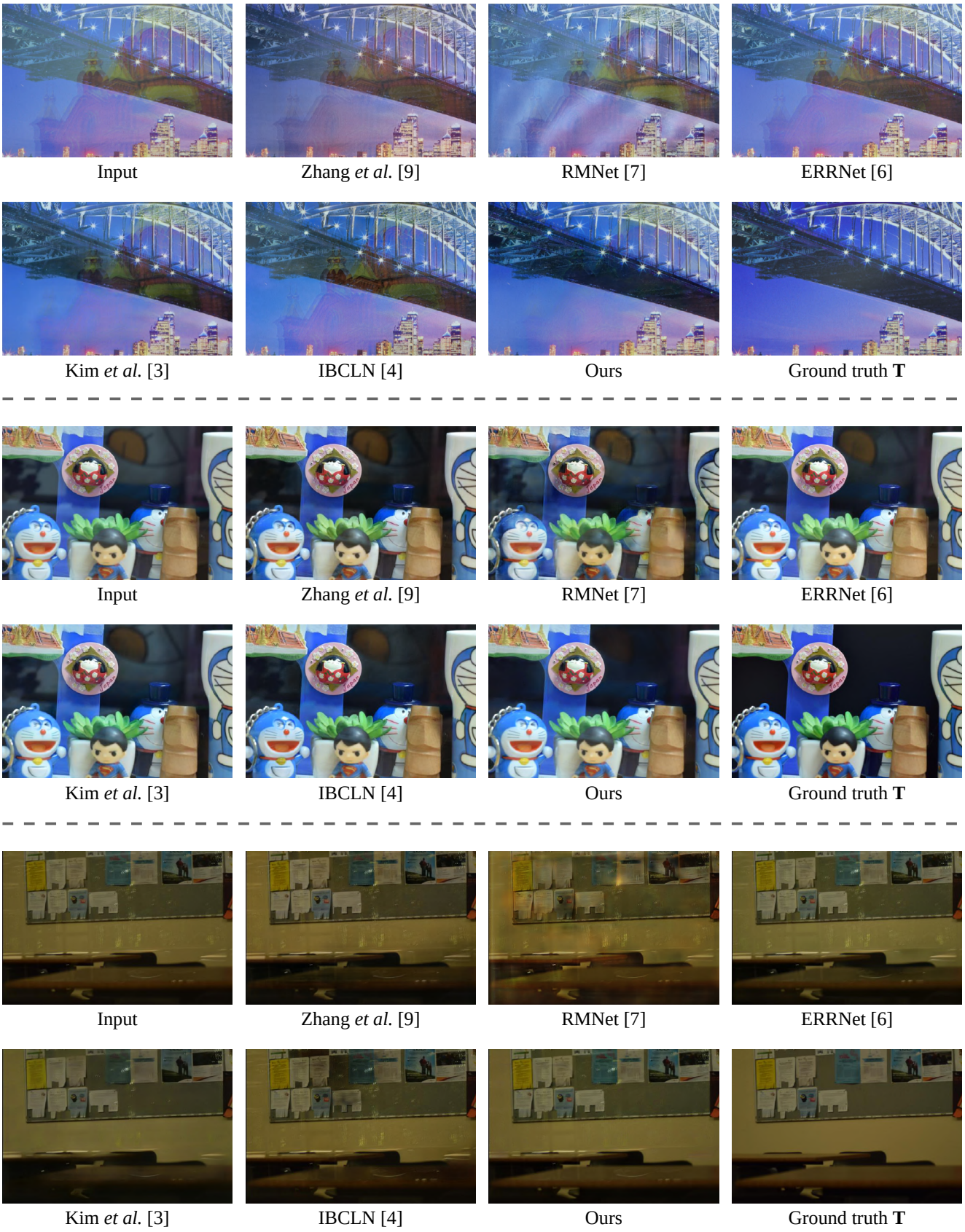


Figure 5. Visual comparisons between our method and five state-of-the-art SIRR methods, Part II.

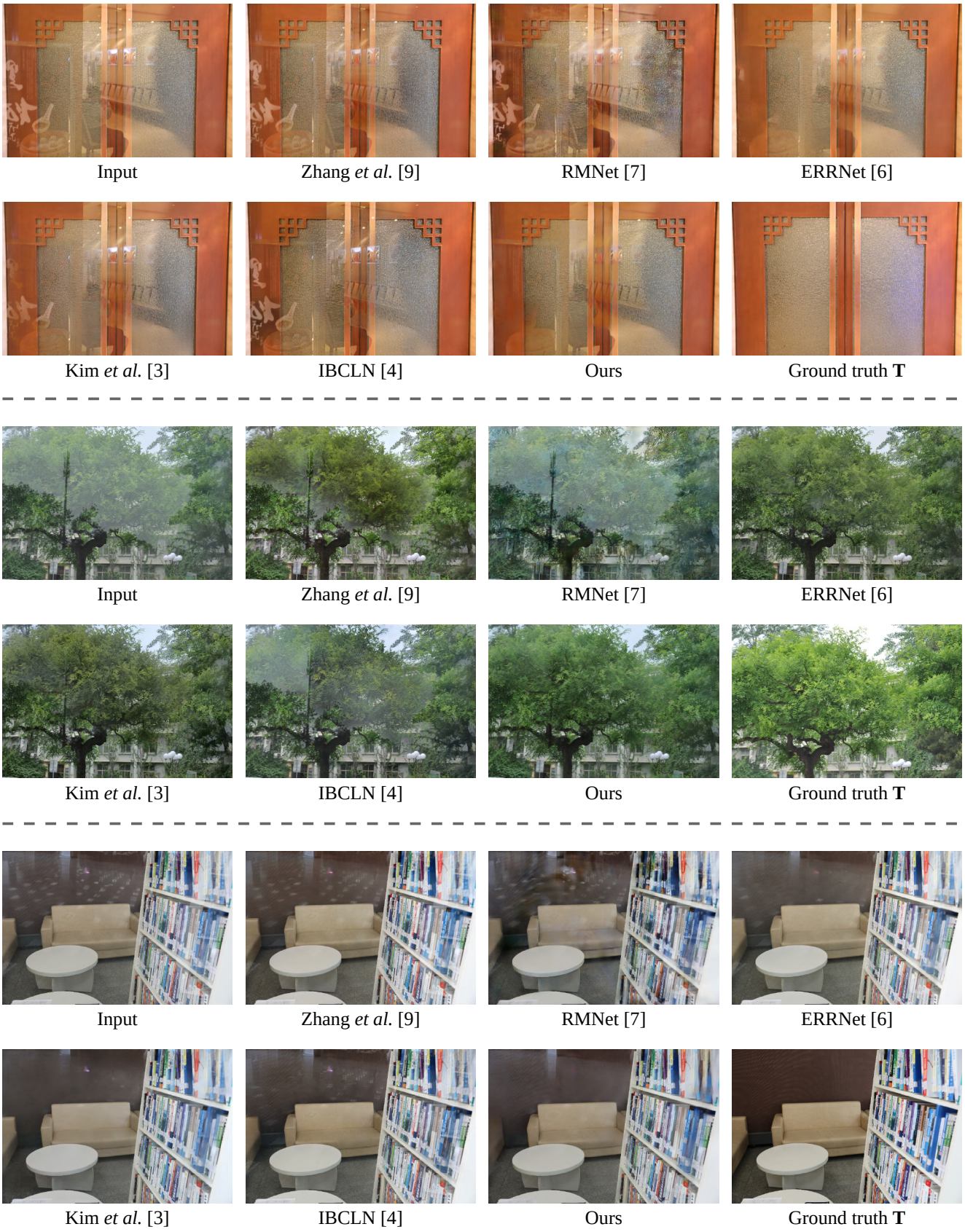


Figure 6. Visual comparisons between our method and five state-of-the-art SIRR methods, Part III.

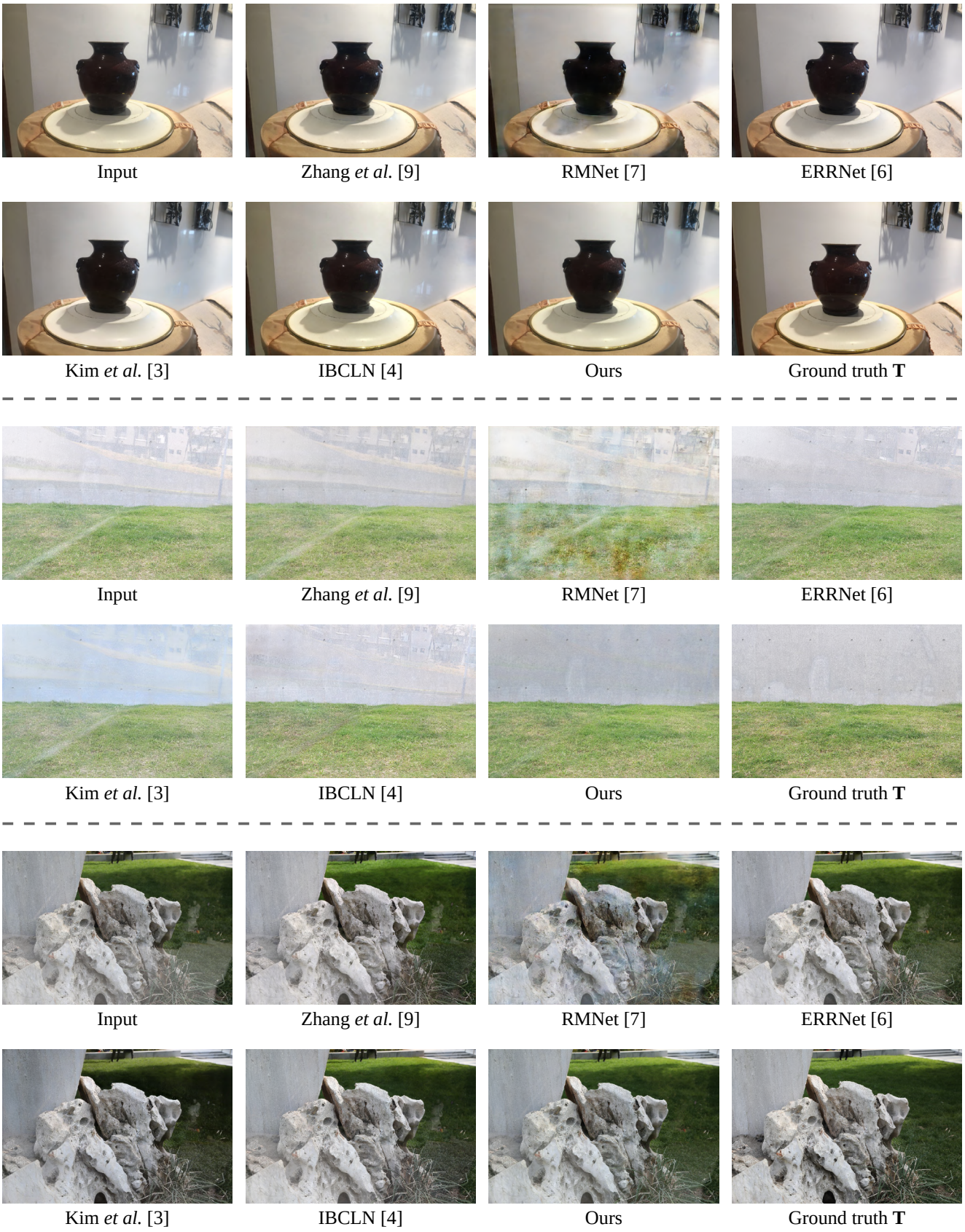


Figure 7. Visual comparisons between our method and five state-of-the-art SIRR methods, Part IV.

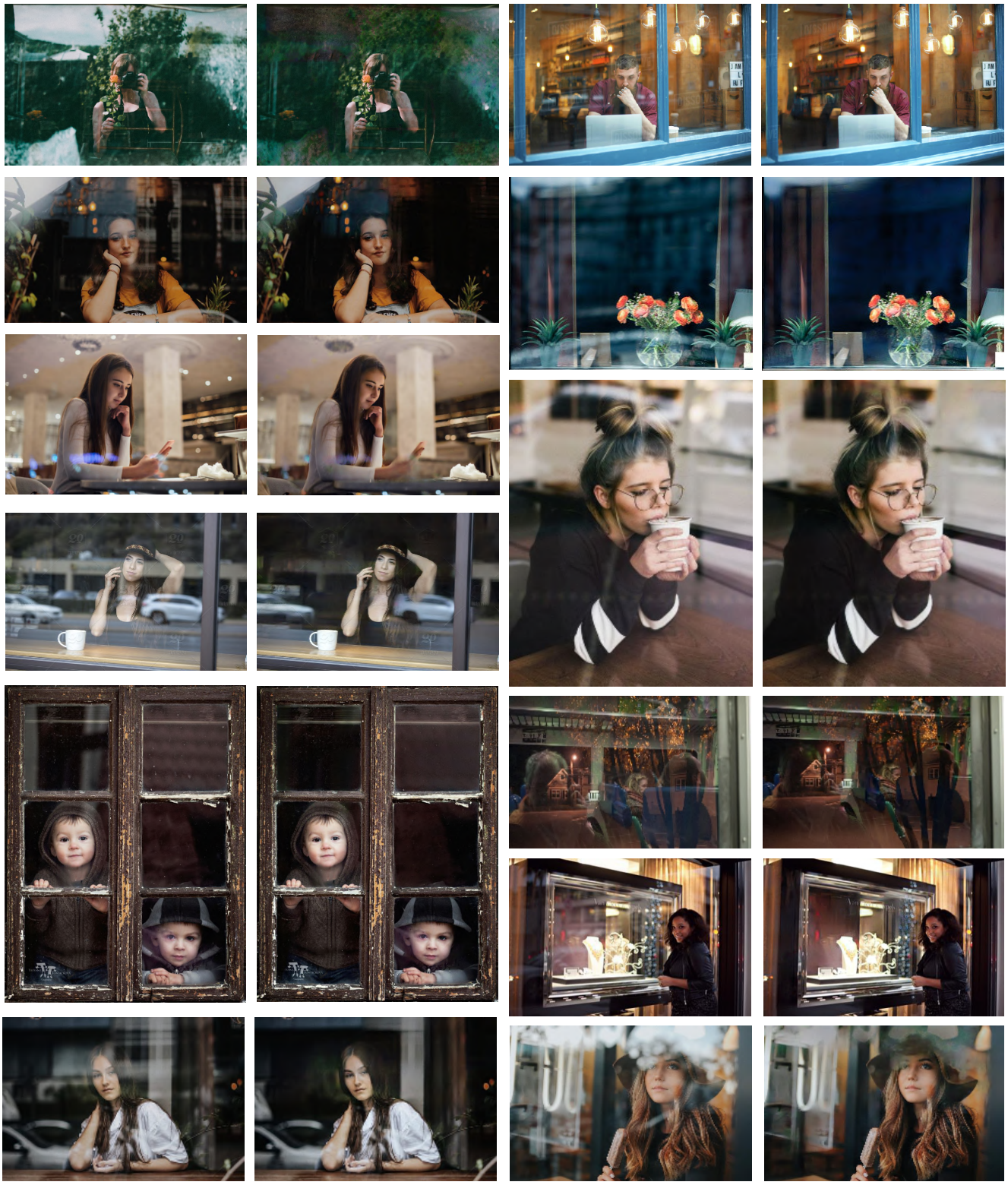


Figure 8. Our SIRR results on Internet images.