Learning Transparent Object Matting
Supplementary Material

Guanying Chen*
The University of Hong Kong
gychen@cs.hku.hk

Kai Han*
University of Oxford
khan@robots.ox.ac.uk

Kwan-Yee K. Wong
The University of Hong Kong
kykwong@cs.hku.hk

Contents

*indicates equal contribution

1. Details of Network Structure

We have proposed a two-stage deep learning framework, called TOM-Net, for learning transparent object matting. The first stage is a multi-scale encoder-decoder network (i.e. CoarseNet) that takes a single image as input and predicts a triplet consisting of an object mask, an attenuation mask and a refractive flow field. Although the estimated object mask in the first stage is robust, the attenuation mask and refractive flow field lack local details. The second stage is a residual network (i.e. RefineNet) that refines the coarse matte to achieve a sharper attenuation mask and a more detailed refractive flow field.

1.1. CoarseNet

CoarseNet is based on the Mirror-Link CNN introduced [7] and we augment it with multi-scale loss [3]. In particular, we use four different scales. The detailed structure is shown in Tab. 1.

Encoder					
layer	**k**	**s**	**chns**	**d-f**	**input**
conv1	3	1	3/16	1	Image
conv1b	3	1	16/16	1	conv1
conv2	3	2	16/16	2	conv1b
conv2b	3	1	16/16	2	conv2
conv3	3	2	16/32	4	conv2b
conv3b	3	1	32/32	4	conv3
conv4	3	2	32/64	8	conv3b
conv4b	3	1	64/64	8	conv4
conv5	3	2	64/128	16	conv4b
conv5b	3	1	128/128	16	conv5
conv6	3	2	128/256	32	conv5b
conv6b	3	1	256/256	32	conv6
conv7	3	2	256/256	64	conv6b
conv7b	3	1	256/256	64	conv7

Decoder					
layer	**k**	**s**	**chns**	**d-f**	**input**
conv_up7_m	3	1	256/256	32	conv7b
conv_up7_a	3	1	256/256	32	conv7b
conv_up7_f	3	1	256/256	32	conv7b
conv_up7=conv_up7_m+conv_up7_a+conv_up7_f					
conv_up6_m	3	1	256/128	16	conv_up7+conv6b
conv_up6_a	3	1	256/128	16	conv_up7+conv6b
conv_up6_f	3	1	256/128	16	conv_up7+conv6b
conv_up6=conv_up6_m+conv_up6_a+conv_up6_f					
conv_up5_m	3	1	128/64	8	conv_up6+conv5b
conv_up5_a	3	1	128/64	8	conv_up6+conv5b
conv_up5_f	3	1	128/64	8	conv_up6+conv5b
conv_up5=conv_up5_m+conv_up5_a+conv_up5_f					
m_4	3	1	128/2	8	conv_up5+conv4b
a_4	3	1	128/1	8	conv_up5+conv4b
f_4	3	1	128/2	8	conv_up5+conv4b
conv_up4_m	3	1	128/32	4	conv_up5+conv4b
conv_up4_a	3	1	128/32	4	conv_up5+conv4b
conv_up4_f	3	1	128/32	4	conv_up5+conv4b
conv_up4=conv_up4_m+conv_up4_a+conv_up4_f					
m_3	3	1	69/2	4	conv_up4+conv3b+(m_4$^{\times 2}$+a_4$^{\times 2}$+f_4$^{\times 2}$)
a_3	3	1	69/1	4	conv_up4+conv3b+(m_4$^{\times 2}$+a_4$^{\times 2}$+f_4$^{\times 2}$)
f_3	3	1	69/2	4	conv_up4+conv3b+(m_4$^{\times 2}$+a_4$^{\times 2}$+f_4$^{\times 2}$)
conv_up3_m	3	1	69/16	2	conv_up4+conv3b+(m_4$^{\times 2}$+a_4$^{\times 2}$+f_4$^{\times 2}$)
conv_up3_a	3	1	69/16	2	conv_up4+conv3b+(m_4$^{\times 2}$+a_4$^{\times 2}$+f_4$^{\times 2}$)
conv_up3_f	3	1	69/16	2	conv_up4+conv3b+(m_4$^{\times 2}$+a_4$^{\times 2}$+f_4$^{\times 2}$)
conv_up3=conv_up3_m+conv_up3_a+conv_up3_f					
m_2	3	1	37/2	2	conv_up3+conv2b+(m_3$^{\times 2}$+a_3$^{\times 2}$+f_3$^{\times 2}$)
a_2	3	1	37/1	2	conv_up3+conv2b+(m_3$^{\times 2}$+a_3$^{\times 2}$+f_3$^{\times 2}$)
f_2	3	1	37/2	2	conv_up3+conv2b+(m_3$^{\times 2}$+a_3$^{\times 2}$+f_3$^{\times 2}$)
conv_up2_m	3	1	37/16	1	conv_up3+conv2b+(m_3$^{\times 2}$+a_3$^{\times 2}$+f_3$^{\times 2}$)
conv_up2_a	3	1	37/16	1	conv_up3+conv2b+(m_3$^{\times 2}$+a_3$^{\times 2}$+f_3$^{\times 2}$)
conv_up2_f	3	1	37/16	1	conv_up3+conv2b+(m_3$^{\times 2}$+a_3$^{\times 2}$+f_3$^{\times 2}$)
conv_up2=conv_up2_m+conv_up2_a+conv_up2_f					
m_1	3	1	37/2	1	conv_up2+conv1b+(m_2$^{\times 2}$+a_2$^{\times 2}$+f_2$^{\times 2}$)
a_1	3	1	37/1	1	conv_up2+conv1b+(m_2$^{\times 2}$+a_2$^{\times 2}$+f_2$^{\times 2}$)
f_1	3	1	37/2	1	conv_up2+conv1b+(m_2$^{\times 2}$+a_2$^{\times 2}$+f_2$^{\times 2}$)

Table 1. Network architecture of CoarseNet. **k** is the kernel size, **s** the stride, **chns** the number of input and output channels for each convolutional layer, **d-f** the down-sampling factor of the output for each layer relative to the input image, and **input** the input of each layer. "+" is a concatenation and "$^{\times 2}$" is a $2\times$ nearest up-sampling operation. All convolutional layers are followed by BatchNorm [5] and ReLU [6] layers, except for the output layers. For decoder, conv_up$n$_m is a convolutional layer in object mask branch followed by Batchnorm, ReLU and nearest up-sampling layers. Similarly, conv_up$n$_a and conv_up$n$_f are the layers in attenuation and refractive flow branch, respectively. m_n, a_n, f_n are the output of the CoarseNet, corresponding to the softmax normalized object mask probability, attenuation mask and refractive flow field of scale n.

1.2. RefineNet

RefineNet is a residual network [4]. For memory and training efficiency, we first use three 2× down-sampling convolutional layers to reduce the spatial size of the input, followed by five residual blocks containing ten convolutional layers. To make the output of the same spatial size with the input, two up-sampling branches, each with three 2× de-convolutional layers, are used to regress a sharper attenuation mask and a more detailed refractive flow field. We implement our residual blocks similar to [2]. Tab. 2 shows the details of the RefineNet.

RefineNet					
layer	**k**	**s**	**chns**	**d-f**	**input**
conv1	9	1	8/64	1	Image+m_1+a_1+f_1
conv2	4	2	64/64	2	conv1
conv3	4	2	64/64	4	conv2
conv4	4	2	64/64	8	conv3
ResBlock1	3	1	64/64	8	conv4
ResBlock2	3	1	64/64	8	ResBlock1
ResBlock3	3	1	64/64	8	ResBlock2
ResBlock4	3	1	64/64	8	ResBlock3
ResBlock5	3	1	64/64	8	ResBlock4
deconv1_a	4	2	64/64	4	ResBlock5
deconv2_a	4	2	64/64	2	deconv1_a
deconv3_a	4	2	64/64	1	deconv2_a
a_refined	3	1	65/1	1	deconv3_a+a_1
deconv1_f	4	2	64/64	4	ResBlock5
deconv2_f	4	2	64/64	2	deconv1_f
deconv3_f	4	2	64/64	1	deconv2_f
f_refined	3	1	66/2	1	deconv3_f+f_1

Table 2. Network architecture of RefineNet. Input for conv1 layer is the concatenation of softmax normalized object mask probability (m_1), attenuation (a_1) and flow field (f_1) from the output of CoarseNet. The final output of RefineNet are the refined attenuation (a_refined) and the refined flow field (f_refined). All convolutional layers are followed by BatchNorm and ReLU, again except for the output layers.

2. More Samples and Results on Synthetic Dataset

As there is no off-the-shelf dataset for learning transparent object matting, and it is tedious and difficult to produce a large-scale real dataset with ground truth matte, we create a large-scale synthetic dataset for training and testing using *POV-Ray* [1]. Here, we provide more samples and results on synthetic data.

2.1. More Synthetic Samples

More samples from the synthetic test dataset are shown in Fig. 1.

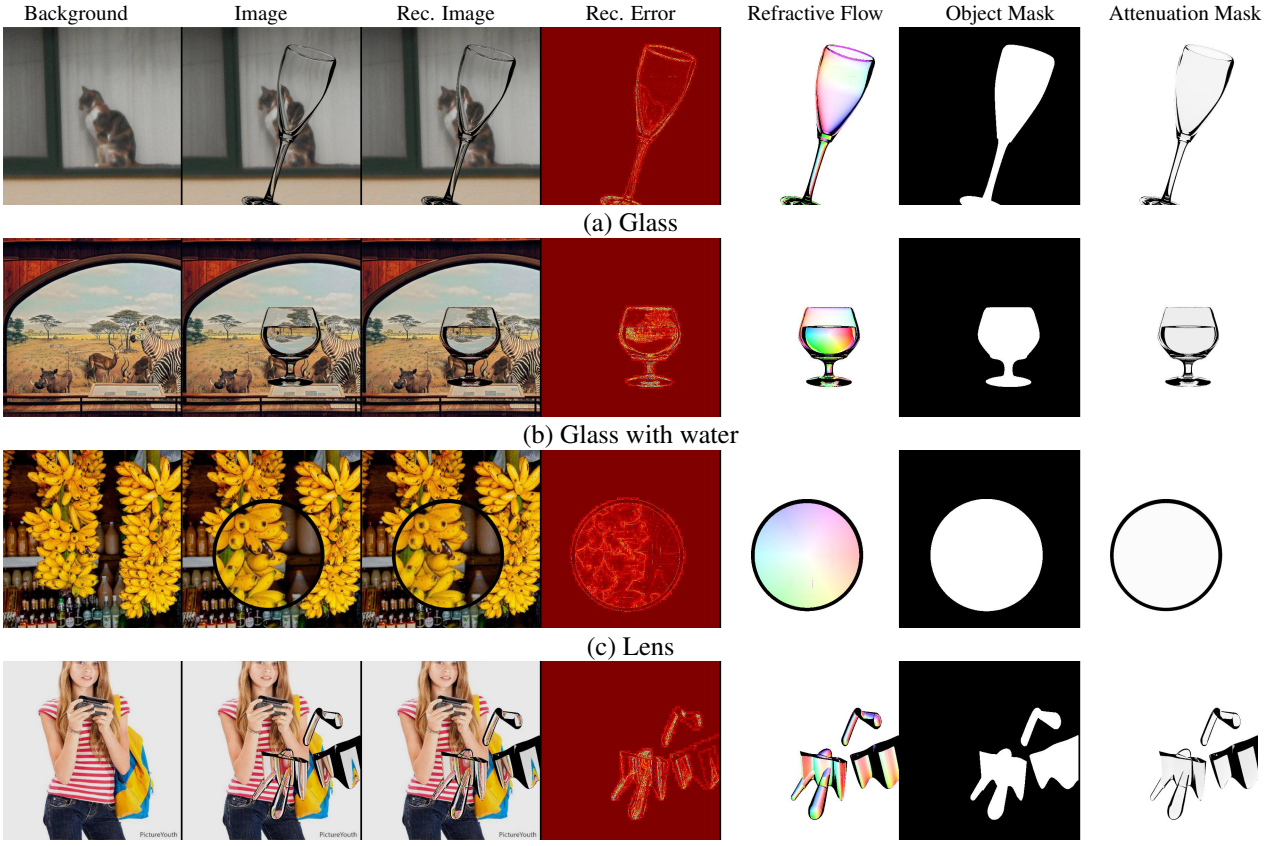


Figure 1. More synthetic samples. The third and fourth columns show the reconstructed images using the ground matte and the reconstruction error, respectively. The reconstruction error is marginal, and the reconstructed images and the original images are visually not distinguishable, illustrating that the ground truth matte is accurate enough to supervise the learning of our deep model.

2.2. More Results on Synthetic Dataset

More results on the synthetic test dataset are shown in Fig. 2 (a-f). Fig. 2 (g & h) show results on the transparent Stanford Bunny and Dragon, which are rendered using the public available 3D models.

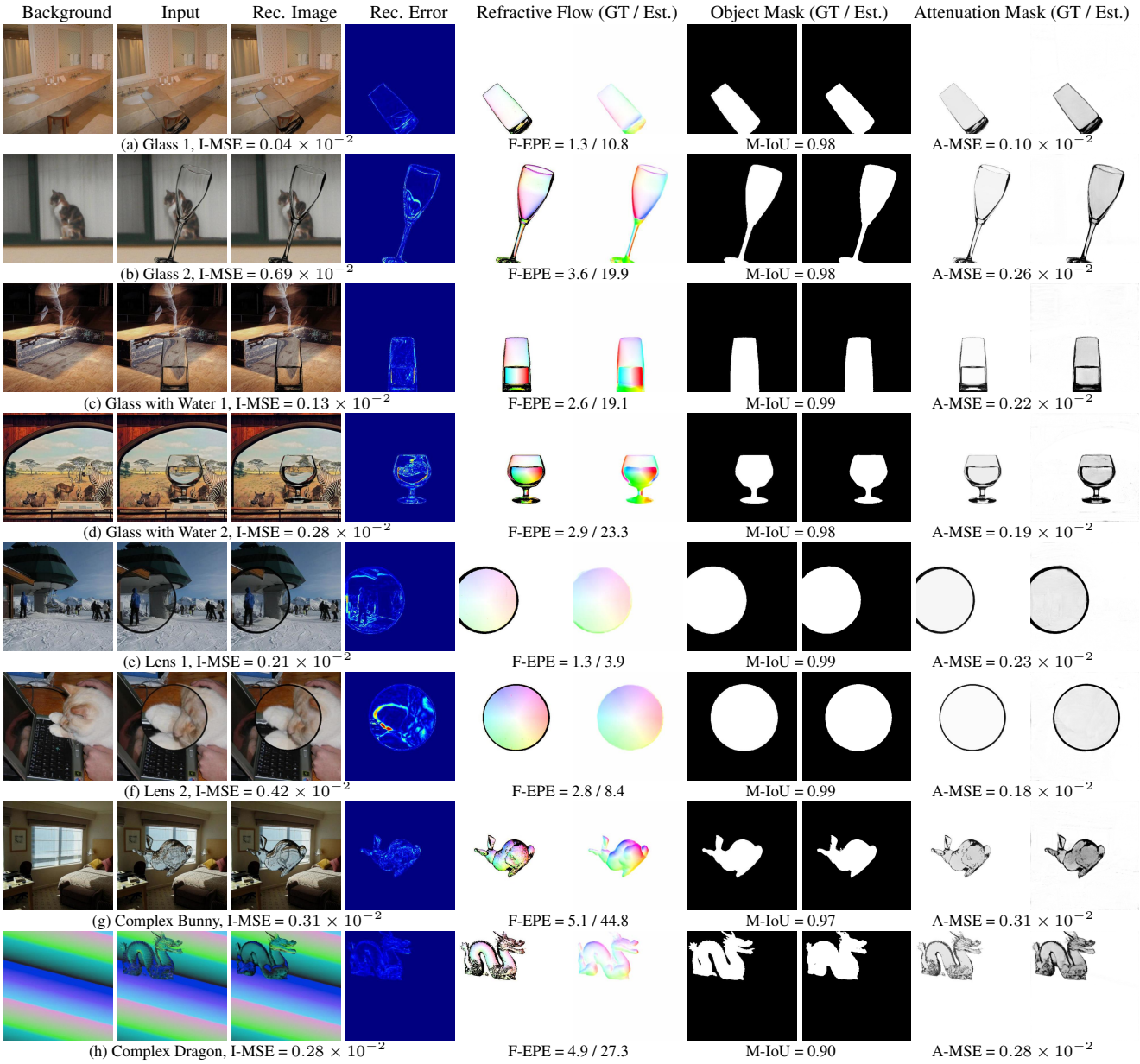
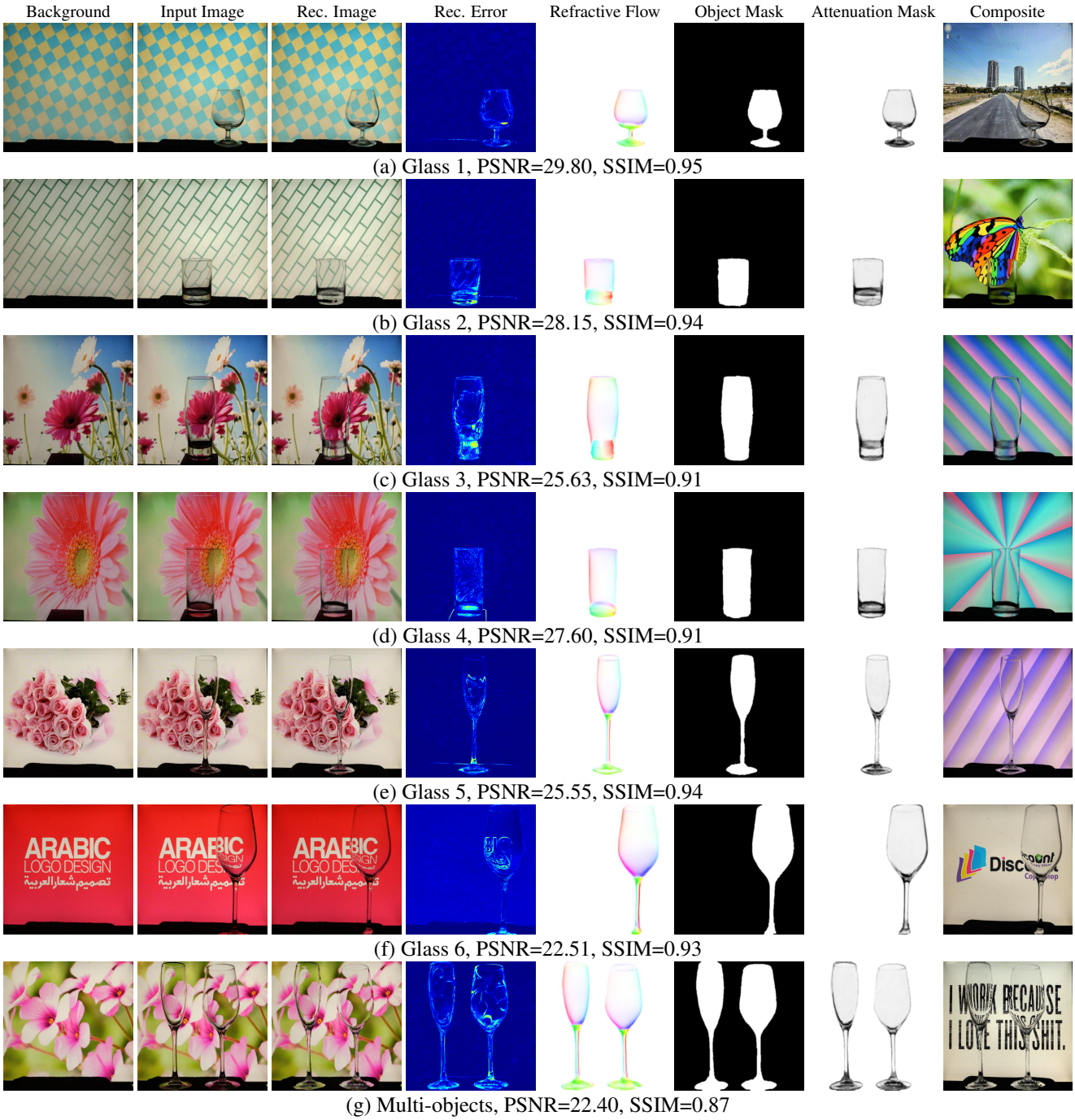


Figure 2. More results on the synthetic data. Quantitative errors are shown below the images. It is worth to note that after training on the combination of multiple basic shapes, our model generalizes well to other more complicated shapes (e.g. Stanford Bunny (g) and Dragon (h)).

3. More Results on Real Dataset

3.1. More Results on Glass

Fig. 3 shows results for different glass shapes in front of different backgrounds, demonstrating the robustness of our model in handling different kinds of glass and different backgrounds.



(a) Glass 1, PSNR=29.80, SSIM=0.95

(b) Glass 2, PSNR=28.15, SSIM=0.94

(c) Glass 3, PSNR=25.63, SSIM=0.91

(d) Glass 4, PSNR=27.60, SSIM=0.91

(e) Glass 5, PSNR=25.55, SSIM=0.94

(f) Glass 6, PSNR=22.51, SSIM=0.93

(g) Multi-objects, PSNR=22.40, SSIM=0.87

Figure 3. More results on real glass samples.

3.2. More Results on Glass with Different Poses

Fig. 4 shows results on a glass under seven largely different poses. Our model can produce reliable mattes for all these poses, demonstrating the robustness of TOM-Net to pose variation.

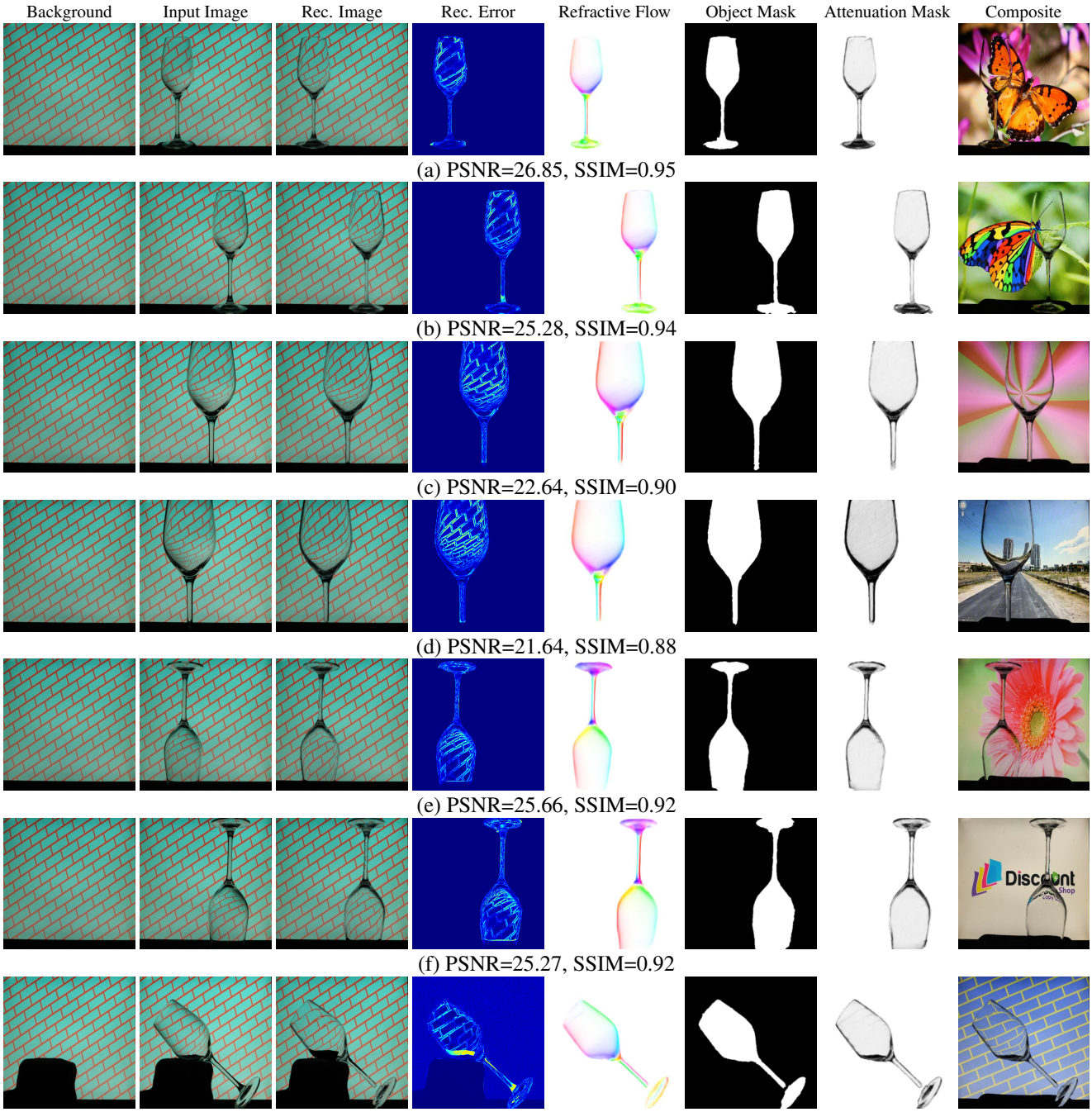
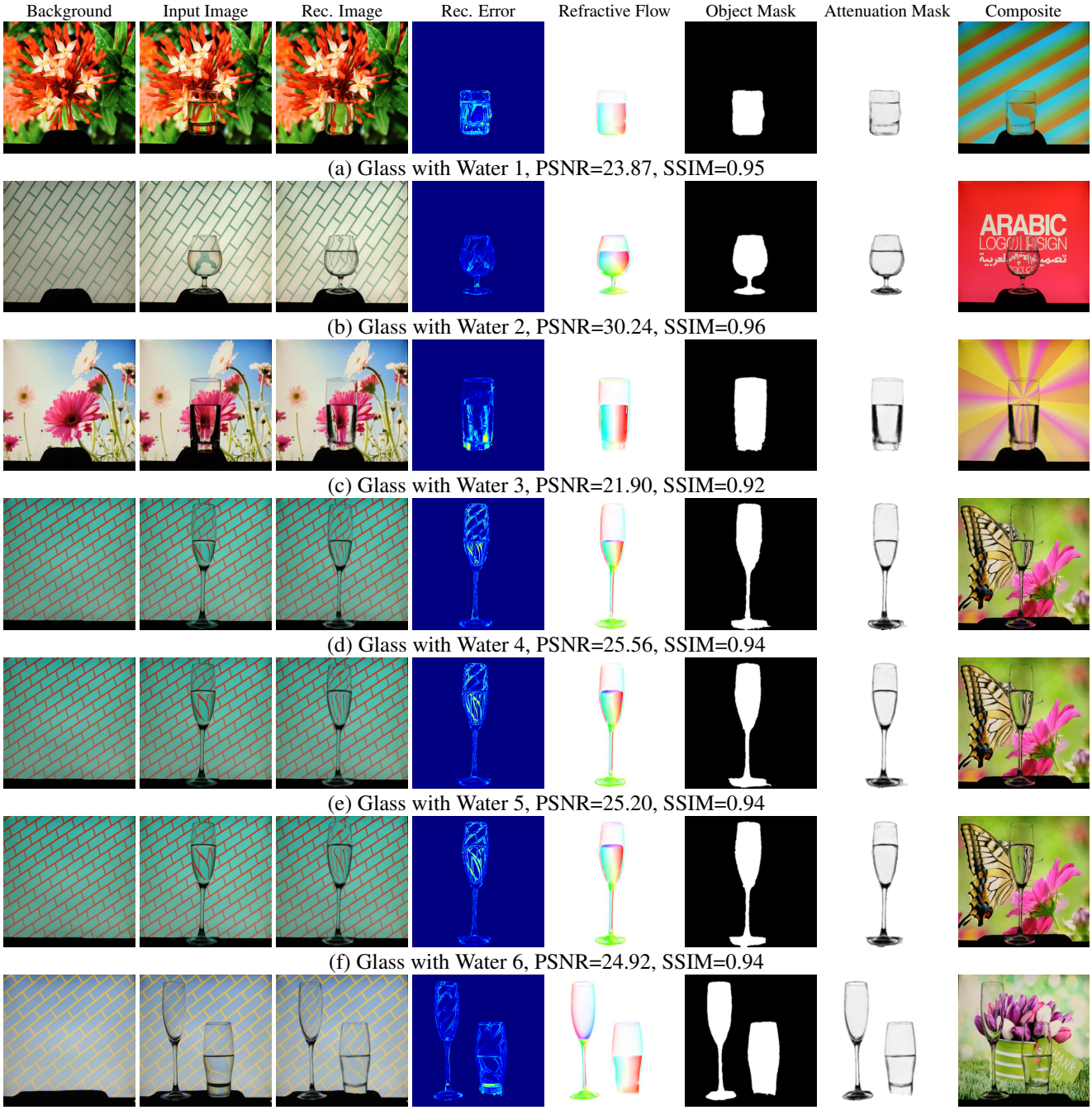


Figure 4. More results on glass under different poses.

3.3. More Results on Glass with Water

Fig. 5 shows results for different glasses with different amount of water.



(a) Glass with Water 1, PSNR=23.87, SSIM=0.95

(b) Glass with Water 2, PSNR=30.24, SSIM=0.96

(c) Glass with Water 3, PSNR=21.90, SSIM=0.92

(d) Glass with Water 4, PSNR=25.56, SSIM=0.94

(e) Glass with Water 5, PSNR=25.20, SSIM=0.94

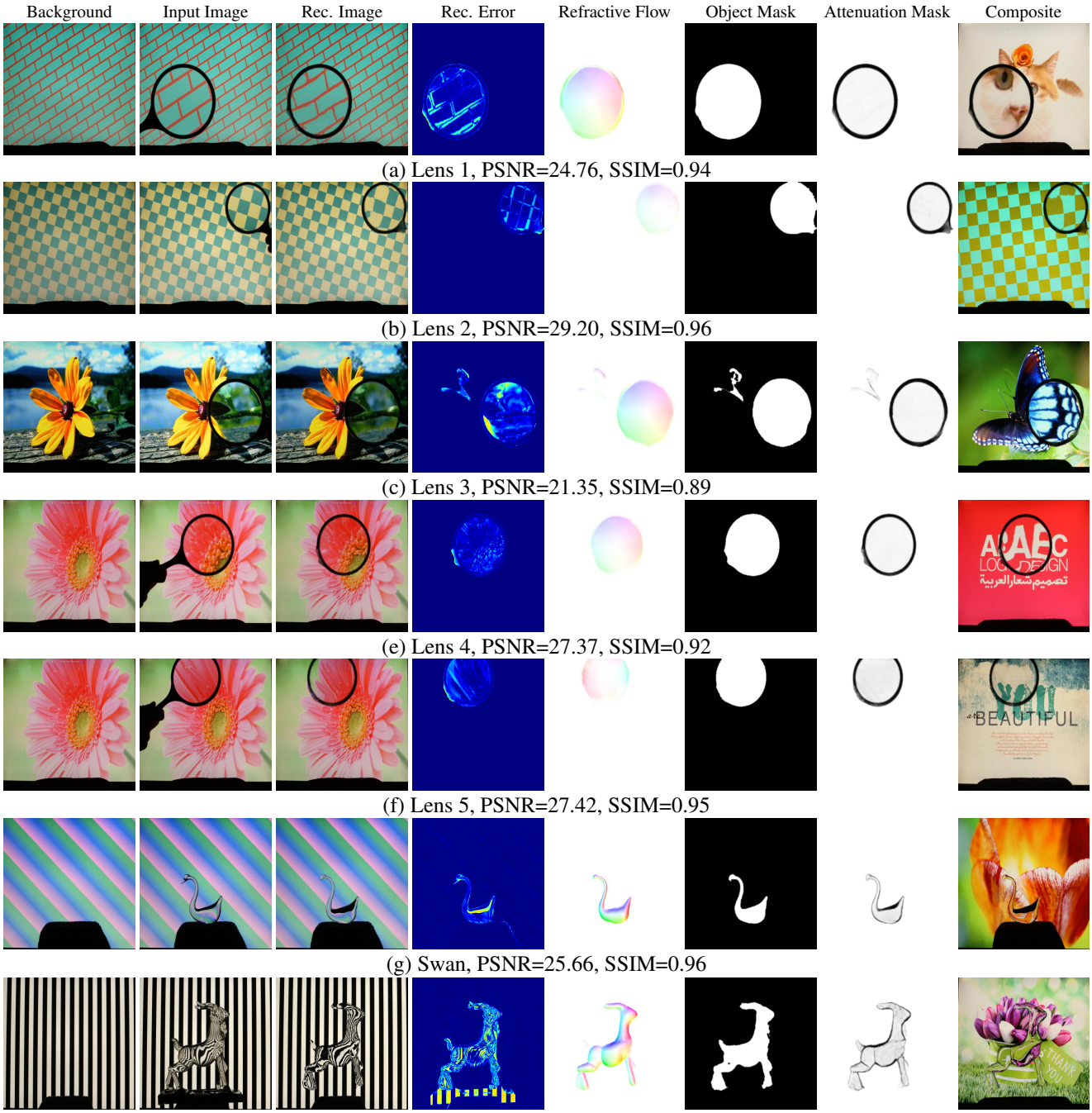
(f) Glass with Water 6, PSNR=24.92, SSIM=0.94

(g) Multi-objects, PSNR=24.87, SSIM=0.93

Figure 5. More results on real Glass with Water. Our model can clearly recognize the water level in different glasses (a, b, c). It can robustly handle glass with different amount of water (d, e, f). (g) shows an challenging example of two glasses with one containing water and the other empty, which can also be handled by TOM-Net.

3.4. More Results on Lens and Complex Shape

Fig. 6 (a-f) shows results on lens at different position in front of different backgrounds, and Fig. 6 (g & h) shows results on the complex swan and sheep.



(a) Lens 1, PSNR=24.76, SSIM=0.94

(b) Lens 2, PSNR=29.20, SSIM=0.96

(c) Lens 3, PSNR=21.35, SSIM=0.89

(e) Lens 4, PSNR=27.37, SSIM=0.92

(f) Lens 5, PSNR=27.42, SSIM=0.95

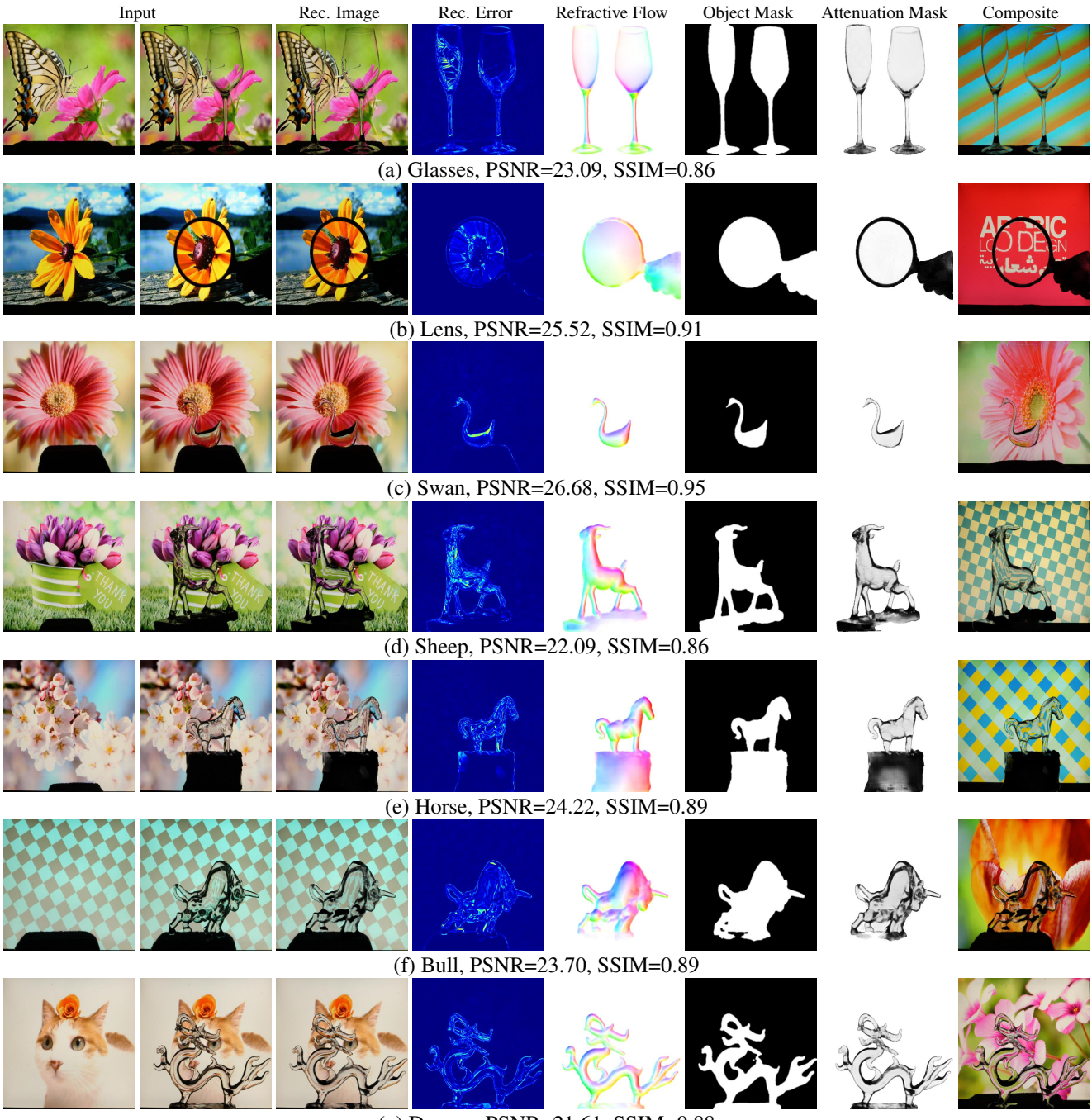
(g) Swan, PSNR=25.66, SSIM=0.96

(h) Sheep. PSNR=14.88, SSIM=0.80

Figure 6. More results on real Lens and Complex Shape.

4. More results of TOM-Net^{+Bg} on Real Dataset

We have investigated how the performance of our method can be improved when the background image is also available. More results of TOM-Net^{+Bg} on real dataset are shown in Fig. 7.



(a) Glasses, PSNR=23.09, SSIM=0.86

(b) Lens, PSNR=25.52, SSIM=0.91

(c) Swan, PSNR=26.68, SSIM=0.95

(d) Sheep, PSNR=22.09, SSIM=0.86

(e) Horse, PSNR=24.22, SSIM=0.89

(f) Bull, PSNR=23.70, SSIM=0.89

(g) Dragon, PSNR=21.61, SSIM=0.88

Figure 7. More results of TOM-Net^{+Bg} on real dataset.

References

[1] Persistence of vision (tm) raytracer. http://www.povray.org/. 4

[2] Q. Fan, J. Yang, G. Hua, B. Chen, and D. Wipf. A generic deep architecture for single image reflection removal and image smoothing. In *Proc. Int. Conf. Comp. Vision*, 2017. 3

[3] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *Proc. Int. Conf. Comp. Vision*, 2015. 2

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, 2016. 3

[5] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. Int. Conf. on Machine Learning*, 2015. 2

[6] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. Int. Conf. on Machine Learning*, 2010. 2

[7] J. Shi, Y. Dong, H. Su, and S. X. Yu. Learning non-lambertian object intrinsics across shapenet categories. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, 2017. 2