Real-Time High-Resolution Background Matting Supplementary Material

Shanchuan Lin* Andrey Ryabtsev* Soumyadip Sengupta
Brian Curless Steve Seitz Ira Kemelmacher-Shlizerman
University of Washington
{linsh,ryabtsev,soumya91,curless,seitz,kemelmi}@cs.washington.edu

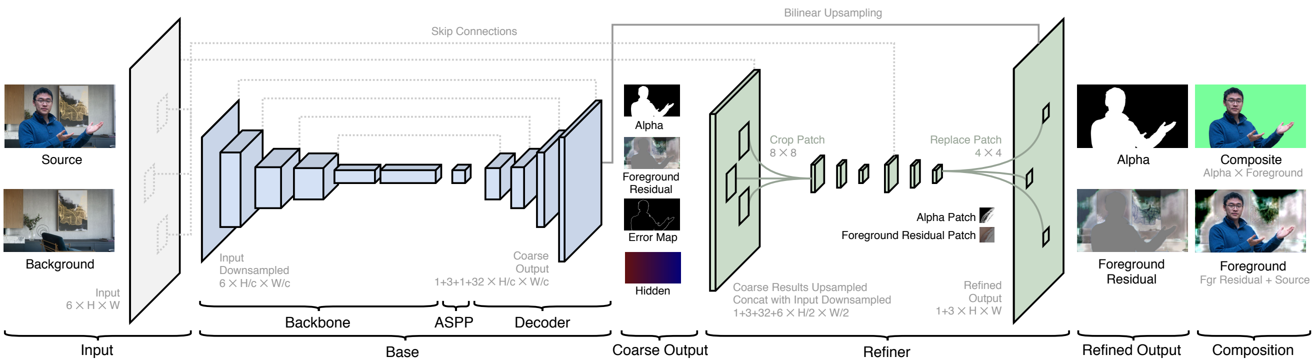

Figure 1: Network architecture. The diagram is repeated in supplementary for clarity. G_{base} (blue) operates on the downsampled input to produce coarse-grained results and an error prediction map. G_{refine} (green) selects error-prone patches and refines them to the full resolution.

A. Overview

We provide additional details in this supplementary. In Sec. B, we describe the details of our network architecture and implementation. In Sec. C, we clarify our use of keywords for crawling background images. In Sec. D, we explain how we train our model and show details of our data augmentations. In Sec. E, we show additional metrics about our method's performance. In Sec. F, we show all the qualitative results used in our user study along with the average score per sample.

B. Network

B.1. Architecture

Backbone Both ResNet and MobileNetV2 are adopted from the original implementation with minor modifications. We change the first convolution layer to accept 6 channels for both the input and the background images. We follow DeepLabV3's approach and change the last downsampling block with dilated convolutions to maintain an output stride of 16. We do not use the multi-grid dilation technique proposed in DeepLabV3 for simplicity.

ASPP We follow the original implementation of ASPP module proposed in DeepLabV3. Our experiment suggests that setting dilation rates to (3, 6, 9) produces the better results.

*Equal contribution.

Decoder

CBR128 - CBR64 - CBR48 - C37

"CBRk" denotes k 3×3 convolution filters with same padding without bias followed by Batch Normalization and ReLU. "Ck" denotes k 3×3 convolution filters with same padding and bias. Before every convolution, decoder uses bilinear upsampling with a scale factor of 2 and concatenates with the corresponding skip connection from the backbone. The 37-channel output consists of 1 channel of alpha α_c, 3 channels of foreground residual F^R_c, 1 channel of error map E_c, and 32 channels of hidden features H_c. We clamp α_c and E_c to 0 and 1. We apply ReLU on H_c.

Refiner

First stage: C*BR24 - C*BR16
Second stage: C*BR12 - C*4

"C*BRk" and "C*k" follow the same definition above except that the convolution does not use padding.

Refiner first resamples coarse outputs α_c, F^R_c, H_c, and input images I, B to $\frac{1}{2}$ resolution and concatenates them as $[n \times 42 \times \frac{h}{2} \times \frac{w}{2}]$ features. Based on the error prediction E_c, we crop out top k most error-prone patches $[nk \times 42 \times 8 \times 8]$. After applying the first stage, the patch dimension becomes $[nk \times 16 \times 4 \times 4]$. We upsample the patches with nearest upsampling and concatenate them with patches at the corresponding location from I and B to form $[nk \times 22 \times 8 \times 8]$ features. After the second stage, the patch dimension becomes $[nk \times 4 \times 4 \times 4]$. The 4 channels are alpha and foreground residual. Finally, we bilinearly upsample the coarse α_c and F_c^R to full resolution and replace the refined patches to their corresponding location to form the final output α and F^R.

B.2. Implementation

We implement our network in PyTorch [1]. The patch extraction and replacement can be achieved via the native vectorized operations for maximum performance. We find that PyTorch's nearest upsampling operation is much faster on small-resolution patches than bilinear upsampling, so we use it when upsampling the patches.

C. Dataset

VideoMatte240K The dataset contains 484 video clips, which consists a total of 240,709 frames. The average frames per clip is 497.3 and the median is 458.5. The longest clip has 1500 frames while the shortest clip has 124 frames. Figure 2 shows more examples from VideoMatte240K dataset.

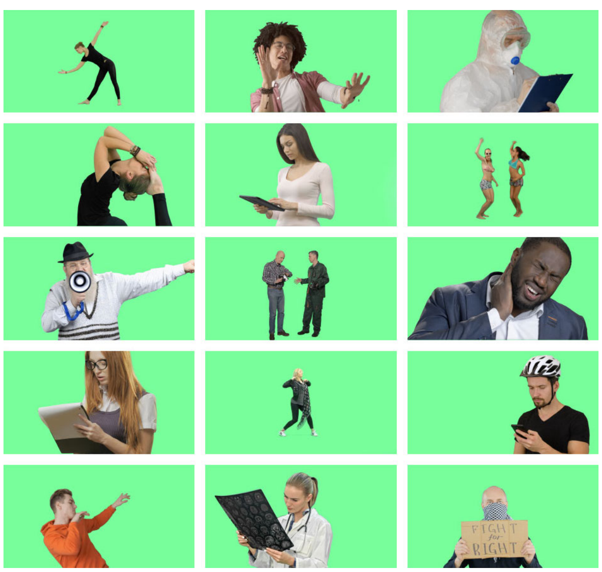

Figure 2: More examples from VideoMatte240K dataset.

Background The keywords we use for crawling background images are:

airport interior	attic	bar interior
bathroom	beach	city
church interior	classroom interior	empty city
forest	garage interior	gym interior
house outdoor	interior	kitchen
lab interior	landscape	lecture hall
mall interior	night club interior	office
rainy woods	rooftop	stadium interior
theater interior	train station	warehouse interior
	workplace interior	

D. Training

Table 1 records the training order, epochs, and hours of our final model on different datasets. We use 1×RTX 2080 TI when training only the base network and 2×RTX 2080 TI when training the network jointly.

Dataset	Network	Epochs	Hours
VideoMatte240K	G_{base}	8	24
VideoMatte240K	$G_{\text{base}} + G_{\text{refine}}$	1	12
PhotoMatte13K	$G_{\text{base}} + G_{\text{refine}}$	25	8
Distinctions	$G_{\text{base}} + G_{\text{refine}}$	30	8

Table 1: Training epochs and hours on different datasets. Time measured on model with ResNet-50 backbone.

Additionally, we use mixed precision training for faster computation and less memory consumption. When using multiple GPUs, we apply data parallelism to split the mini-batch across multiple GPUs and switch to use PyTorch's Synchronized Batch Normalization to track batch statistics across GPUs.

D.1. Training augmentation

For every alpha and foreground training sample, we rotate to composite with backgrounds in a "zip" fashion to form a single epoch. For example, if there are 60 training samples and 100 background images, a single epoch is 100 images, where the 60 samples first pair with the first 60 background images, then the first 40 samples pair with the rest of the 40 background images again. The rotation stops when one set of images runs out. Because the datasets we use are very different in sizes, this strategy is used to generalize the concept of an epoch.

We apply random rotation (±5deg), scale (0.3∼1), translation (±10%), shearing (±5deg), brightness (0.85∼1.15), contrast (0.85∼1.15), saturation (0.85∼1.15), hue (±0.05), gaussian noise (σ^2 ≤0.03), box blurring, and sharpening independently to foreground and background on every sample. We then composite the input image using $I = \alpha F + (1 - \alpha)B$.

We additionally apply random rotation (±1deg), translation (±1%), brightness (0.82∼1.18), contrast (0.82∼1.18), saturation (0.82∼1.18), and hue (±0.1) only on the background 30% of the time. This small misalignment between input I and background B increases model's robustness on real-life captures.

We also find creating artificial shadows increases model's robustness because subjects in real-life often cast shadows on the environment. Shadows are created on I by darkening some areas of the image behind the subject following the subject's contour 30% of the time. Examples of composited images are shown in Figure 3. The bottom row shows examples of shadow augmentation.

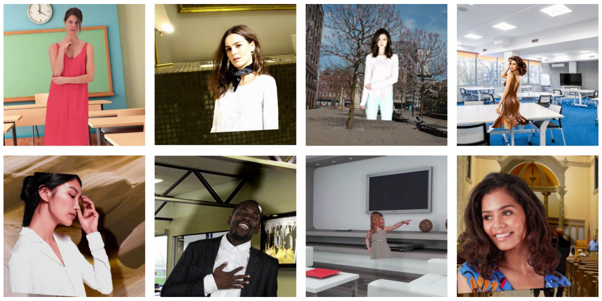

Figure 3: Training samples with augmentations. Bottom row are samples with shadow augmentation. Actual samples have different resolutions and aspect ratios.

D.2. Testing augmentation

For AIM and Distinctions, which have 11 human test samples each, we pair every sample with 5 random backgrounds from the background test set. For PhotoMatte85, which has 85 test samples, we pair every sample with only 1 background. We use the method and metrics described in [2] to evaluate the resulting sets of 55, 55, and 85 images.

We apply a random subpixel translation (±0.3 pixels), random gamma (0.85∼1.15), and gaussian noise ($\mu = \pm 0.02$, $0.08 \leq \sigma^2 \leq 0.15$) to background B only, to simulate misalignment.

The trimaps used as input for trimap-based methods and for defining the error metric regions are obtained by thresholding the grouth-truth alpha between 0.06 and 0.96, then applying 10 iterations of dilation followed by 10 iterations of erosion using a 3×3 circular kernel.

E. Performance

Table 2 shows the performance of our method on two Nvidia RTX 2000 series GPUs: the flagship RTX 2080 TI and the entry-level RTX 2060 Super. The entry-level GPU yields lower FPS but is still within an acceptable range for many real-time applications. Additionally, Table 3 shows that switching to a larger batch size and a lower precision can increase the FPS significantly.

F. Additional Results

In Figures 4, 5, 6, we show all 34 examples in the user study, along with their average rating and results by different methods. Figure 7 shows the web UI for our user-study.

References

[1] Adam Paszke, S. Gross, Francisco Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, Alban Desmaison, Andreas Köpf, E. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, B. Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *ArXiv*, abs/1912.01703, 2019. 2

GPU	Backbone	Reso	FPS
RTX 2080 TI	ResNet-50	HD	60.0
		4K	33.2
	MobileNetV2	HD	100.6
		4K	45.4
RTX 2060 Super	ResNet-50	HD	42.8
		4K	23.3
	MobileNetV2	HD	75.6
		4K	31.3

Table 2: Performance on different GPUs. Measured with batch size 1 and FP32 precision.

Backbone	Reso	Batch	Precision	FPS
MobileNetV2	HD	1	FP32	100.6
		8	FP32	138.4
		8	FP16	200.0
	4K	8	FP16	64.2

Table 3: Performance using different batch sizes and precisions. Measured on RTX 2080 TI.

[2] Christoph Rhemann, Carsten Rother, Jue Wang, Margrit Gelautz, Pushmeet Kohli, and Pamela Rott. A perceptually motivated online benchmark for image matting. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1826–1833. IEEE, 2009. 3

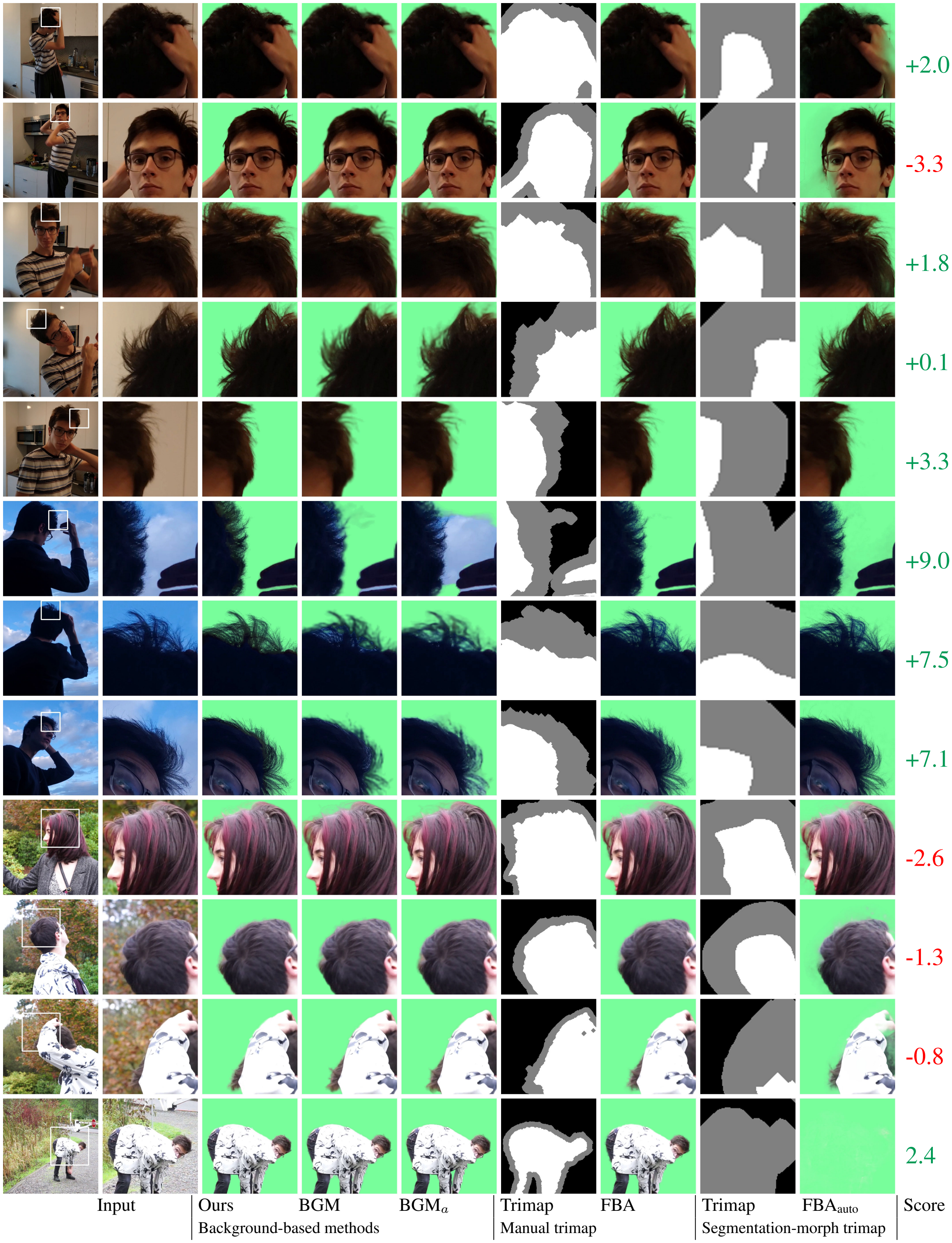

Figure 4: Additional qualitative comparison (1/3). Average user ratings between Ours and BGM are included. A score of -10 denotes BGM is "much better", -5 denotes BGM is "slightly better", 0 denotes "similar", +5 denotes Ours is "slightly better", +10 denotes Ours is "much better". Our method receives an average 3.1 score.

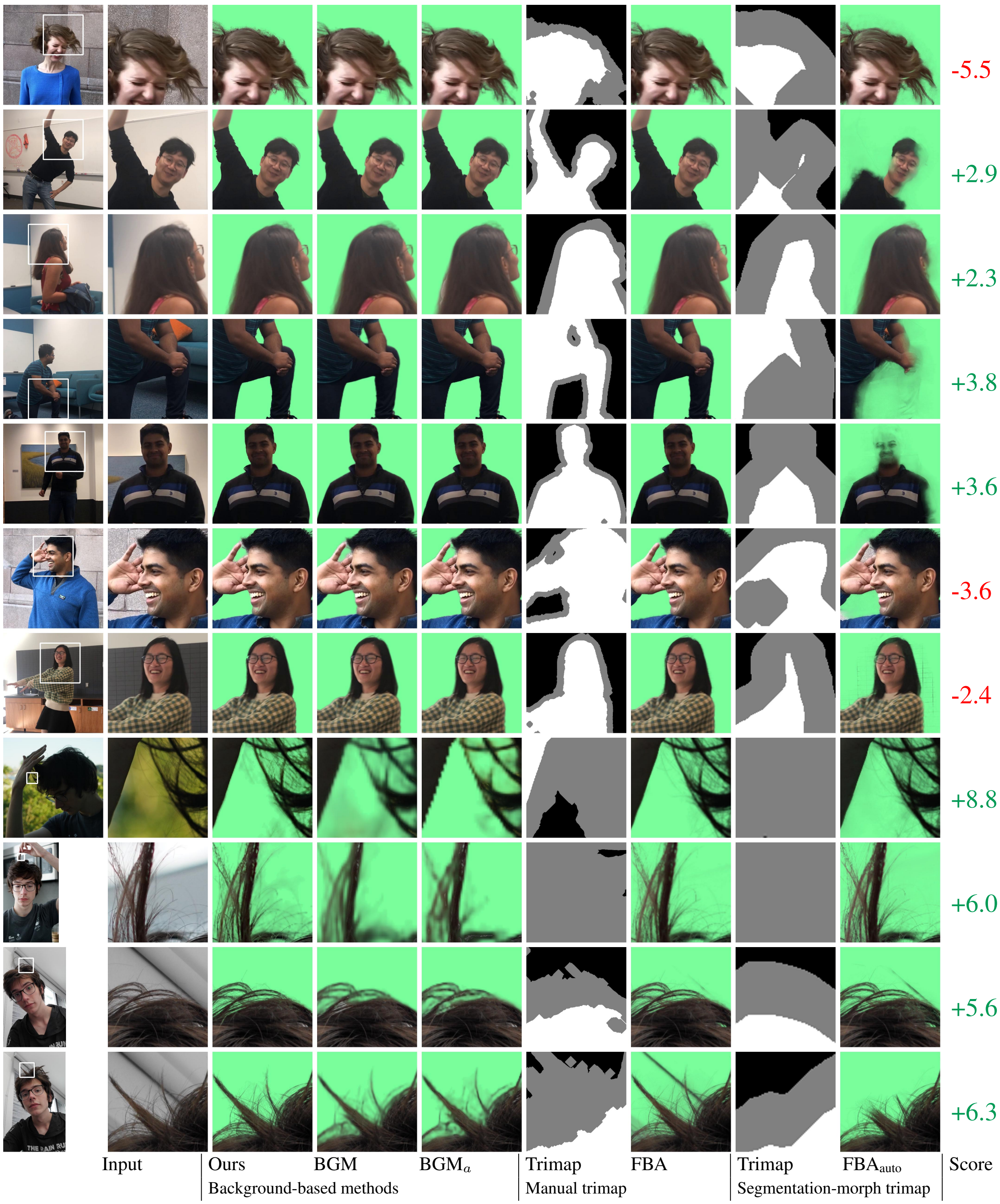

Figure 5: Additional qualitative comparisons (2/3)

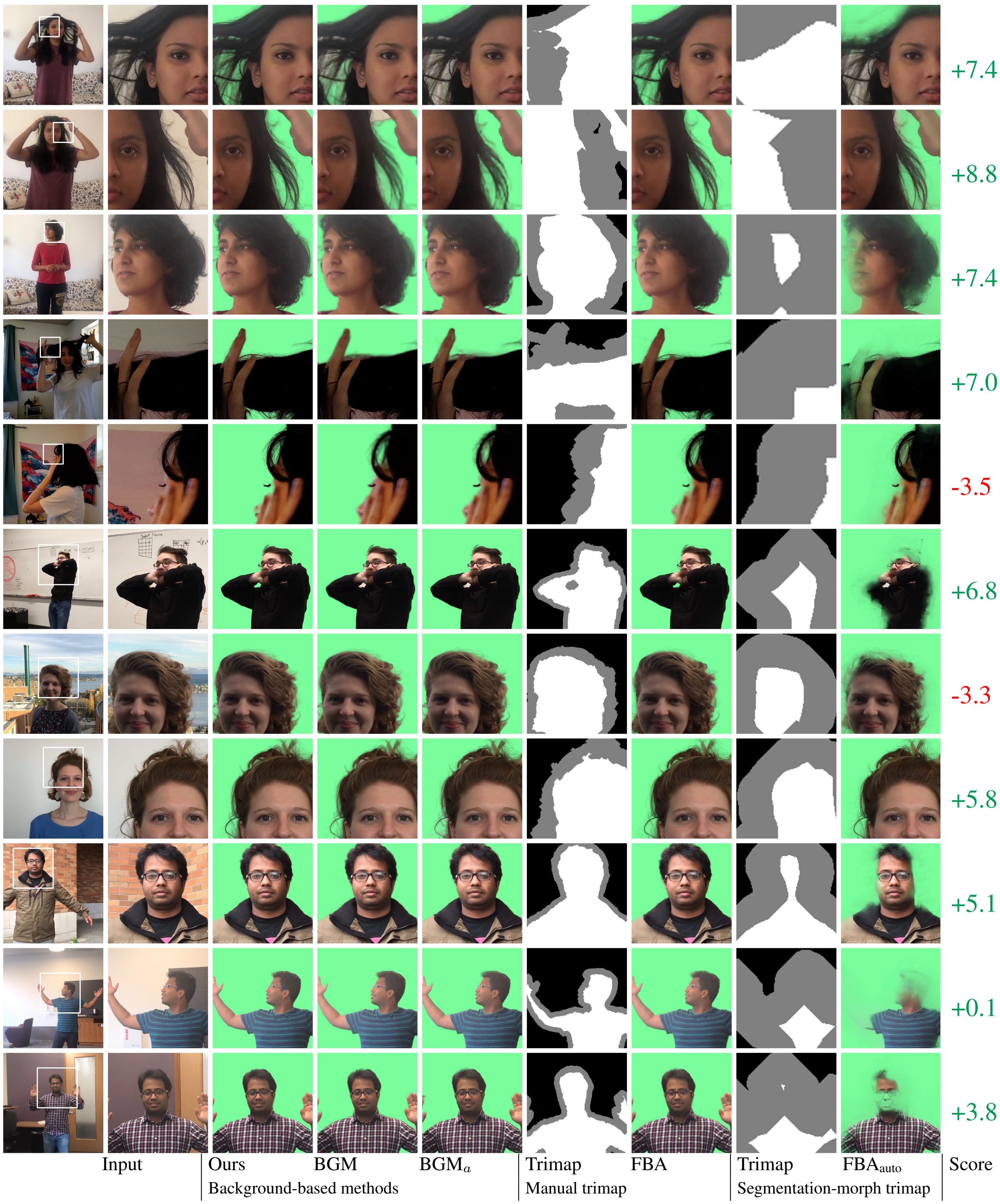

Figure 6: Additional qualitative comparisons (3/3)

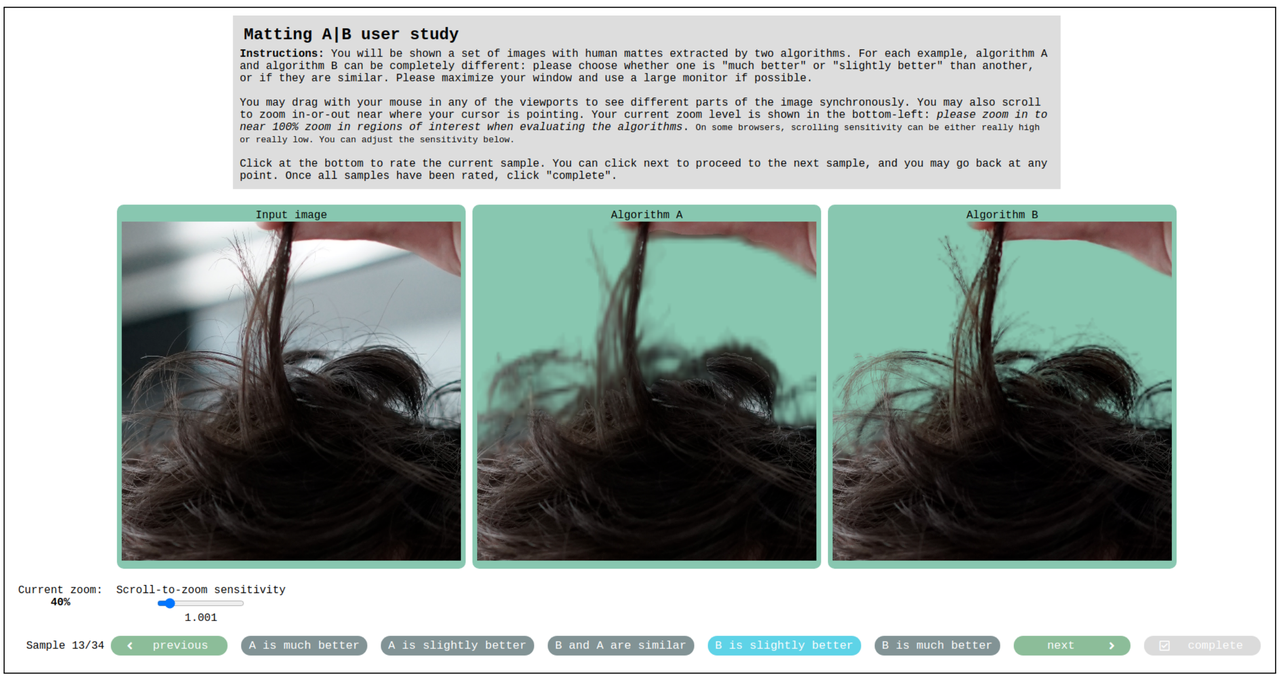

Figure 7: The web UI for our user study. Users are shown the original image and two result images from Ours and BGM methods. Users are given the instruction to rate whether one algorithm is ”much better”, ”slightly better”, or both as ”similar”.