Texture Mixer: A Network for Controllable Synthesis and Interpolation of Texture (Supplemental Material)

Ning Yu[1,2,4] Connelly Barnes[3,4] Eli Shechtman[3] Sohrab Amirghodsi[3] Michal Lukáč[3]
[1]University of Maryland [2]Max Planck Institute for Informatics
[3]Adobe Research [4]University of Virginia
ningyu@mpi-inf.mpg.de connelly@cs.virginia.edu {elishe, tamirgho, lukac}@adobe.com

1. Shuffling procedure visualization

We visualize our shuffling procedure in Figure 1.

2. Texture palette and brush examples

In order to diversify our applications, we, in addition, collected a *plant texture* dataset from Adobe Stock and randomly split it into $1,074$ training and 119 testing images. We show the texture palette and brush application on the *earth texture* and *plant texture* datasets in Figure 2. Furthermore, we show in Figure 3 a camouflage effect of brush painting on the *animal texture* dataset, intentionally given the background patterns similar to brush patterns. It indicates the smooth interpolation over different textures. The dynamic processes of drawing such paintings plus the painting of Figure 1 in the main paper are demonstrated in the videos at GitHub. The videos are encoded using MP4 libx265 codec at 60 frame rate and 16M bit rate.

3. Texture dissolve examples

We shows in Figure 4 additional sequences of video frame samples with gradually varying weights on the *earth texture* and *plant texture* datasets. The corresponding videos plus the video for Figure 3 in the main paper are at GitHub. Two of them are also attached with this material. The videos are encoded using MP4 libx265 codec at 60 frame rate and 16M bit rate.

4. Animal hybridization details and examples

In Figure 5, we show and illustrate the pipeline to hybridize a dog and a bear by interpolating their furs in the hole for the transition region. Two additional results are show in Figure 6.

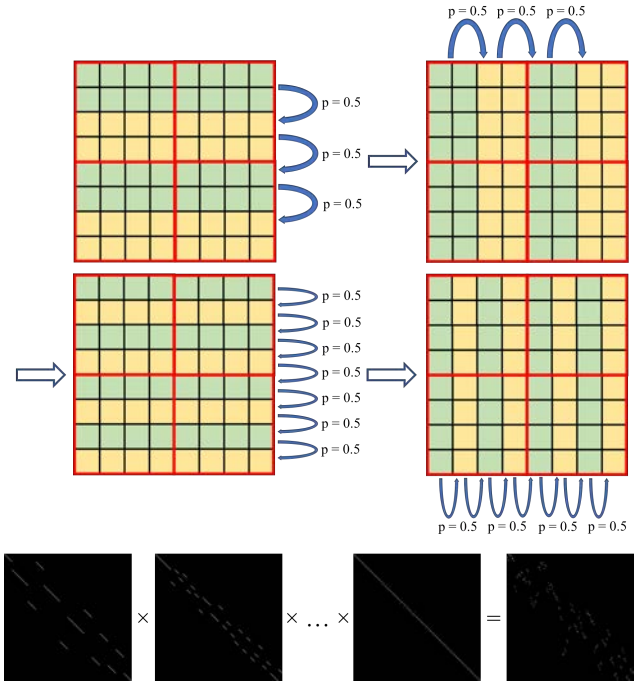

Figure 1. Our shuffling procedure. On the top figure we visualize the procedure example $P_0 \circ P_1(T(z_i^l))$, where z_i^l is a 4×4 cell in a 2×2 grid framed in red. The procedure is composed of the random swapping operations between a green strip and its subsequently adjacent yellow strip in four directions: top-down, bottom-up, left-to-right, and right-to-left. The swapping operations start at scale 2 (the 1st row) and then are repeated at scale 1 (the 2nd row). The bottom figure (zoom-in to check) demonstrates the composition of swapping operations at several scales applied to an identity matrix. The resulting composed matrix can serve as a row permutation matrix left-multiplied to $T(z_i^l)$. Another similar matrix can serve as a column permutation matrix right-multiplied to $T(z_i^l)$. The row and column permutation matrices are independently sampled for each training iteration.

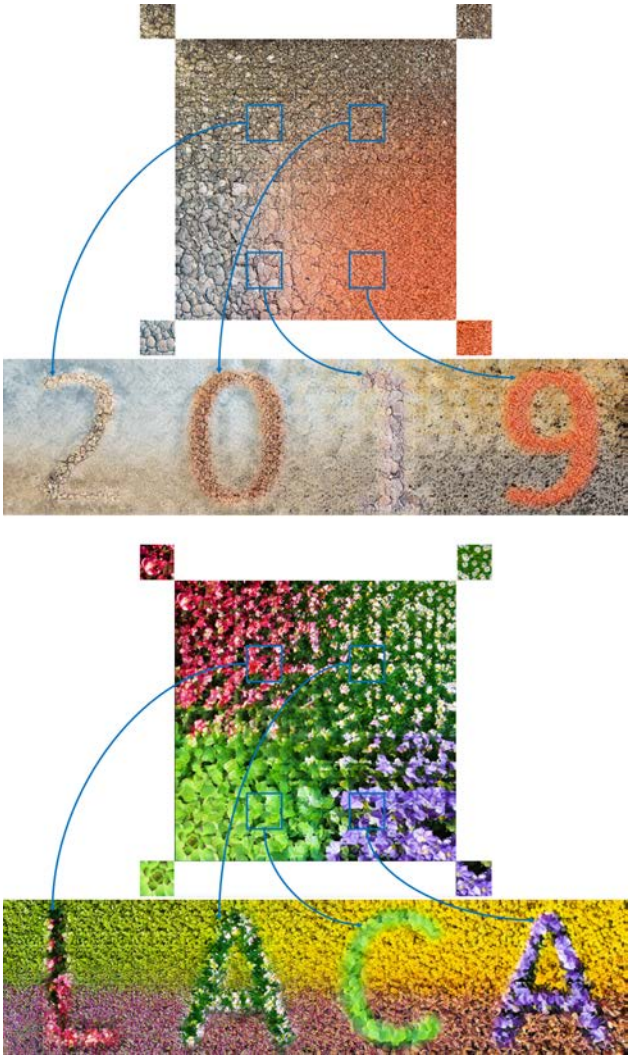

Figure 2. Texture interpolation and texture painting using our network on the *earth texture* and *plant texture* datasets. The top part shows a 1024×1024 palette created by interpolating four source textures at the corners outside the palette. The bottom part shows a 512×2048 painting of letters with different textures sampled from the palette. The letters are interpolated by our method with the background, also generated by our interpolation.

5. Network architecture details

We set the texture image size to be 128 throughout our experiments. The proposed E^l, E^g, D^{rec}, and D^{itp} architectures are employed or adapted from the discriminator architecture in [7], where layers with spatial resolutions higher than 128×128 are removed. We also adopt their techniques including *pixel normalization* instead of batch normalization, and *leaky ReLU activation*. The *minibatch standard deviation channel* is also preserved for D^{rec} and D^{itp}, but not for E^l and E^g. For E^l, we truncate the architecture so that the output local latent tensor is m times

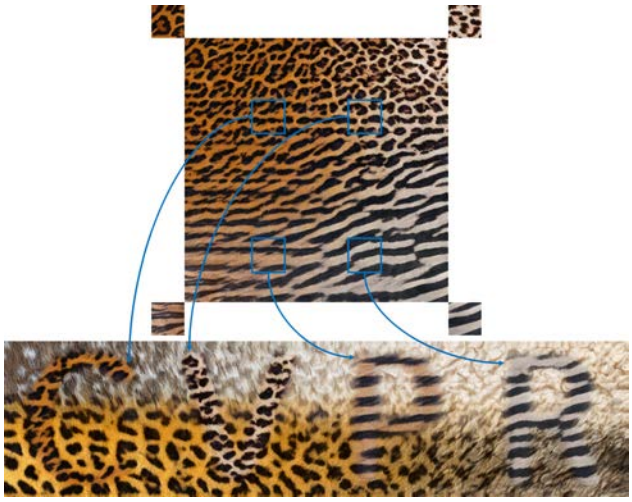

Figure 3. Texture interpolation and texture painting with camouflage effect using our network on the *animal texture* dataset. The top part shows a 1024×1024 palette created by interpolating four source textures at the corners outside the palette. The bottom part shows a 512×2048 painting of letters with different textures sampled from the palette. The letters are interpolated by our method with the background, also generated by our interpolation.

smaller than the input texture, where $m = 4$ in all our experiments. We tried using deeper architectures but noticed this does not favor reconstruction quality. For E^g, we truncate the architecture at 1×1 resolution right before the fully-connected layer, because we are doing encoding rather than binary classification.

Our G is modified from the fully-convolutional generator architecture from Karras *et al*. [7] with three changes. First, the architecture is truncated to accept an input spatial resolution that is $m = 4$ times smaller than the texture size, and to output the original texture size. Second, the local and global latent tensor inputs are concatenated together along the channel dimension after they are fed into G. A third important point is that since our goal is to interpolate a larger texture image output, at the bottleneck layer the receptive field should be large enough to cover the size of input image. We do this by inserting a chain of five residual blocks [4] in the generator after local and global latent tensor concatenation and before the deconvolution layers from [7].

6. Training details

Our training procedure again follows the progressive growing training in [7], where E^l, E^g, G, D^{rec}, and D^{itp} simultaneously grow from image spatial resolution at 32×32 to 128×128. We repeatedly alternate between performing one training iteration on D^{rec} and D^{itp}, and then four training iterations on E^l, E^g, and G. At each intermediate resolution during growth, the *stabilization stage* takes 1 epoch of training and the *transition stage* takes 3 epochs.

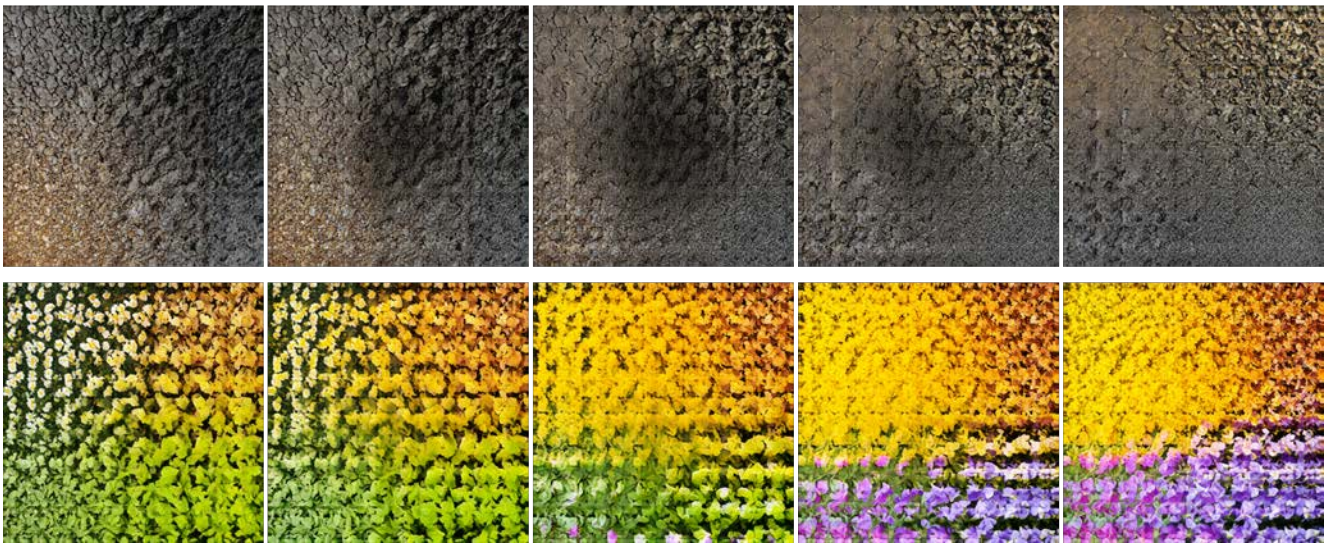

Figure 4. Sequences of dissolve video frame samples with size 1024×1024 on the *earth texture* and *plant texture* datasets, where each frame is also with effect of interpolation.

After the growth is completed, we keep training the model until a total of 20 epochs is reached.

We use Adam [8] as our optimization approach with no exponential decay rate $\beta_1 = 0.0$ for the first moment estimates and with the exponential decay rate for the second moment estimates $\beta_2 = 0.99$. The learning rate is set to 0.001 before the model grows to the final resolution 128×128 and then is set to 0.0015 at 128×128. The trainable weights of the autoencoder and discriminator are initialized with the *equalized learning rate* technique from [7]. We train and test all our models on 8 NVIDIA GeForce GTX 1080 Ti GPUs with 12GB of GPU memory each. Based on the memory available and the training performance, we set the batch size at 64, and the training lasts for 3 days.

The weights of losses is not sensitive to the dataset. We simply set them to balance the order of magnitude of each loss: $\lambda_1 = 100$, $\lambda_2 = \lambda_4 = 0.001$, and $\lambda_3 = \lambda_5 = 1$.

7. Experimental evaluation details

Seam classifier. The architecture and training details of seam classifier are almost the same as those of D^{rec} and D^{itp} except (1) we remove the minibatch standard deviation channel, (2) we add a sigmoid activation layer after the output layer for the binary cross-entropy loss computation, and (3) we exclude the progressive growing process. We directly use the sigmoid output of the classifier as the seam score for each input image.

Repetition classifier. The architecture and training details of repetition classifier are almost the same as those of the seam classifier except the input image size is 128×256 instead of 128×128, where the negative examples are random crops of size 128×256 from real datasets and the positive examples are horizontally tiled twice from random crops of size 128×128 from real datasets.

Inception model finetuning. Our inception scores are computed from the state-of-the-art *Inception-ResNet-v2* inception model architecture [13] finetuned with our two datasets separately.

8. Baseline method details

Naïve α-blending. We split the output into 8 square tiles, where the end textures are copied as-is, and the intervening tiles (copies of the two boundaries) are linearly per-pixel α-blended.

Image Melding [3]. We selected Image Melding in its inpainting mode as a representative of patch-based methods. We use the default setting of the official public implementation[1].

AdaIN [5]. Style transfer techniques can potentially be leveraged for the interpolation task by using random noise as the content image and texture sample as the style. We interpolate the neural features of the two source textures to vary the style from left to right. We consider AdaIN as one representative of this family of techniques, as it can run with arbitrary content and style images. However, with the default setting of the official implementation[2] and their pre-trained model, AdaIN has some systematic artifacts as it over-preserves the noise appearance. Therefore, we only show qualitative results in Figure 4 in the main paper, and in Figure 9 to Figure 13 here. We did not include this method in the quantitative evaluation.

[1] https://www.ece.ucsb.edu/~psen/melding

[2] https://github.com/xunhuang1995/AdaIN-style

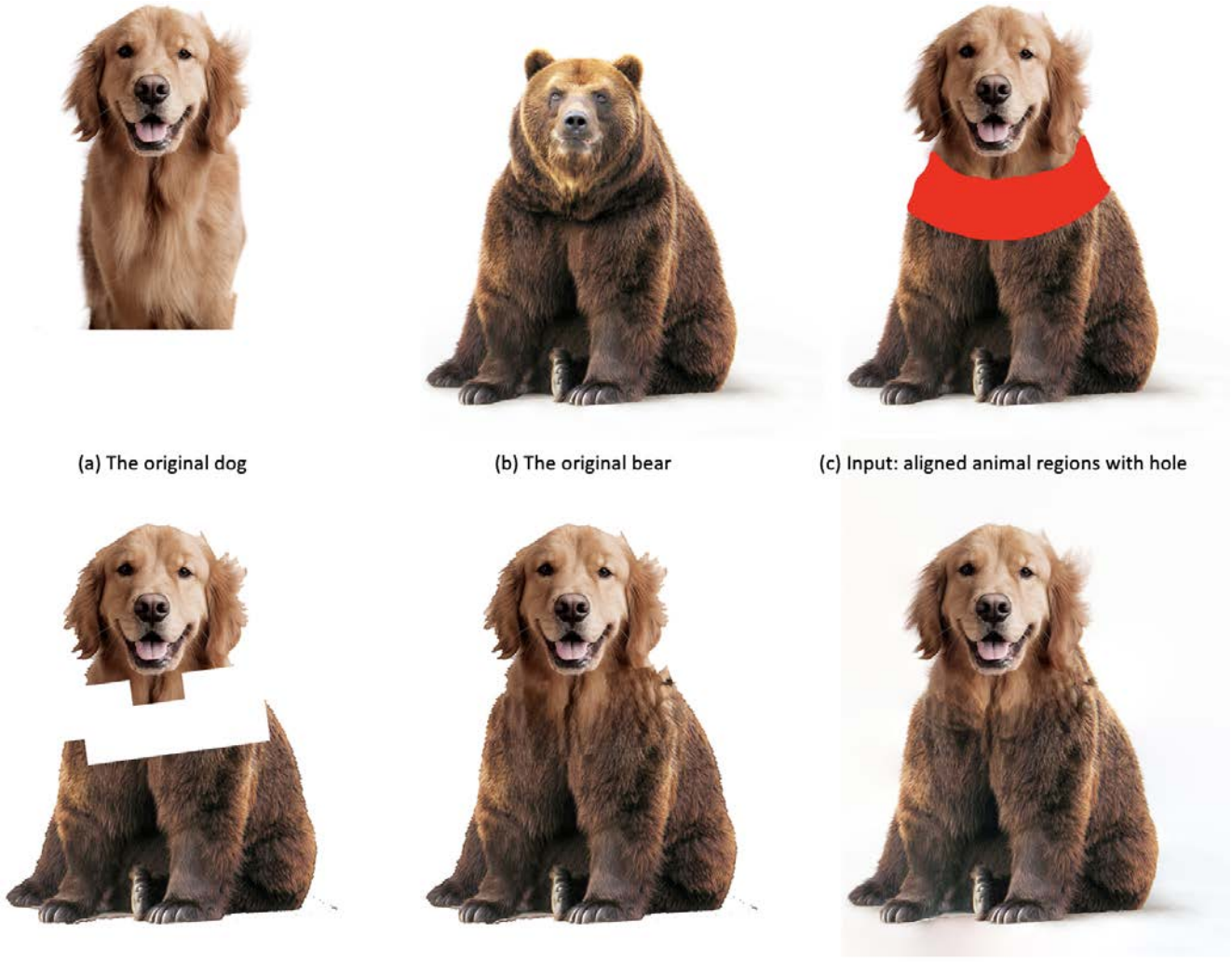

Figure 5. Animal hybridization pipeline. (a) and (b) are two original images. (c) is the input to the pipeline, composed of the aligned regions of (a) and (b) in the same image and the hole for the transition region. (d) shows that we rasterize the hole because Texture Mixer works on square patches. The patch size is 128×128. (e) shows that we interpolate in the rasterized hole region using adjacent texture patches, and then composite this back on top of the original image. This involves two details: (1) if a texture patch covers background, those background pixels are replaced by foreground pixels using the Content-Aware Fill function in Photoshop; and (2) we blend latent tensors between two images using spatially varying weights. (f) We use graph cuts [9] and standard Poisson blending [12] to postprocess the boundaries.

WCT [10]. WCT is an advancement over AdaIN with whitening and coloring transforms (WCT) as the stylization technique and works better on our data. We use its official public implementation[3] with default setting and their pre-trained model. By design, this method does not guarantee accurate reconstruction of input samples.

DeepFill [14]. Texture interpolation can be considered an instance of image hole-filling. The training code for the most recent work in this area [11] is not released yet. We, therefore, tried another recent method called DeepFill [14] with their official code[4]. We re-trained it for our two texture datasets separately with 256×256 input image size, 128×128 hole size, and all the other default settings. The interpolation results suffered from two major problems: (i) the method is not designed for inpainting wide holes (in our experiment 128×768) because of lack of such wide ground truth; (ii) even for a smaller hole with size 128×128, as shown in the failure cases in Figure 4 in the main paper and

[3] https://github.com/Yijunmaverick/UniversalStyleTransfer

[4] https://github.com/JiahuiYu/generative_inpainting

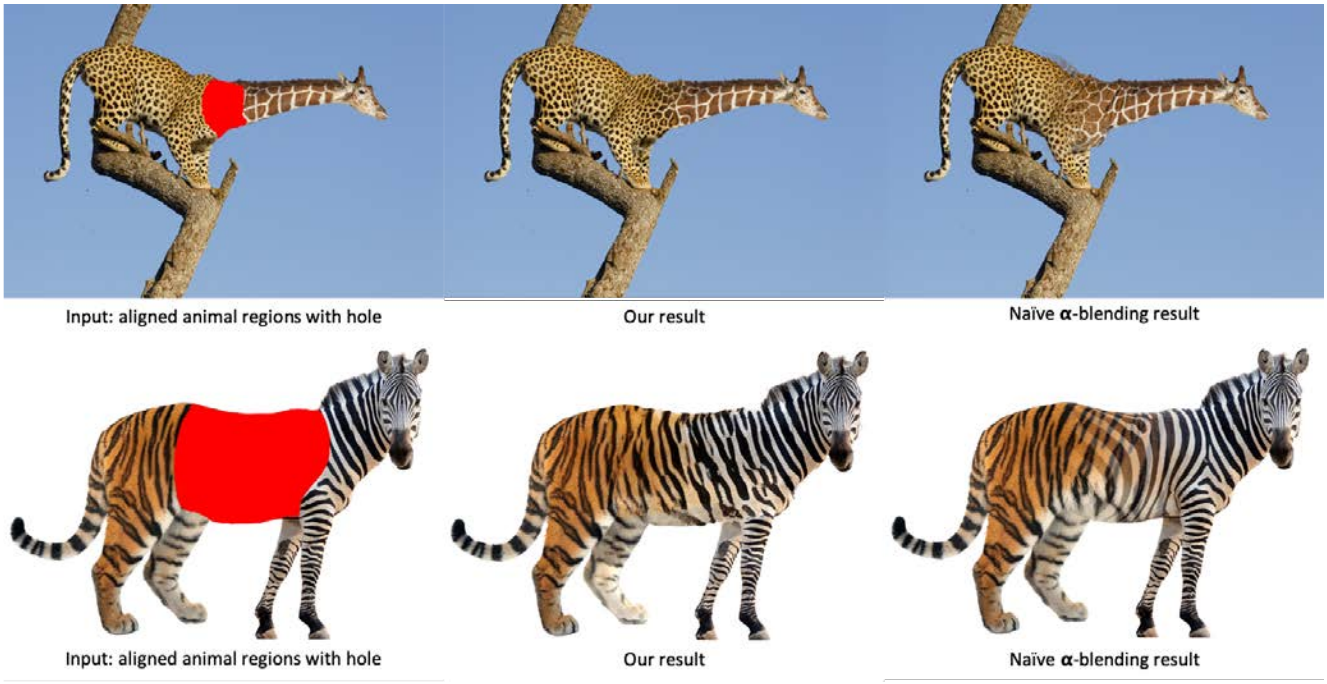

Figure 6. Two animal hybridization examples. The top image is in the size 2636×3954 and the bottom image is in the size 2315×2664. Our interpolations between the two animal furs is smoother, has less ghosting, and is more realistic than those of the Naïve α-blending.

in Figure 9 to Figure 13, this work systematically failed to merge the two source textures gradually. We, therefore, excluded this method from our quantitative comparisons.

PSGAN [1]. The most closely related work to ours, PSGAN, learns a smooth and complete neural manifold that favors interpolation. However, it only supports constraining the interpolation in latent space, and lacks a mechanism to specify end texture conditions using image examples. To allow for a comparison, we have trained a PSGAN model for each of our datasets separately, using the official code[5] and default settings. Then, we optimize for the latent code that corresponds to each of the end texture images by back-propagating through L-BFGS-B [2]. We use the gradients of the L_1 reconstruction loss and the Gram matrix loss [6] and initialize randomly the latent vectors. We use 100 different initializations and report the best result.

9. More qualitative comparisons

More qualitative comparisons are shown from Figure 9 to Figure 13. They are all used for quantitative comparison and user study as reported in Table 1 in the main paper. In addition, Figure 14 demonstrates one of our failure examples when dealing with strong structural textures.

[5] https://github.com/zalandoresearch/psgan

10. User study details

Our user study webpage is shown in Figure 7. To guarantee the accuracy of users' feedback, we insert sanity check by comparing our interpolation results with another naive baseline results where the transition regions are filled with constant pixel values. The constant value is computed as the mean value of the two end texture pixels, as shown in Figure 8. The preference should be obvious and deterministic without subjective variance. In our statistics, only two users made a mistake once on the sanity check questions. We then manually checked their answers to other real questions but didn't notice any robot or laziness style. We there trust and accept all users' feedback.

References

[1] U. Bergmann, N. Jetchev, and R. Vollgraf. Learning texture manifolds with the periodic spatial GAN. In *Proceedings of the 34th International Conference on Machine Learning*, pages 469–477, 2017. 5

[2] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995. 5

[3] S. Darabi, E. Shechtman, C. Barnes, D. B. Goldman, and P. Sen. Image melding: Combining inconsistent images using patch-based synthesis. *ACM Trans. Graph.*, 31(4):82–1, 2012. 3

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE con-*

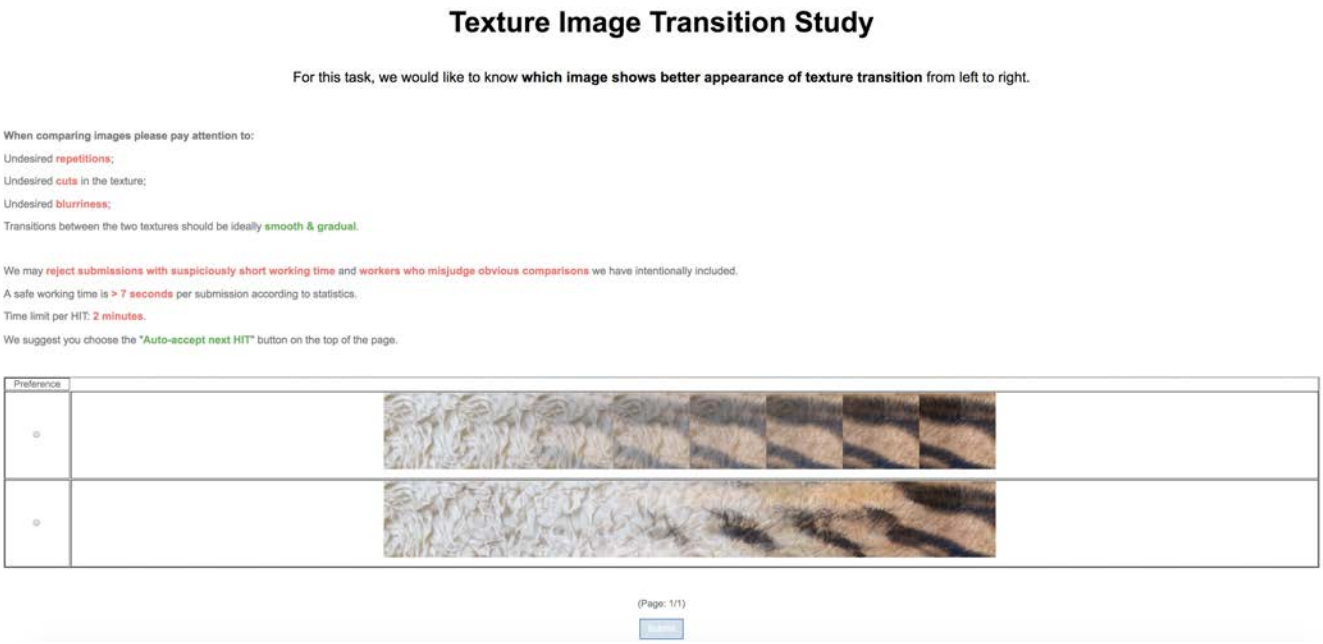

Figure 7. User study webpage design.

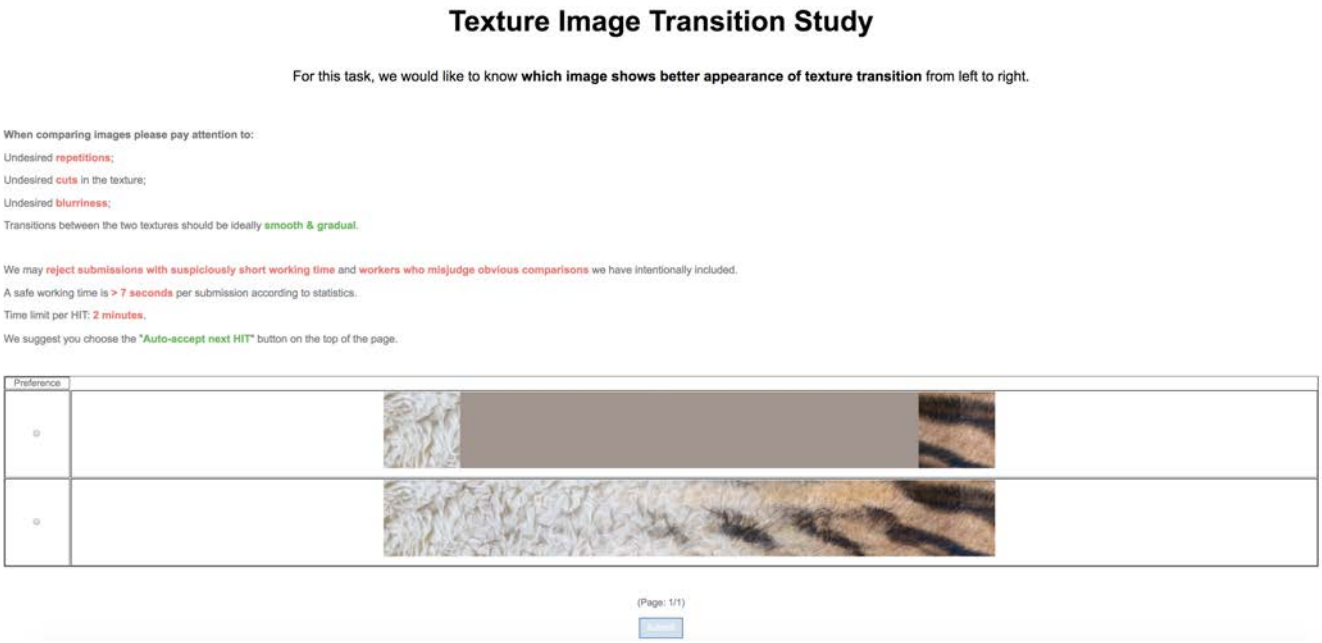

Figure 8. A user study with sanity check where the preference should be obvious and deterministic without subjective variance.

ference on computer vision and pattern recognition, pages 770–778, 2016. 2

[5] X. Huang and S. J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, pages 1510–1519, 2017. 3

[6] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016. 5

[7] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017. 2, 3

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3

[9] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics (ToG)*, 22(3):277–286, 2003. 4

[10] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang. Universal style transfer via feature transforms. In *Advances in Neural Information Processing Systems*, pages 386–396, 2017. 4

[11] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 4

[12] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Transactions on graphics (TOG)*, 22(3):313–318, 2003. 4

[13] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017. 3

[14] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang. Generative image inpainting with contextual attention. *arXiv preprint arXiv:1801.07892*, 2018. 4

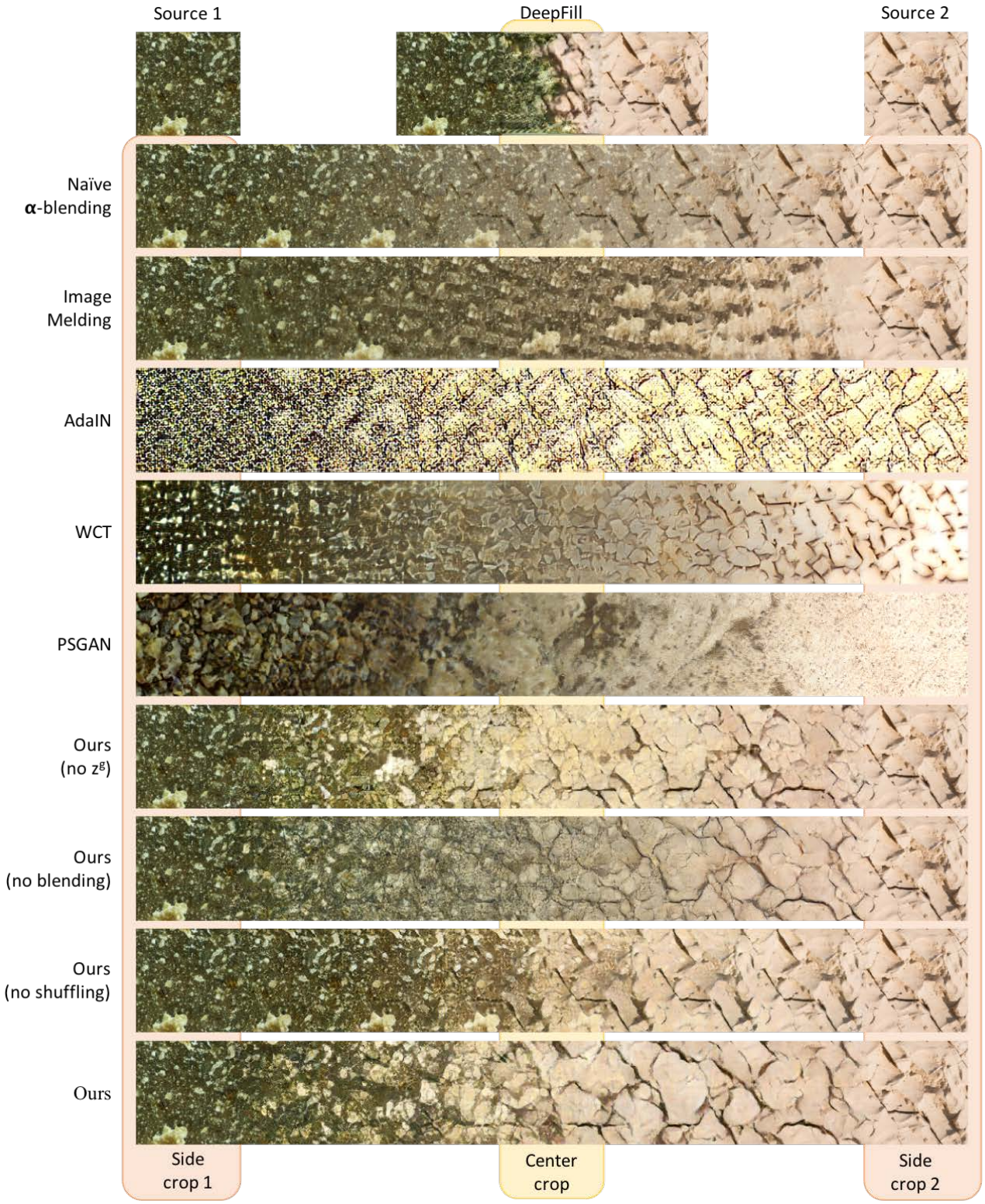

Figure 9. Qualitative demonstrations and comparisons of horizontal interpolation in the size of 128×1024 on the *earth texture* samples. We use the two side crops with the orange background for SPD measurement, and the center crop with the light yellow background for the other proposed quantitative evaluations.

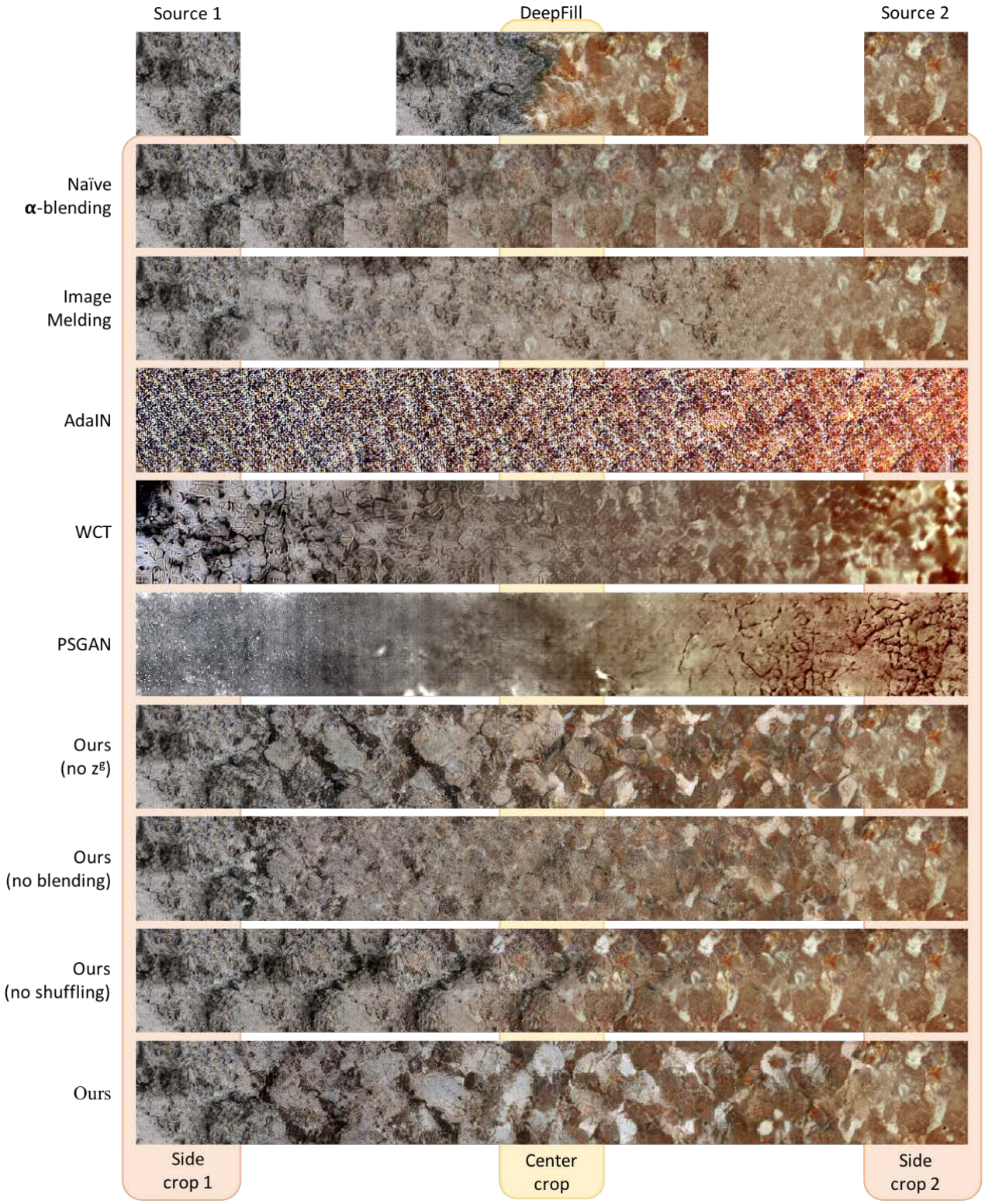

Figure 10. Qualitative demonstrations and comparisons of horizontal interpolation in the size of 128×1024 on the *earth texture* samples. We use the two side crops with the orange background for SPD measurement, and the center crop with the light yellow background for the other proposed quantitative evaluations.

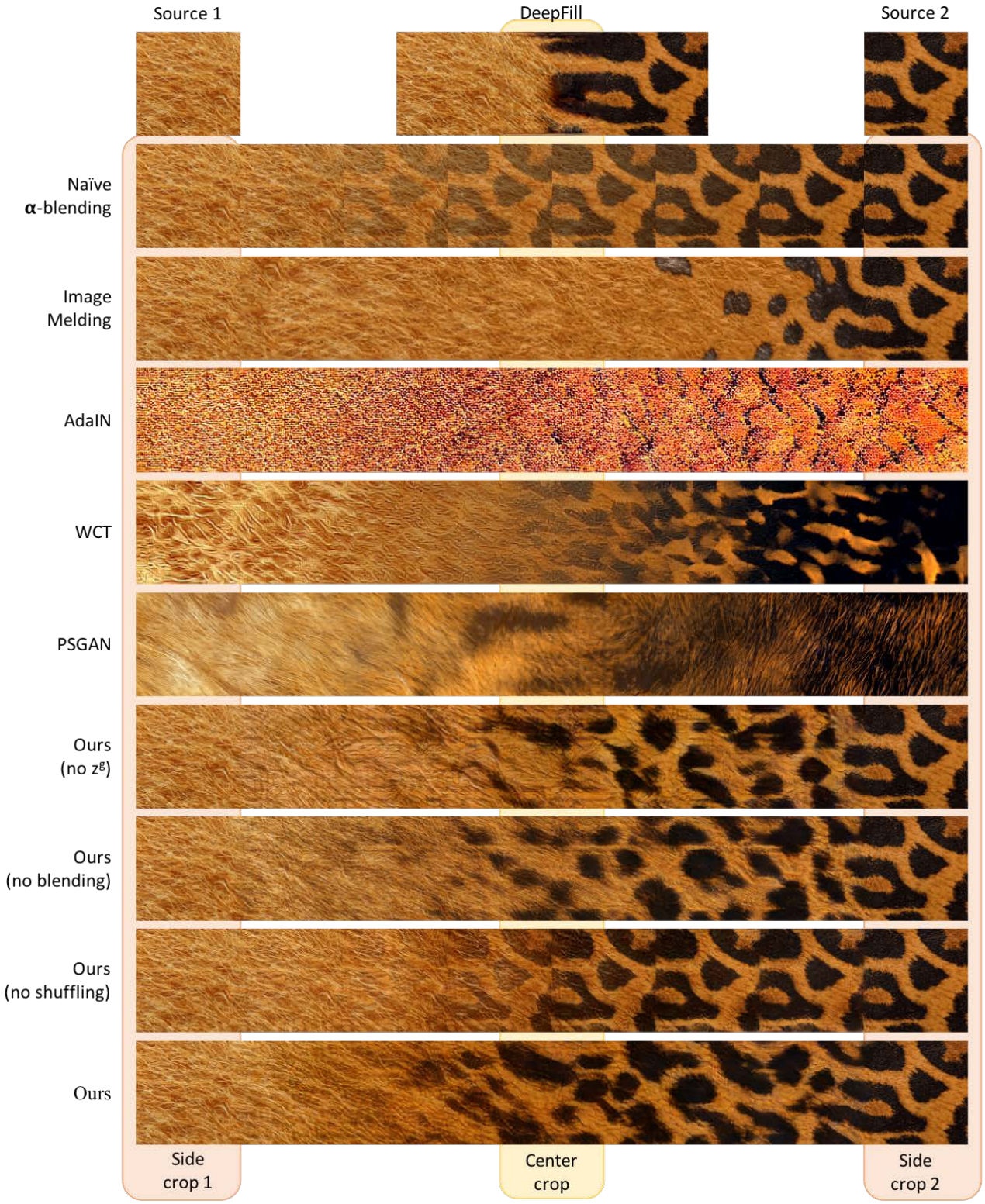

Figure 11. Qualitative demonstrations and comparisons of horizontal interpolation in the size of 128×1024 on the *animal texture* samples. We use the two side crops with the orange background for SPD measurement, and the center crop with the light yellow background for the other proposed quantitative evaluations.

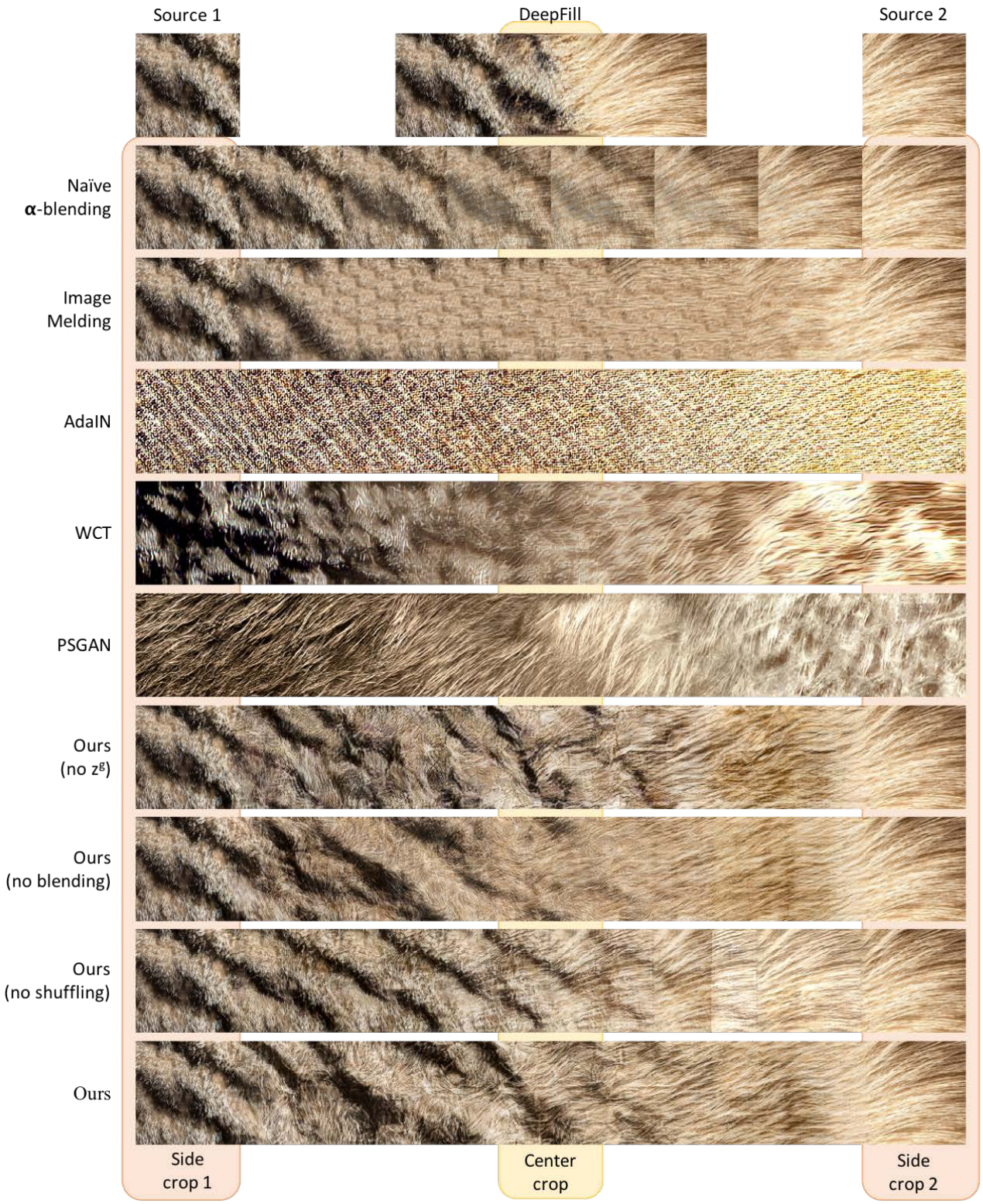

Figure 12. Qualitative demonstrations and comparisons of horizontal interpolation in the size of 128×1024 on the *animal texture* samples. We use the two side crops with the orange background for SPD measurement, and the center crop with the light yellow background for the other proposed quantitative evaluations.

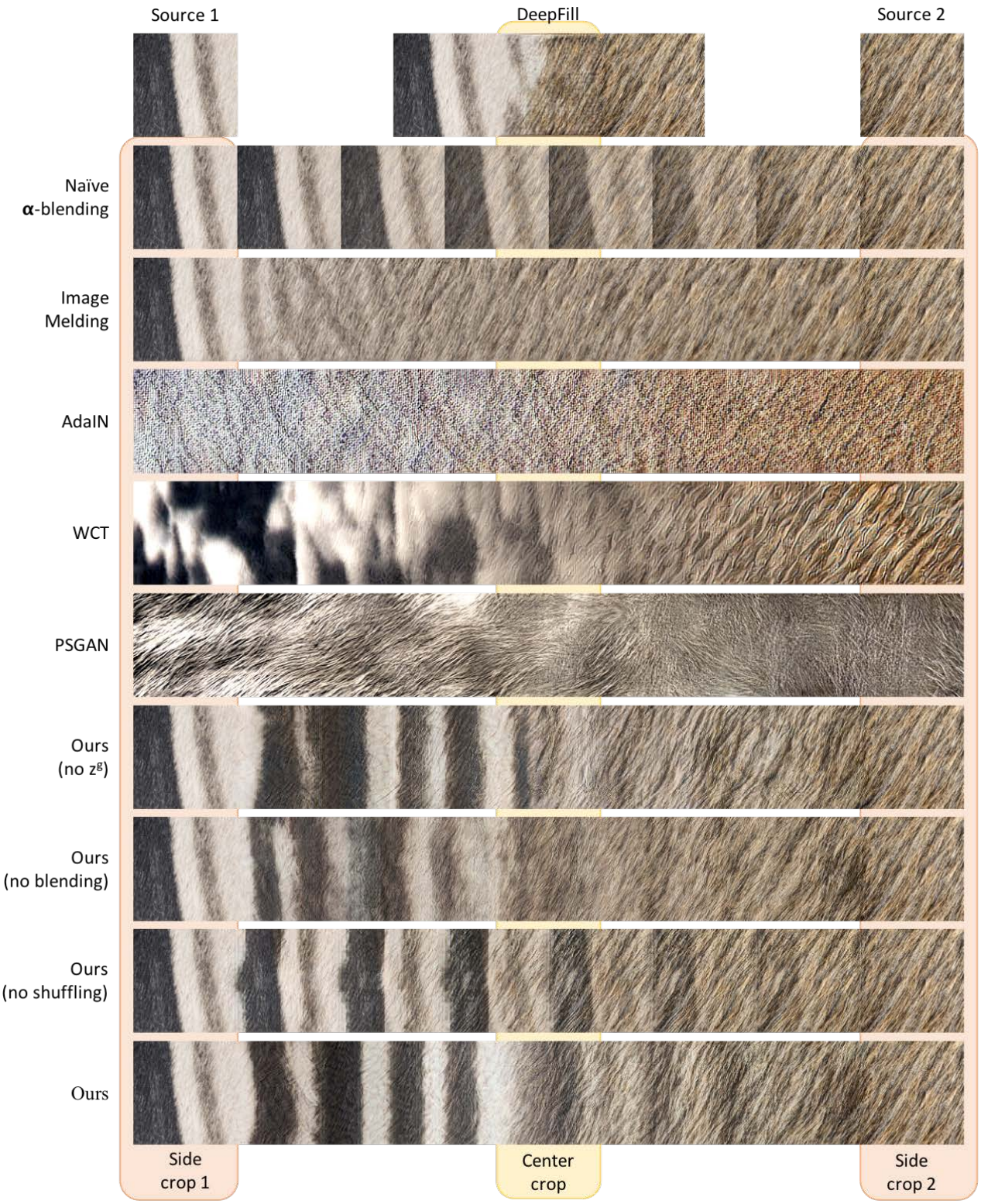

Figure 13. Qualitative demonstrations and comparisons of horizontal interpolation in the size of 128×1024 on the *animal texture* samples. We use the two side crops with the orange background for SPD measurement, and the center crop with the light yellow background for the other proposed quantitative evaluations.

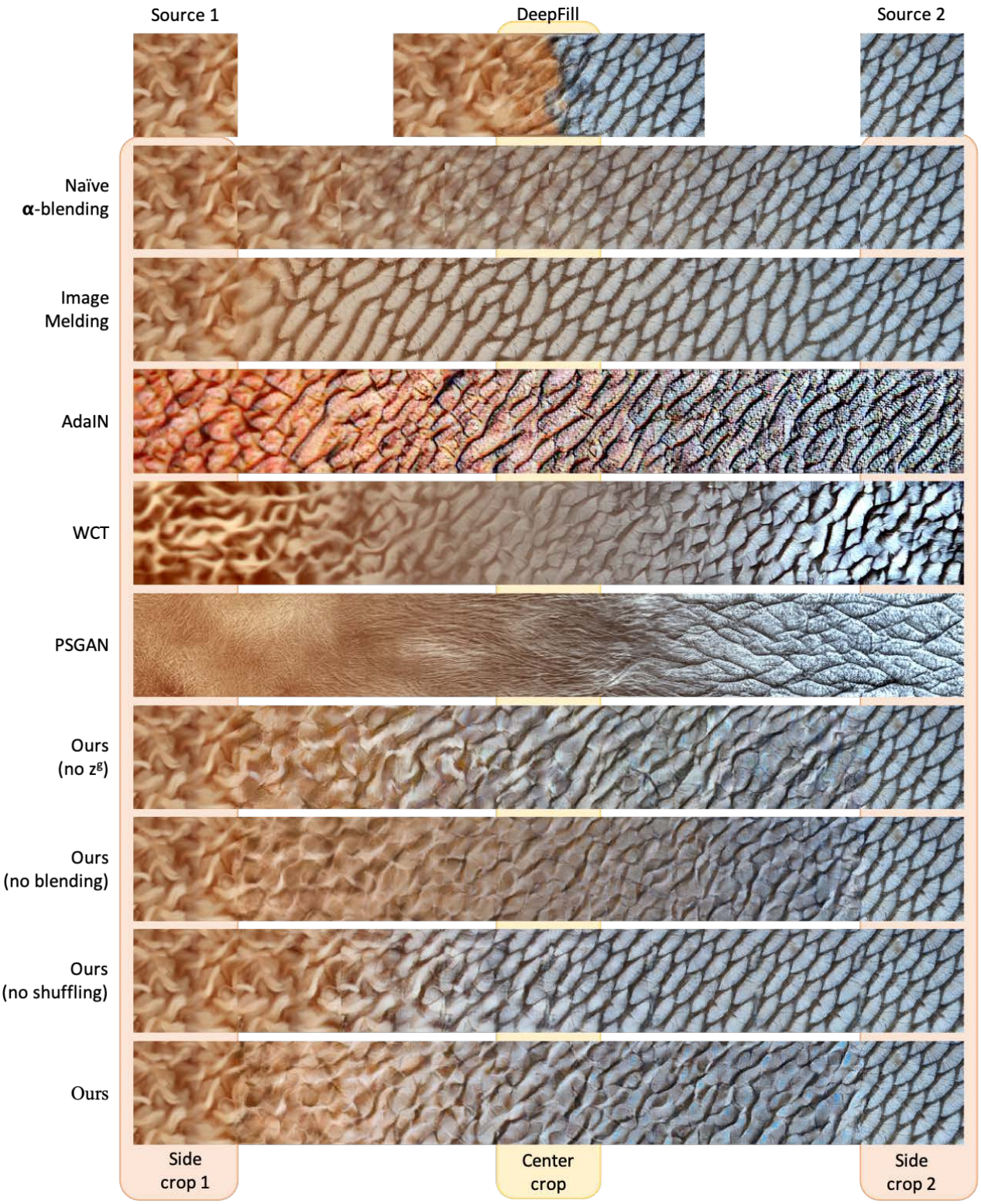

Figure 14. A qualitative demonstration of one of our failure examples. When dealing with strong structural source texture on the right, our full method didn't outperform ours without random shuffling during training, and didn't outperform Image Melding either.