SeamlessGAN: Self-Supervised Synthesis of Tileable Texture Maps Supplementary Material

Carlos Rodriguez-Pardo, Elena Garces

1 Introduction

In this supplementary material, we provide further implementation details, results of SeamlessGAN, and comparison with other methods. In particular:

- Section 2 contains a thorough description of the implementation details.
- Section 3 describes the implementation details used for our comparison with other tileable texture synthesis methods.
- Figure 1 shows further results on our ablation study on the design of the generator architecture $\mathcal{G}$ for the texture stack synthesis problem.
- Figure 2 shows more examples on the quality metric obtained using the discriminator.
- Figure 3 shows further examples on the capability of SeamlessGAN for generating multiple texture maps from a single input.
- Figure 4 shows an extended ablation study on the influence of the loss function on the generated results.
- Figures 5 and 6 show an ablation study on the influence of the number of layers in the discriminator and generator on the quality of the synthesized textures.
- Figure 7 shows additional comparisons with previous methods on Moritz's dataset [1].
- Figure 8 shows an experiment on tileable texture transfer.
- Figure 9 shows additional results on multi-layer tileable texture synthesis.
- In pages 12-22, we include a large pool of high-resolution results of SeamlessGAN trained only using the albedo, showcasing the tileability of our outputs.

2 Implementation details

Training details: We use PyTorch [2] as our learning framework, and Adam [3] as our optimization algorithm. For it, we use an initial learning rate of $\alpha = 0.0002$, which is divided by 5, after iterations 30000 and 40000. The networks are trained for a total of 50000 iterations, using a batch size of 1.The momentum parameters are kept to the default values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. We additionally apply an ℓ_2 regularization of 10^{-5} to the optimization of both networks, which may help obtain better synthesized images [4]. This configuration is the same for the training of both the discriminator and the generator. In contrast to other work in unsupervised image generation [5], both networks are updated after each iteration. The models are trained and evaluated using a single NVIDIA GeForce GTX 1080Ti, leveraging half-precision training and automatic gradient scaling [6]. Evaluation is done in half precision for faster inference and reduced memory consumption. Using reduced numeric precision does not yield worse synthesized textures.

Network architecture: We follow closely the architectures defined in [7]. For the **Generator** $\mathcal{G}$, we use replication padding as the padding method for the entire architecture, ReLU non-linearities and Instance Normalization [8] before each non-linearity. This network receives textures of $3 \times k \times k$ pixels, which are first processed by an encoder $\mathcal{E}$, which transforms it to a $256 \times \frac{k}{4} \times \frac{k}{4}$ latent vector x_0. Following standard practice on residual convolutional networks [9], the first layer of $\mathcal{E}$ is comprised by 64 7×7 convolutional filters. The rest of the network, unless stated otherwise, uses standard 3×3 kernels. This latent vector is further processed by 5 residual blocks $\mathrm{x}_i \leftarrow \mathcal{R}_i(\mathrm{x}_{i-1}) + \mathrm{x}_{i-1}$, which maintain the latent vector dimensions. Then, a decoder **G** takes x_5 and, using three convolutional layers with transposed convolutions [10], followed by ReLU operations, it returns the estimated texture, with $2k \times 2k$ resolution. As in the encoder, the last layer of **G** contains 64 7×7 convolutional filters. There are as many decoders **G** as individual maps in the texture stack. When tiling the latent space x_0, this doubles it spatial resolution, thus becoming a $256 \times \frac{k}{2} \times \frac{k}{2}$ latent vector. As such, once processed by the residual blocks and the decoders, the network outputs a $4k \times 4k$ texture stack.

For the **discriminator** $\mathcal{D}$, we simply follow a *PatchGAN* architecture [5], [11], [12], [7], which outputs local estimations of the probability of the input image being real or generated. We use 4 blocks of layers, each comprised of, in order, a convolutional layer with 4×4 kernels, an Instance

- *Carlos Rodriguez - Pardo is with SEDDI (28007, Madrid, Spain) and with Universidad Carlos III de Madrid (28005, Madrid, Spain). E-mail: carlos.rodriguezpardo.jimenez@gmail.com*
- *Elena Garces is with SEDDI (28007, Madrid, Spain) and with Universidad Rey Juan Carlos (28933, Madrid, Spain) E-mail: elena.garces@seddi.com*

Normalization [8] layer and a Leaky ReLU non-linearity [13]. We use a stride of 2 in the first and last layers to decrease the dimensions of the latent representations. The first block of layers contains 64 trainable kernels, which are doubled after each block. The last layer thus contains 512 trainable filters. In the last layer, we remove the normalization operation and substitute the non-linearity by a Sigmoid operation, to transform the output into the $(0, 1)$ range. The discriminator receives $2k \times 2k$ resolution stacks, and outputs a $1 \times \frac{k}{2} \times \frac{k}{2}$ resolution estimation map.

3 Comparison with other methods

In this section, we outline the implementation details used for our comparison with other tileable texture synthesis methods, which included Histogram-Preserving Blending by Deliot and Heitz [14], Graph-Cuts synthesis by Li *et al.* [15], Repeated Pattern Detection by Rodriguez - Pardo *et al.* [16], Periodic Spatial GAN by Bergmann *et al.* [17] and Self-Organizing Textures by Niklasson *et al.* [18].

Histogram-Preserving Blending: We use the implementation provided by the authors. The only hyperparameter to be changed is the blending border size, which is set to the default value of 33% for all the textures.

Graph-Cuts synthesis: To allow for comparison between single-map texture synthesis methods, we run this method to output only the albedo map of the stack, using the default values of $\lambda_d = 1$, a ratio of 0.64, a crop ratio of 0.6 and a margin size of 72 pixelsThe images are not rescaled to any input or target size, instead, we use their original resolution.

Repeated Pattern Detection: This paper proposed the use of some preprocessing methods for improving the regularity of the input images, which were not used for our comparison. We use their default hyperparameter values, including $\delta = 0.65$, $\lambda = 0.8$ and the deep perceptual loss weighting proposed in [19].

Periodic Spatial GAN: This method can handle both single-image and multiple image datasets. We are interested in the former, and, as such, each GAN is trained using only one texture sample. We train their model using Theano [20], and use their default configuration of a learning rate of 0.0002, a $\beta_1 = 0.5$ as the momentum term for the optimization algorithm, a batch size of 25, and train their network for 10 epochs, each comprised of 25000 training steps. The network parameters and sizes are exactly as defined in the paper.

Self-Organizing Textures: This method is computationally expensive and, as such, it requires that the input textures are rescaled to 128×128 pixels. We train their method using Tensorflow [21] for 5000 iterations. We use the default training configuration, with an starting learning rate of 0.002, which is then reduced by half at iterations 2000 and 4000. We do not modify the image parametrization design or loss function in any way.

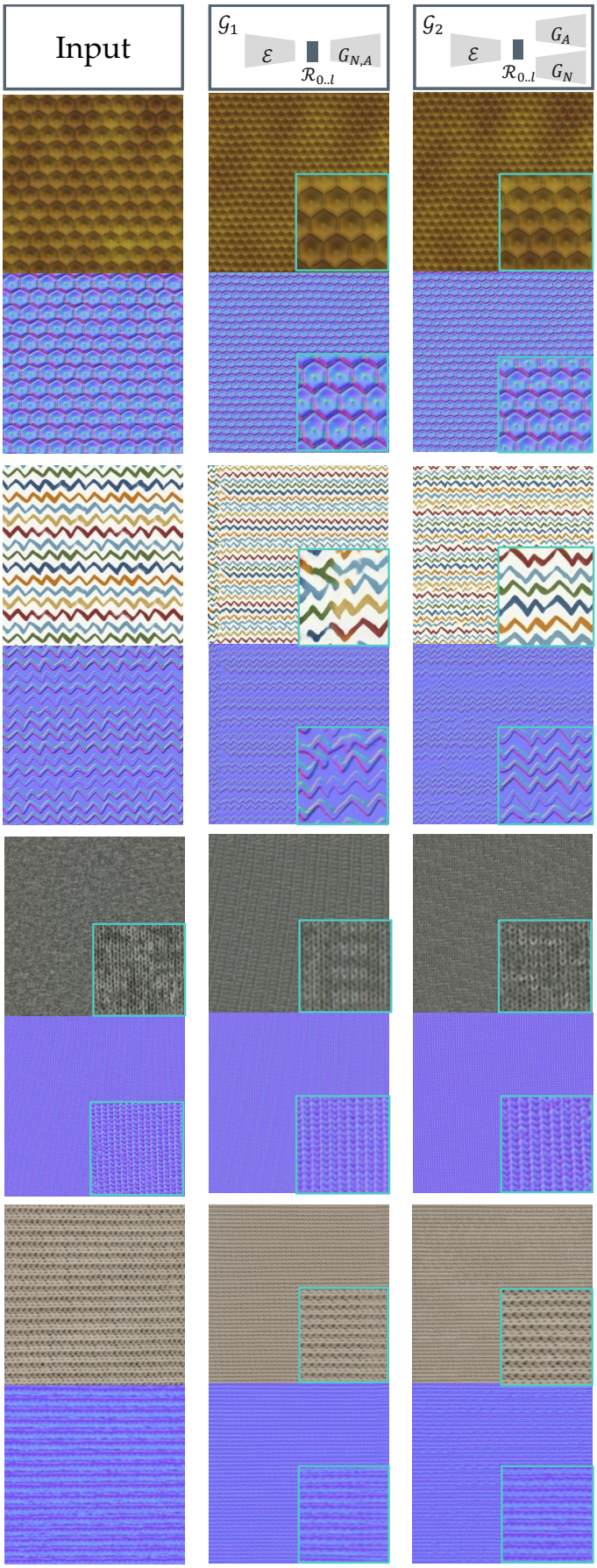


Fig. 1: Further results of our two proposed generator architectures $\mathcal{G}$. As shown, separating the decoders ultimately results in more detailed and varied maps. In blue, a closeup of the synthesized stacks, to help visualization.

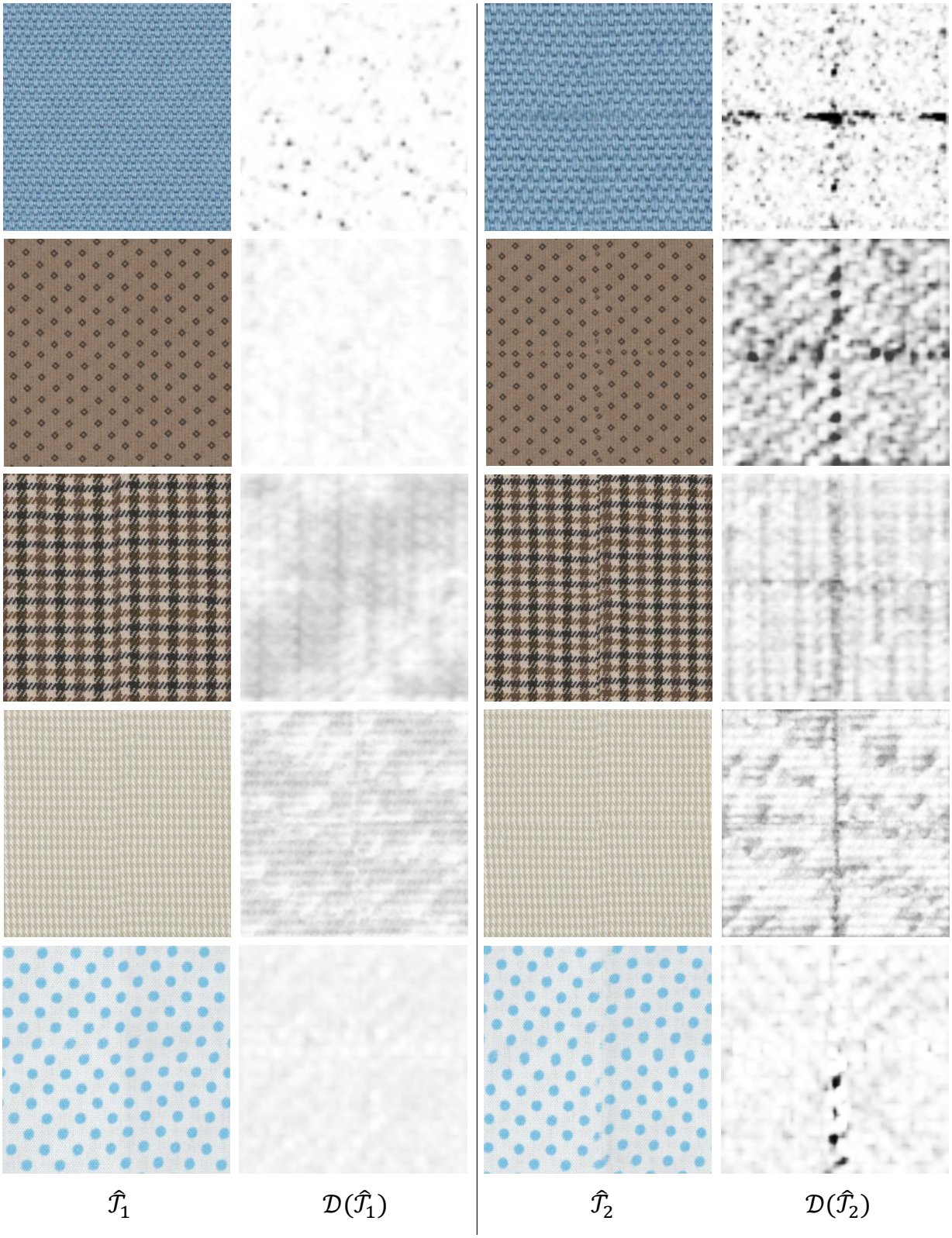


Fig. 2: More examples on the outputs of $\mathcal{D}$ with textures of different qualities. On the left, the generated albedos $\hat{\mathcal{T}}_1$ are artifact-free on the search areas, thus the discriminator ($\mathcal{D}(\hat{\mathcal{T}}_1)$) is fooled to believe $\hat{\mathcal{T}}_1$ are real. On their right, failed generated samples $\hat{\mathcal{T}}_2$ cannot fool the discriminator, which finds artifacts $\mathcal{D}(\hat{\mathcal{T}}_2)$.

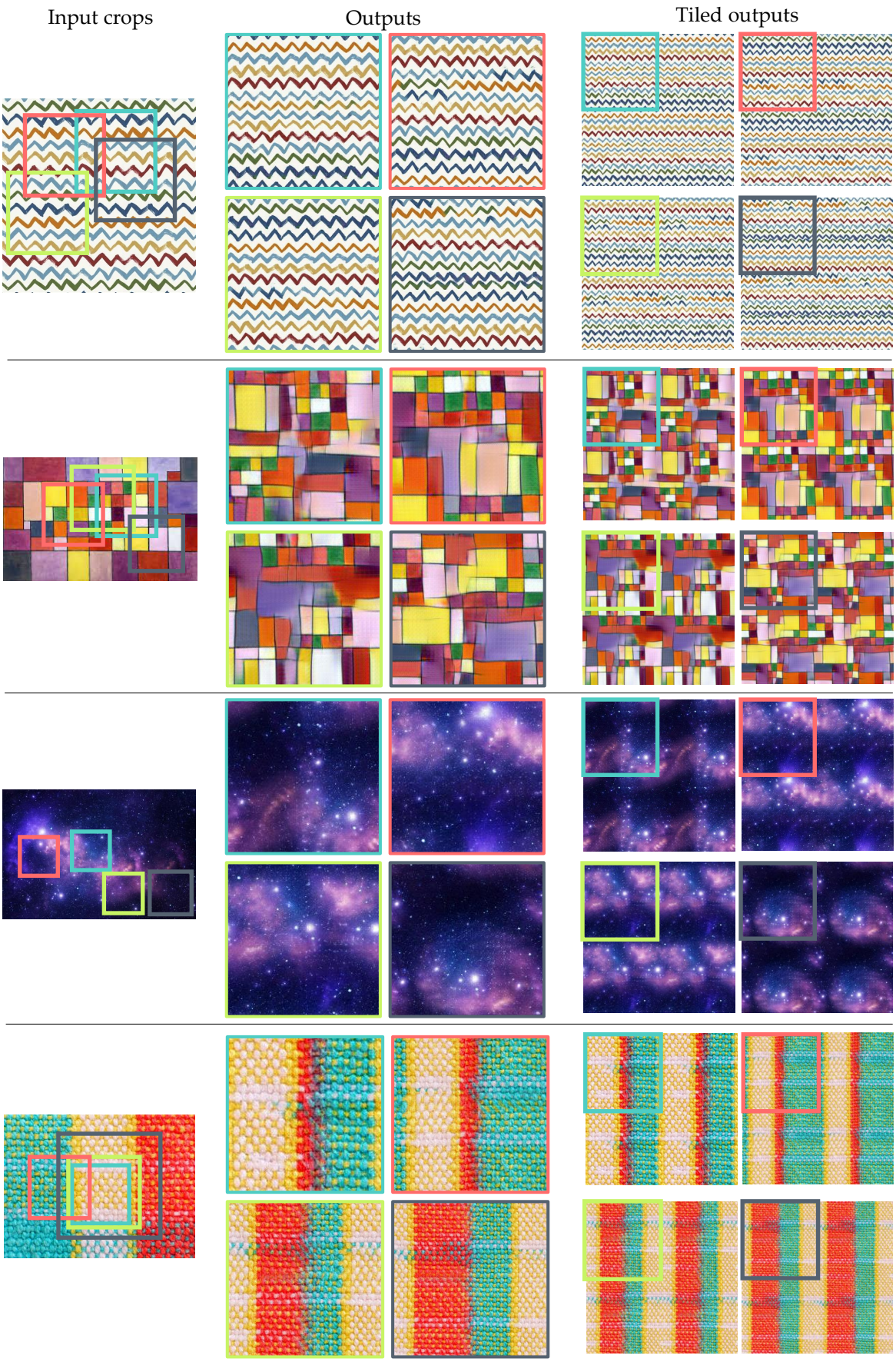


Fig. 3: More examples on the capabilities of SeamlessGAN on generating multiple tileable textures from a given input. On the left, we show the input sample $\mathcal{T}$ and four crops used for generating tileable maps. On the middle column, the outputs of the generative model are shown. On their right, the same textures are tiled 2×2 times for visualization purposes.

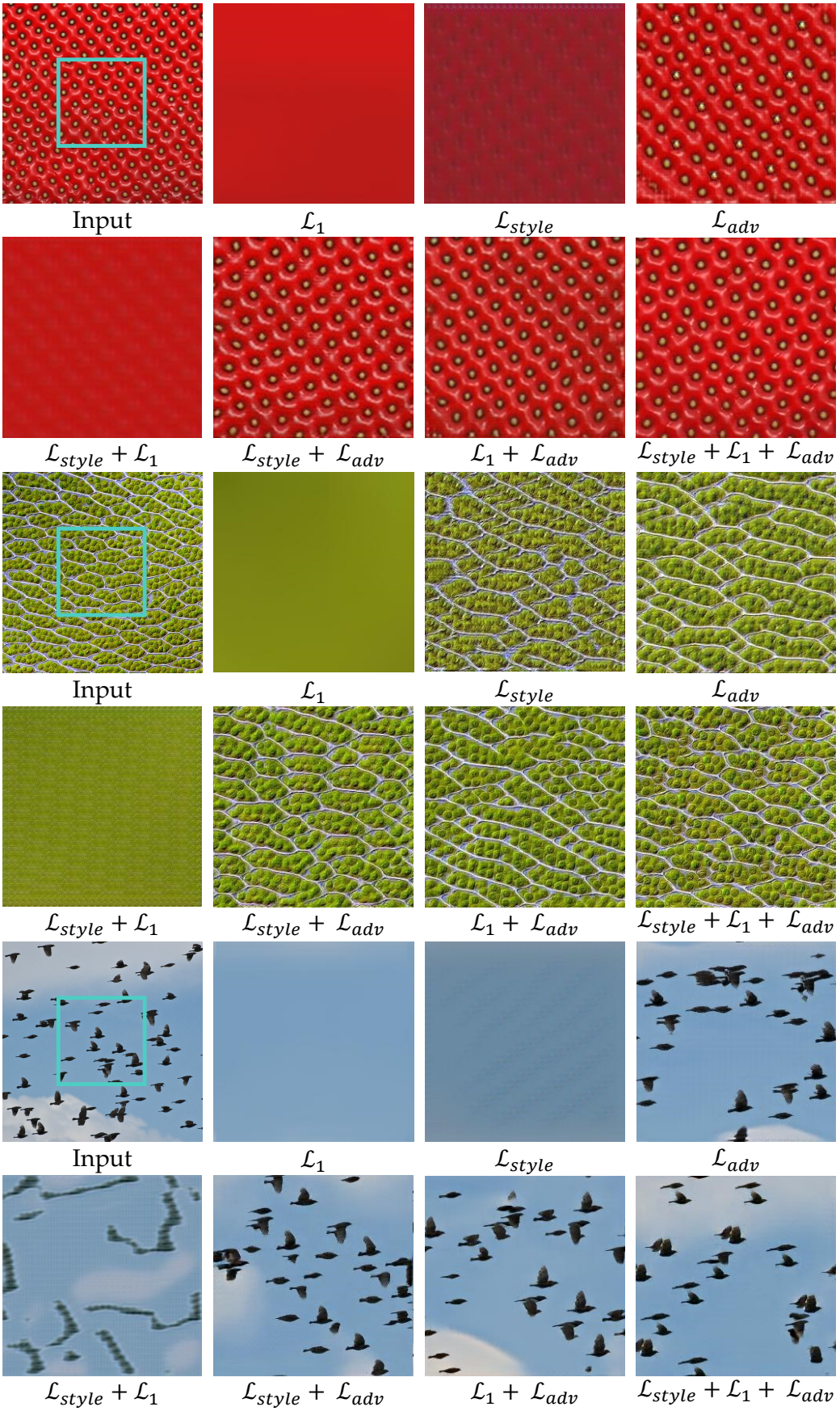


Fig. 4: Extended ablation study on the impact of the loss function on the quality of the synthesized textures. We evaluate each network using the same input, marked using a blue box. As shown, training $\mathcal{G}$ using the full loss function yields the best results.

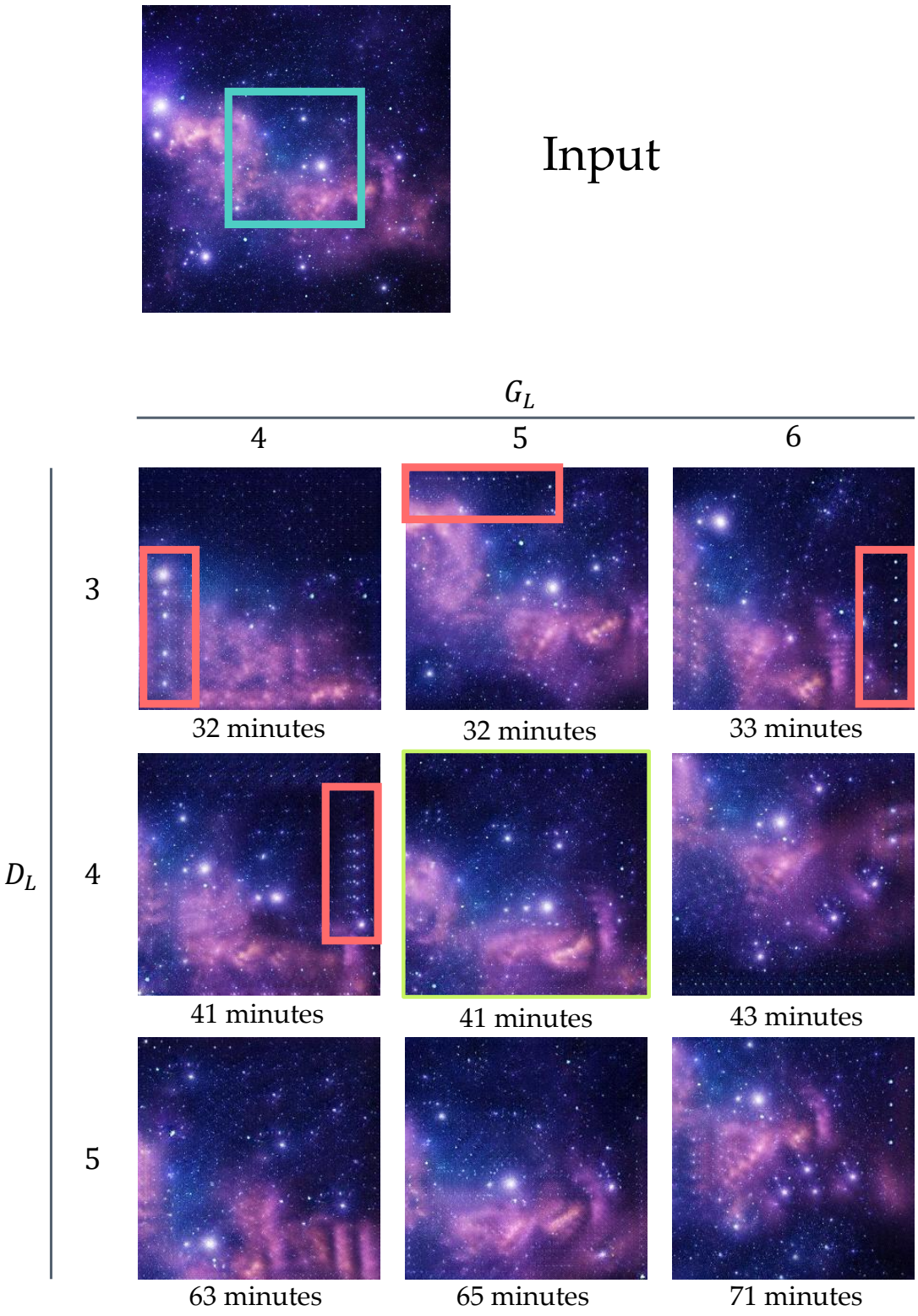


Fig. 5: An ablation study on the influence of the number of layers in each model in the quality of the generated outputs. We evaluate each network using the same input, marked using a blue box. We compare evaluate the influence of using discriminators with $L \in \{3, 4, 5\}$ layers, marked as D_L and generators with $L \in \{4, 5, 6\}$ residual blocks, marked as G_L. We show the configuration we use for every experiment in this paper using a green box. On the bottom of each image, we show the total training time for 50000 iterations. As shown, using a discriminator with 3 layers yields repetitive large-scale patterns, due to its reduced receptive field. Using $D_L = 5$ yields marginal improvements with respect to $D_L = 4$, with the cost of significantly increased runtimes. In terms of G_L, we find that there are no significant visual differences between using 5 or 6 residual blocks, but we favor the former due to its reduced training time. Artifacts highlighted in red boxes.

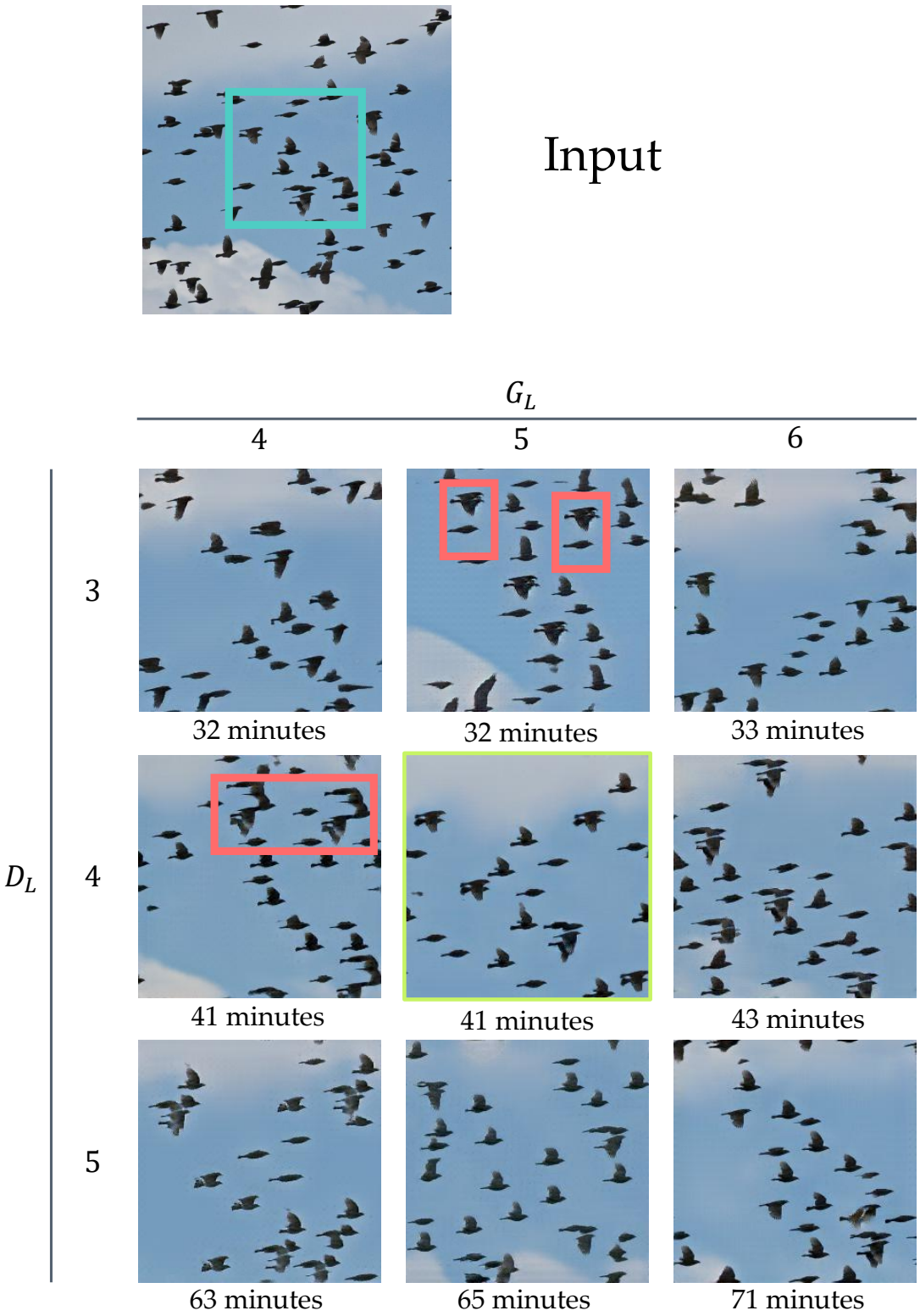


Fig. 6: An ablation study on the influence of the number of layers in each model in the quality of the generated outputs. We evaluate each network using the same input, marked using a blue box. We compare evaluate the influence of using discriminators with $L \in \{3, 4, 5\}$ layers, marked as D_L and generators with $L \in \{4, 5, 6\}$ residual blocks, marked as G_L. We show the configuration we use for every experiment in this paper using a green box. On the bottom of each image, we show the total training time for 50000 iterations. As shown, using a discriminator with 3 layers yields repetitive large-scale patterns, due to its reduced receptive field. Using $D_L = 5$ yields marginal improvements with respect to $D_L = 4$, with the cost of significantly increased runtimes. In terms of G_L, we find that there are no significant visual differences between using 5 or 6 residual blocks, but we favor the former due to its reduced training time. Artifacts highlighted in red boxes.

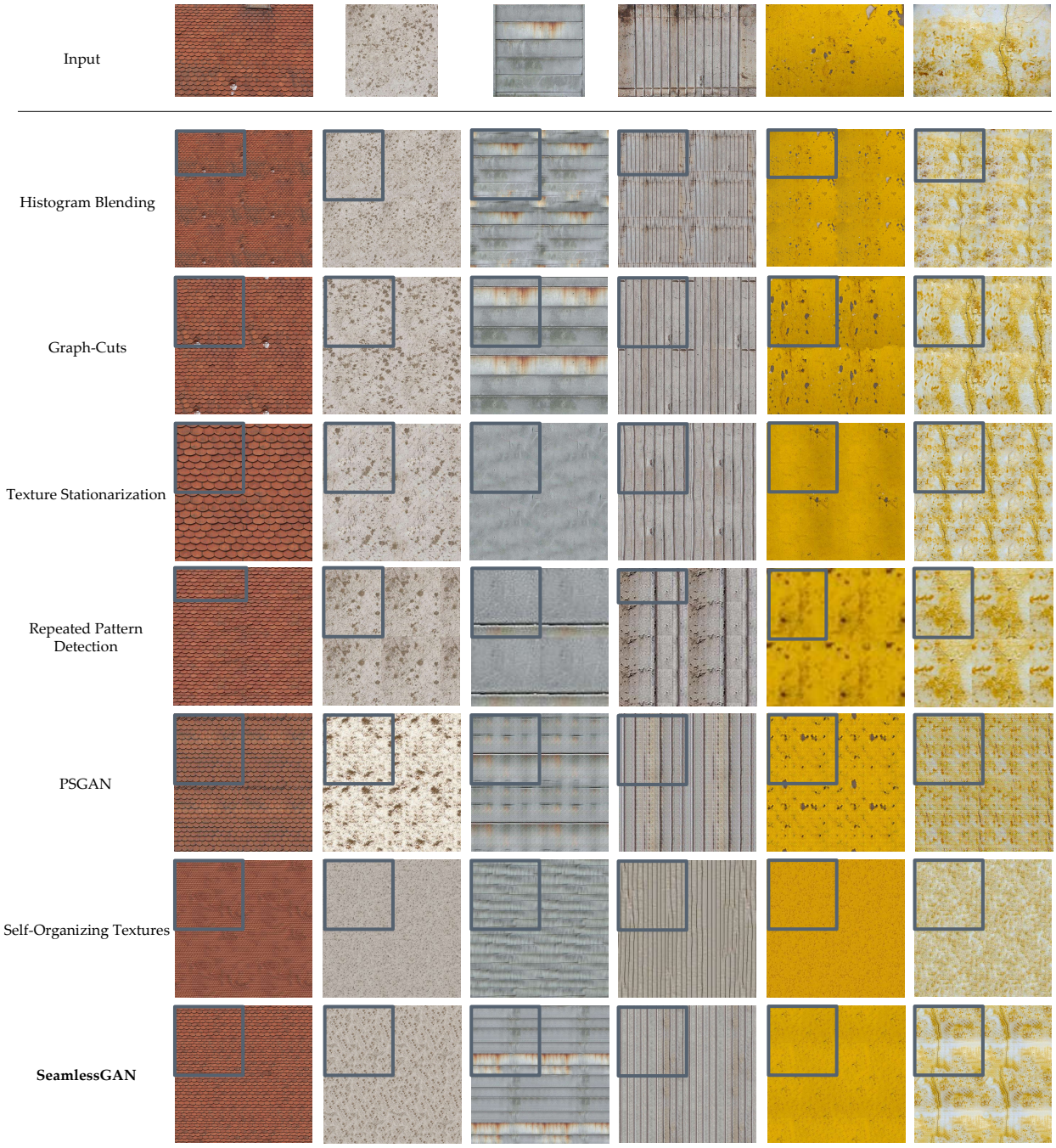


Fig. 7: Additional comparison with previous methods. From left to right, top to bottom: Input texture, the resuts of Histogram Blending, by Deloit *et al.* [14], the GraphCuts algorithm by Li *et al.* [15], Texture Stationarization by Moritz *et al.* [1], Repeated Pattern Detection by Rodriguez - Pardo *et al.* [16], PSGAN by Bergmann *et al.* [17], Self-Organizing Textures by Niklasson *et al.* [18]; and ours. Outputs are tiled a similar number of times (at least twice in each dimension) for better visualization. Our method generally captures better the overall structure of the texture, while providing seamless and semantically coherent borders, for enhancing tileability. From left to right, we sample $c = 27, 36, 29, 1, 17,$ and 7 input crops before obtaining a tileable texture.

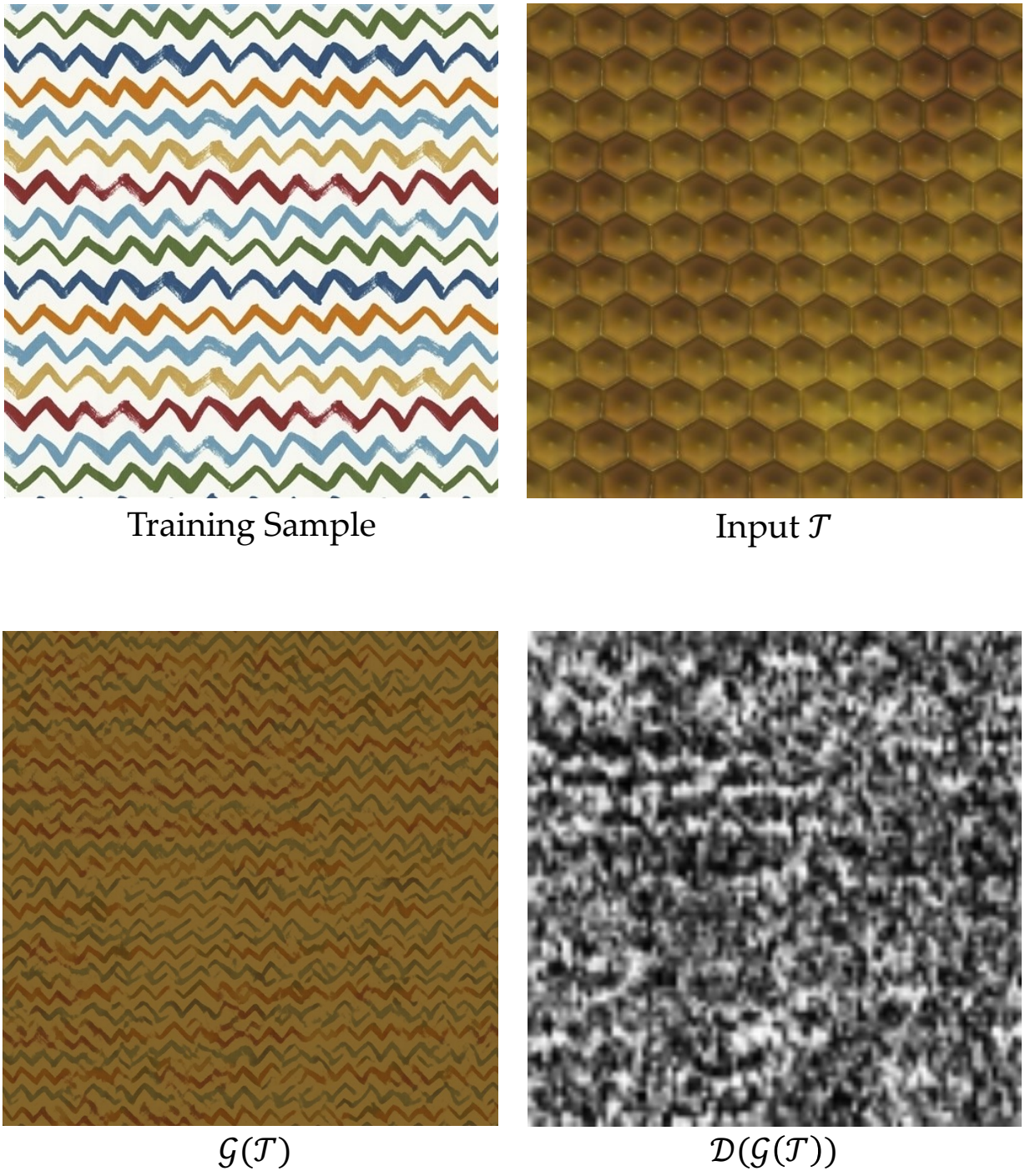


Fig. 8: An experiment on texture transfer using our framework. We train a SeamlessGAN on the Zigzag texture, and, during inference time, we evaluate it using a different image, the $\mathcal{T}$ honeycomb texture at the top. As shown, the results are not visually pleasing, as the generated texture only preserves the overall color present in the input sample, while the overall structure is a somewhat broken version of the training sample. Further, the discriminator outputs a very low overall probability of the texture being real, breaking the discriminator-guided sampling. The generated texture looks too fake to the discriminator, so it does not focus on small tileability artifacts. As such, we can argue that the SeamlessGAN framework does not work for tileable texture transfer applications.

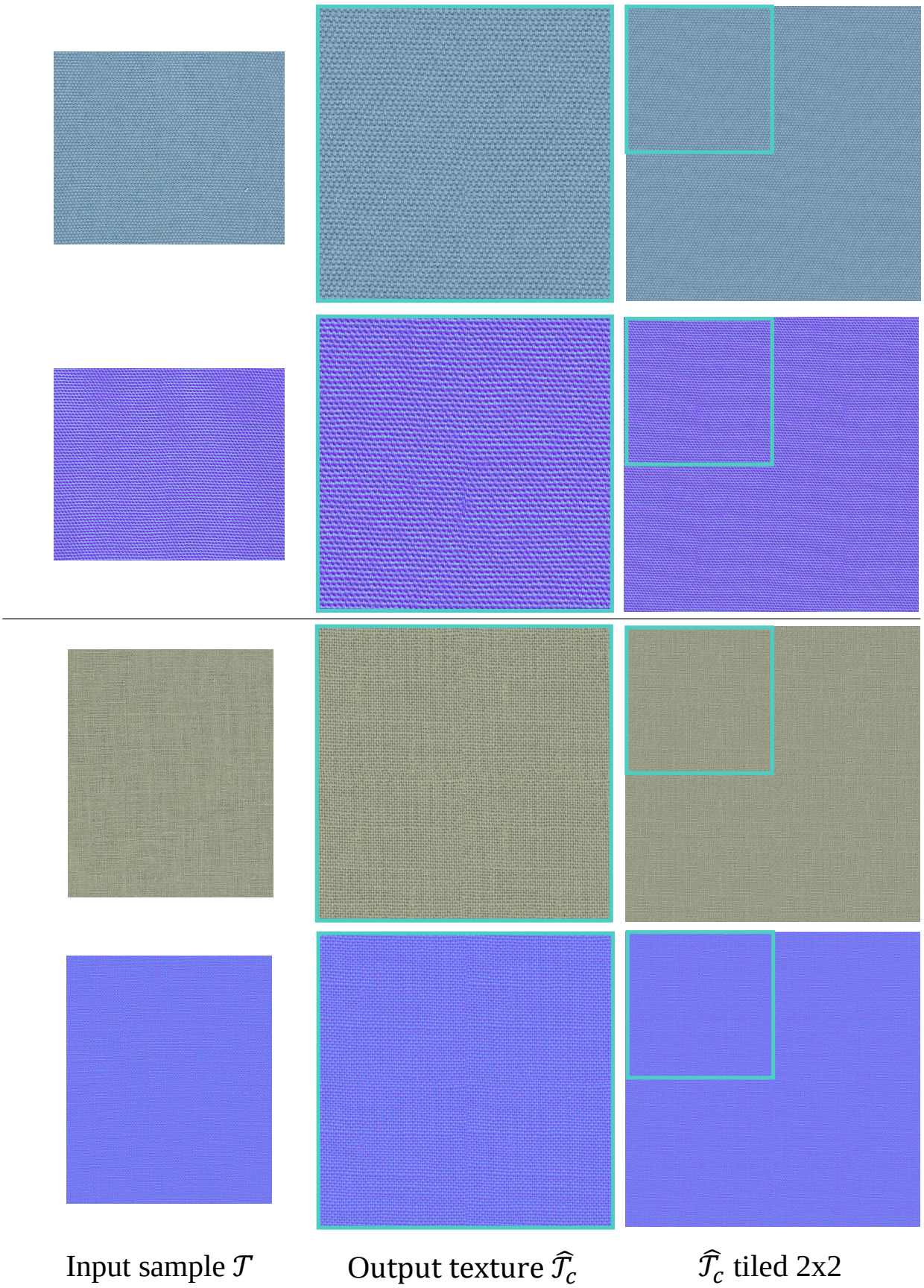
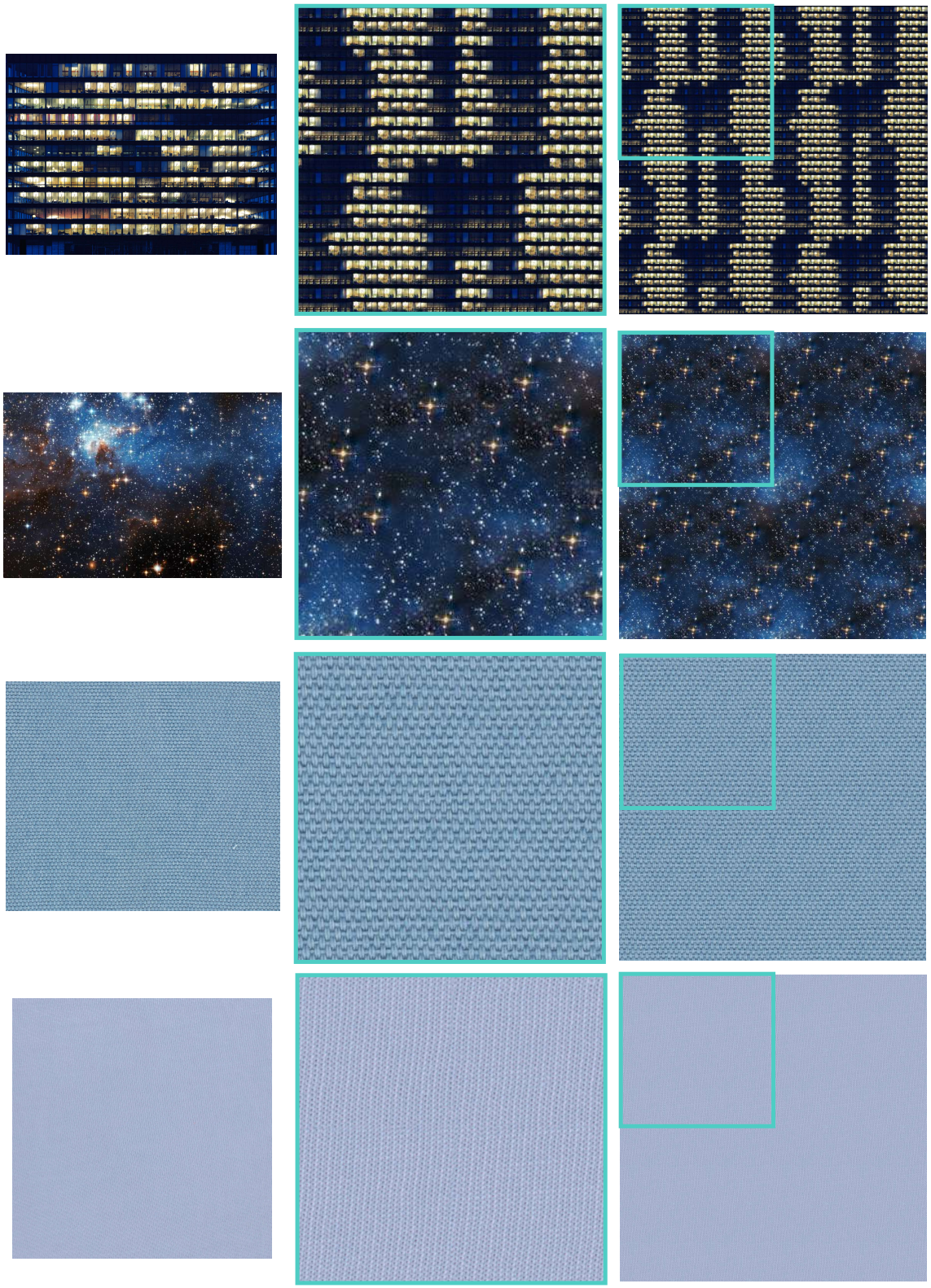
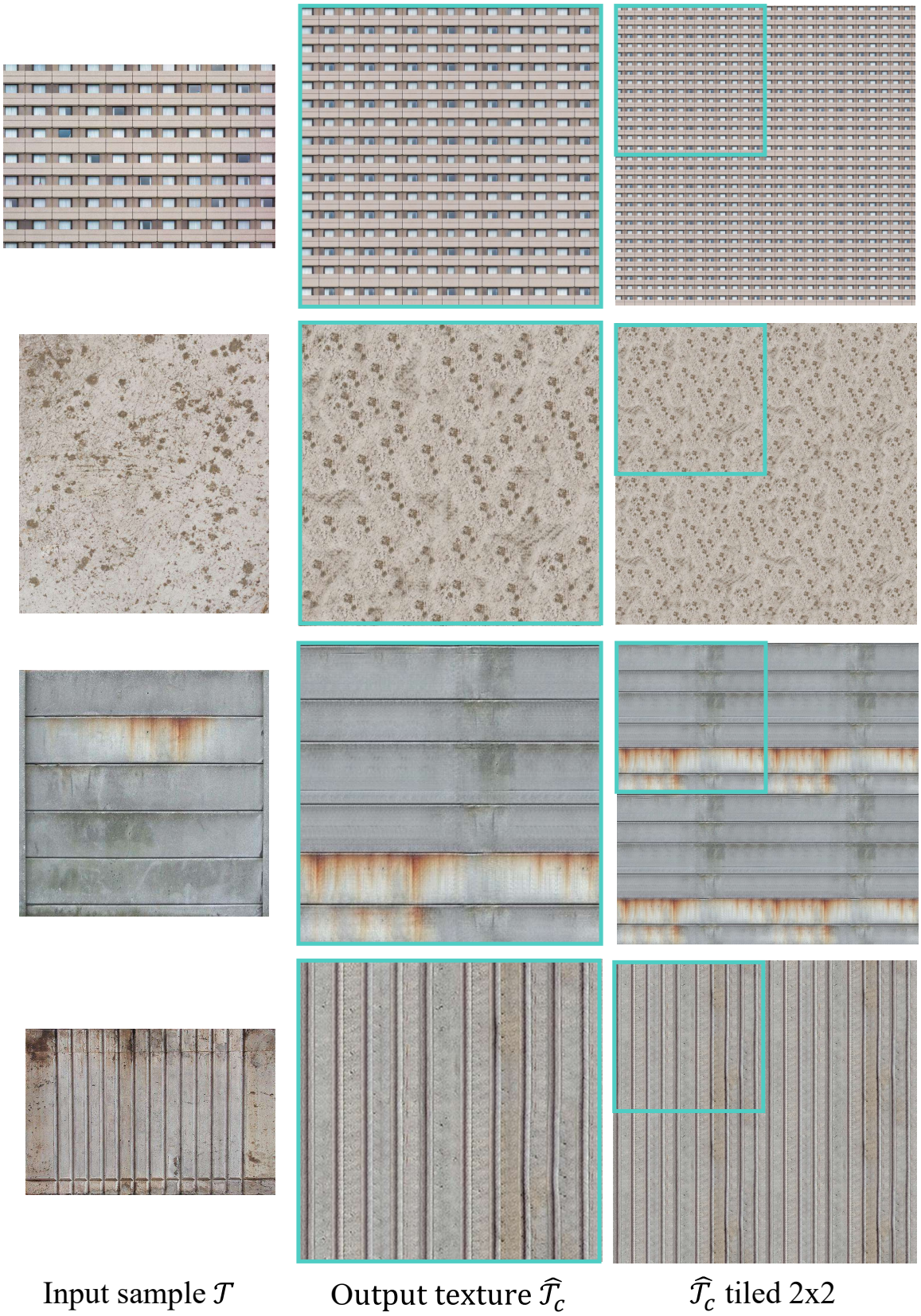
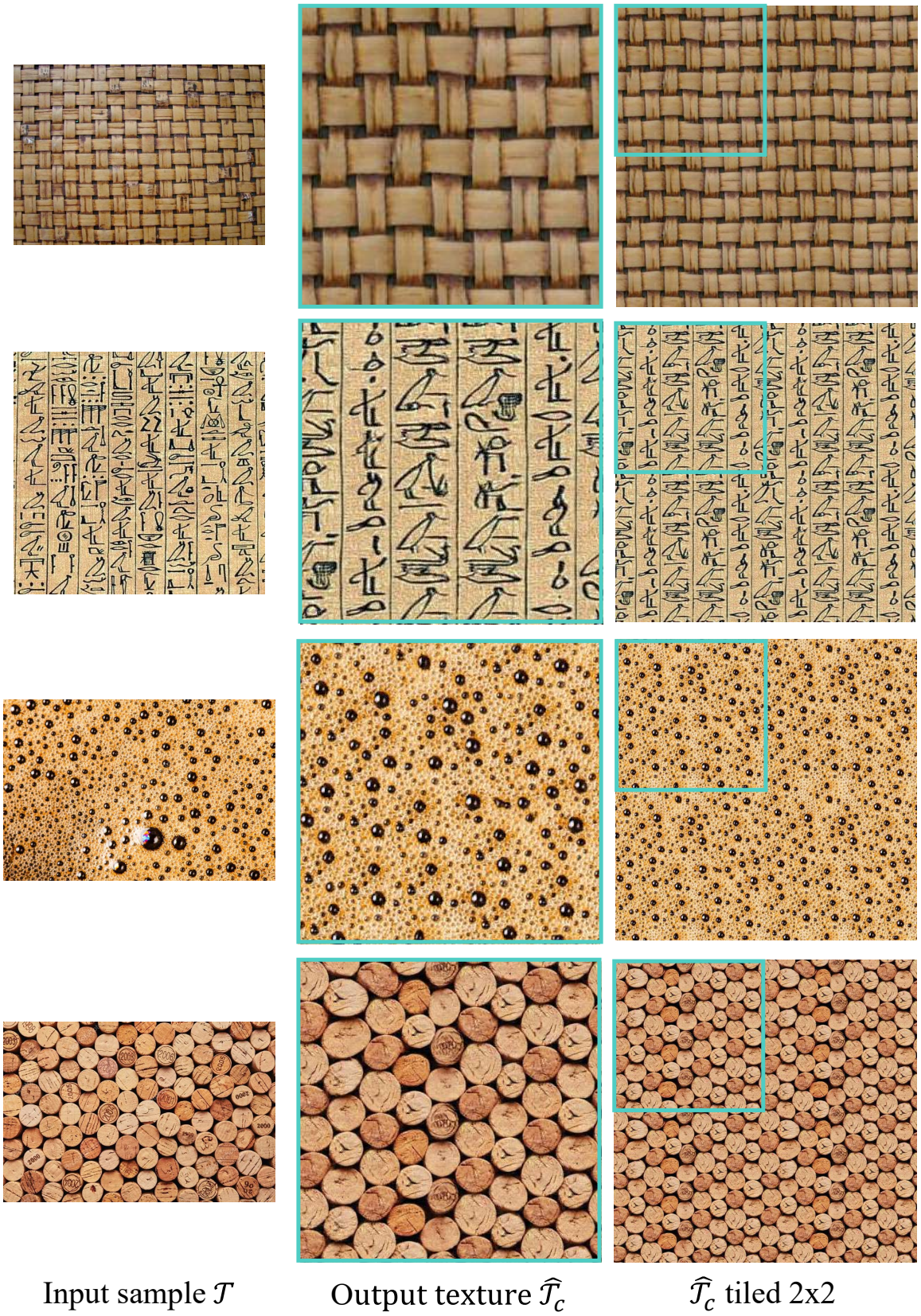


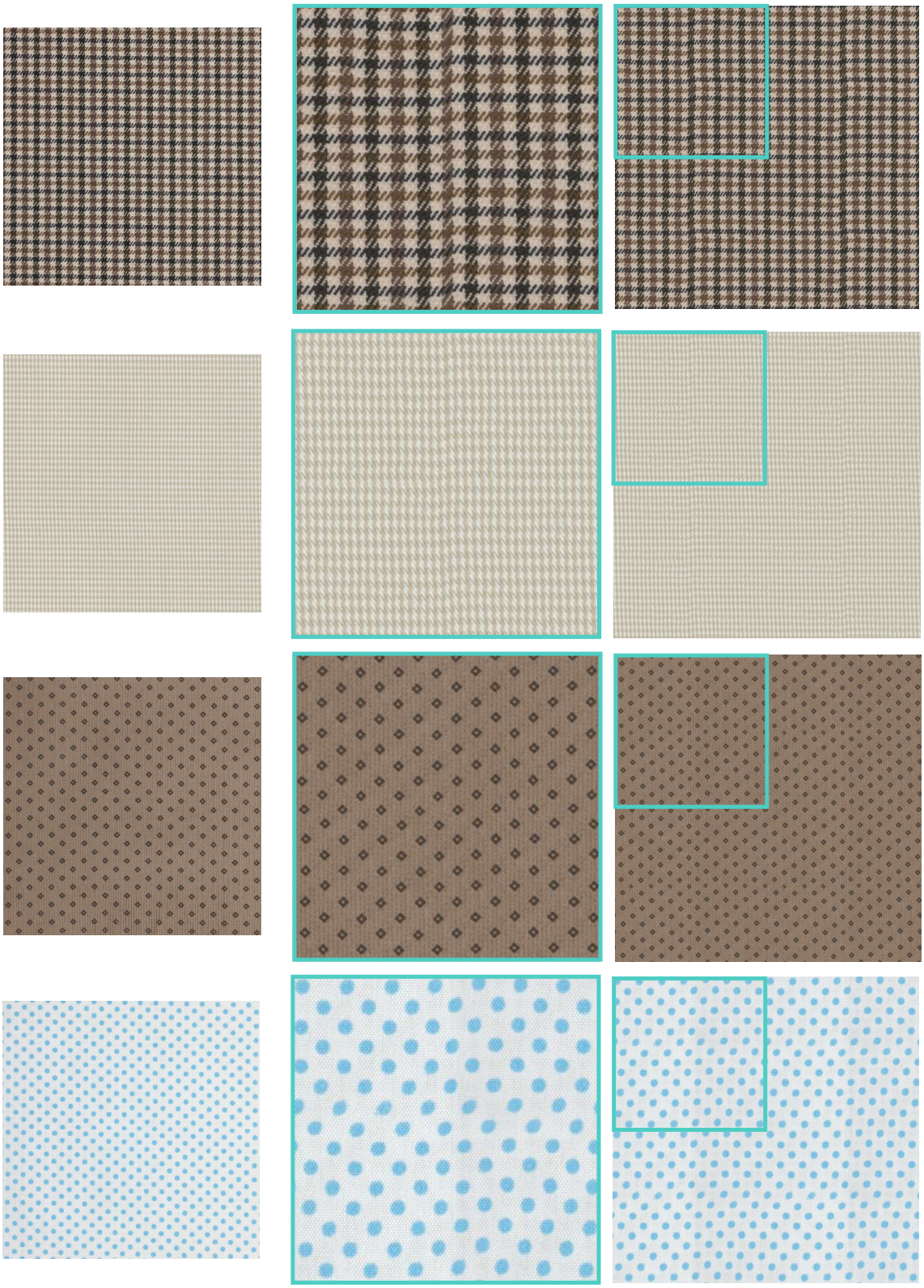
Fig. 9: Additional results on multi-layer tileable texture synthesis. Normals were captured using Photometric Stereo [22].


Input sample $\mathcal{T}$
Output texture $\widehat{\mathcal{T}}_c$
$\widehat{\mathcal{T}}_c$ tiled 2x2

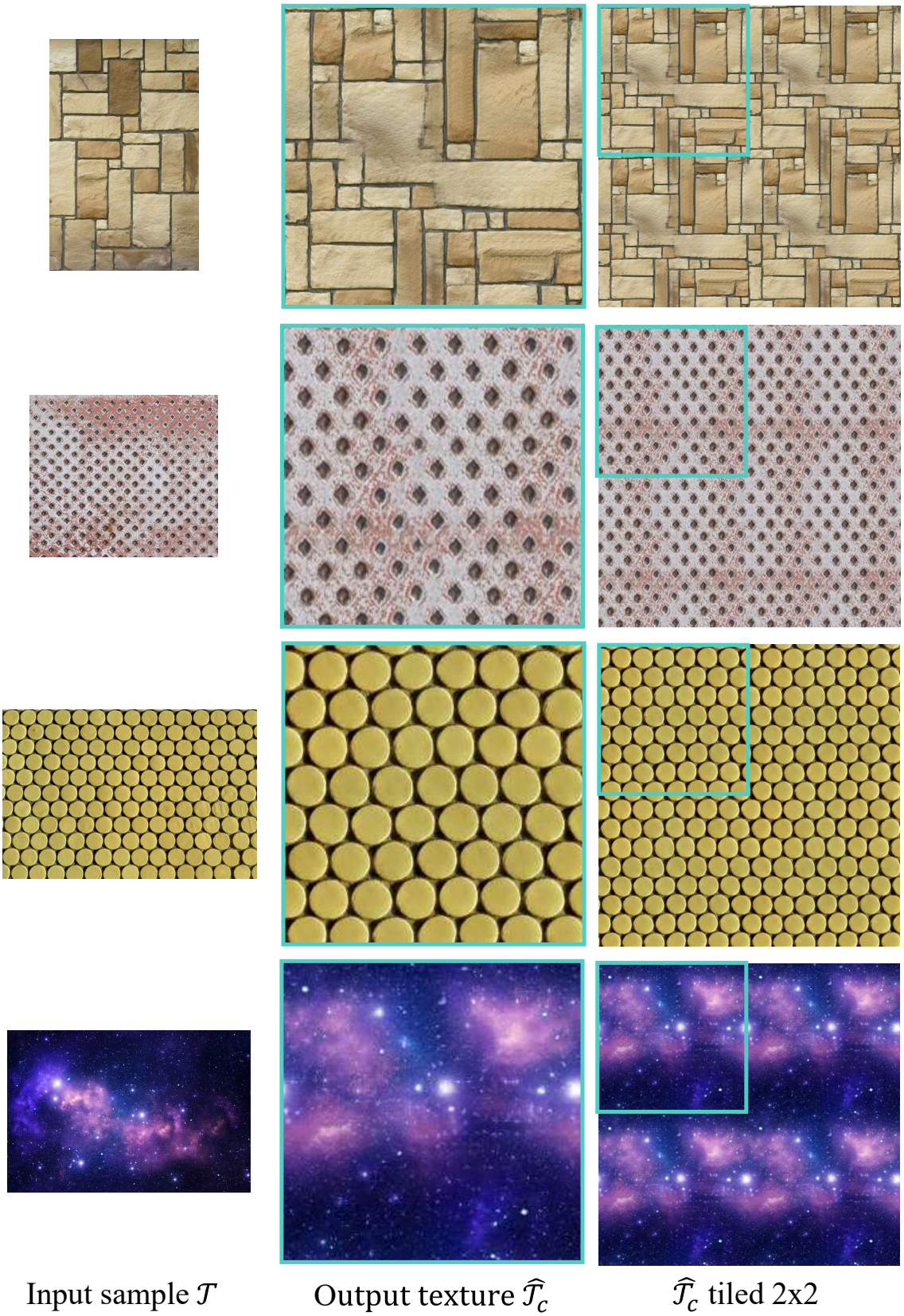

Input sample $\mathcal{T}$
Output texture $\widehat{\mathcal{T}}_c$
$\widehat{\mathcal{T}}_c$ tiled 2x2



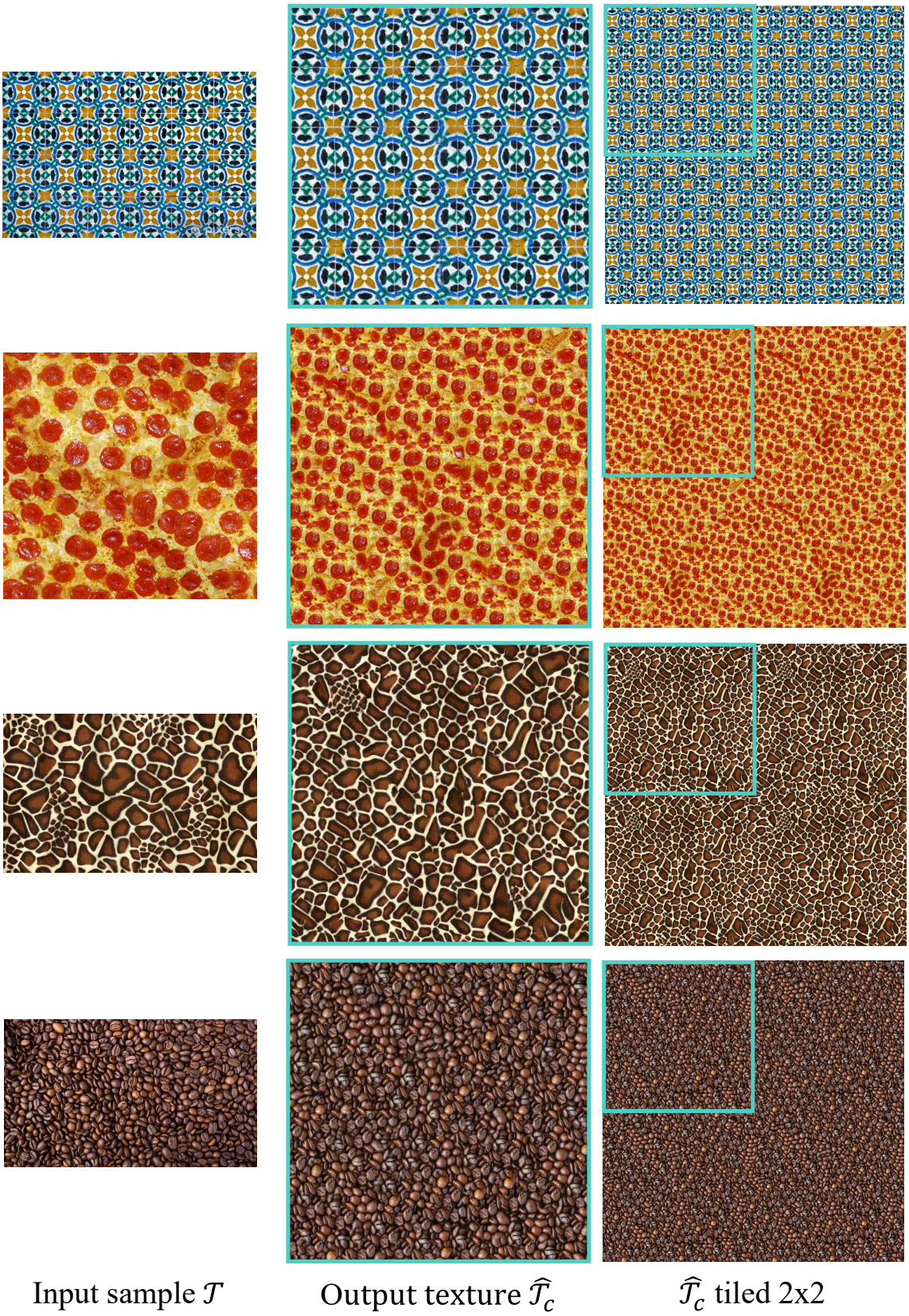
Input sample $\mathcal{T}$ Output texture $\widehat{\mathcal{T}}_c$ $\widehat{\mathcal{T}}_c$ tiled 2x2



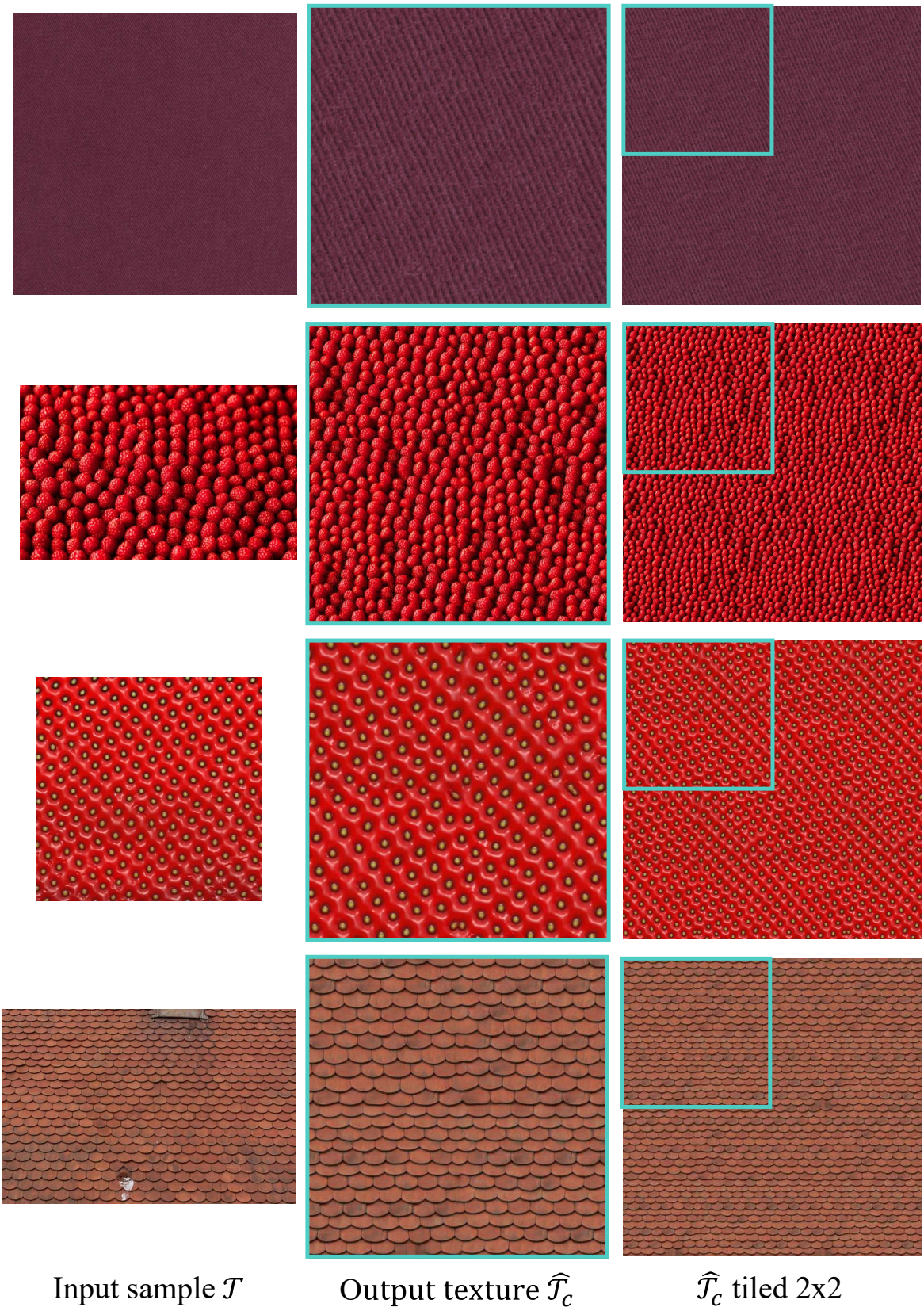
Input sample $\mathcal{T}$ | Output texture $\widehat{\mathcal{T}}_c$ | $\widehat{\mathcal{T}}_c$ tiled 2x2

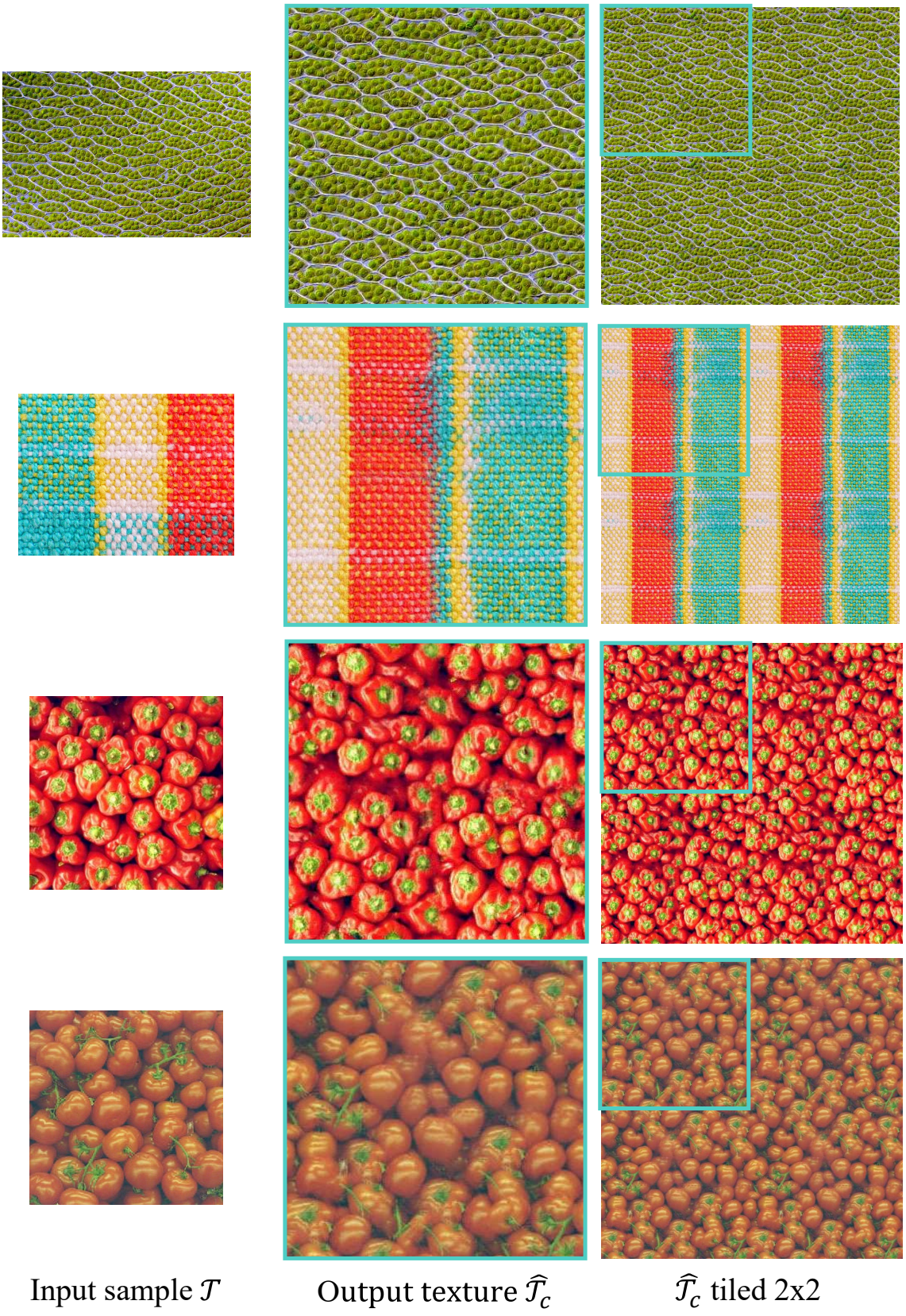


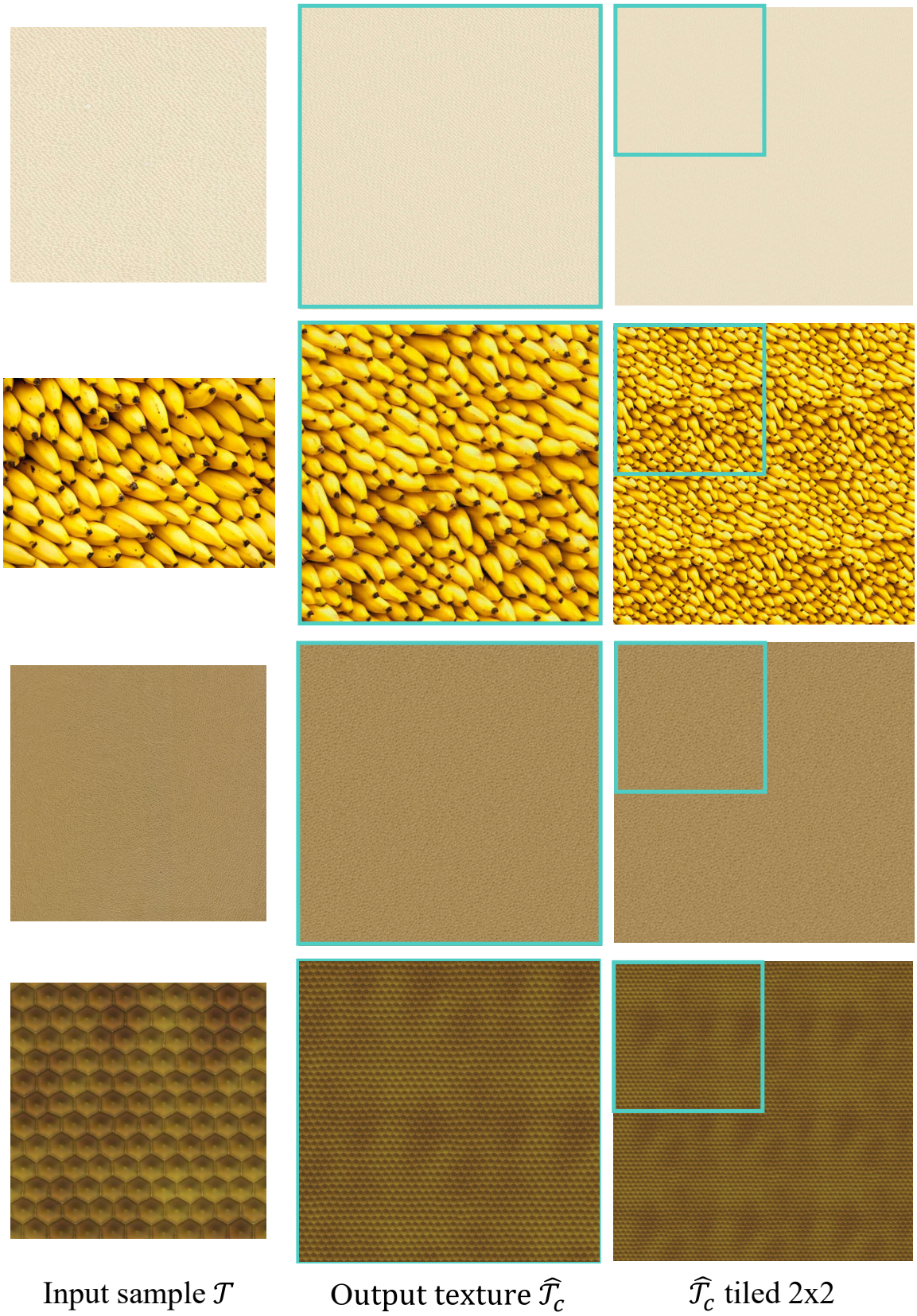
Input sample $\mathcal{T}$ Output texture $\widehat{\mathcal{T}}_c$ $\widehat{\mathcal{T}}_c$ tiled 2x2

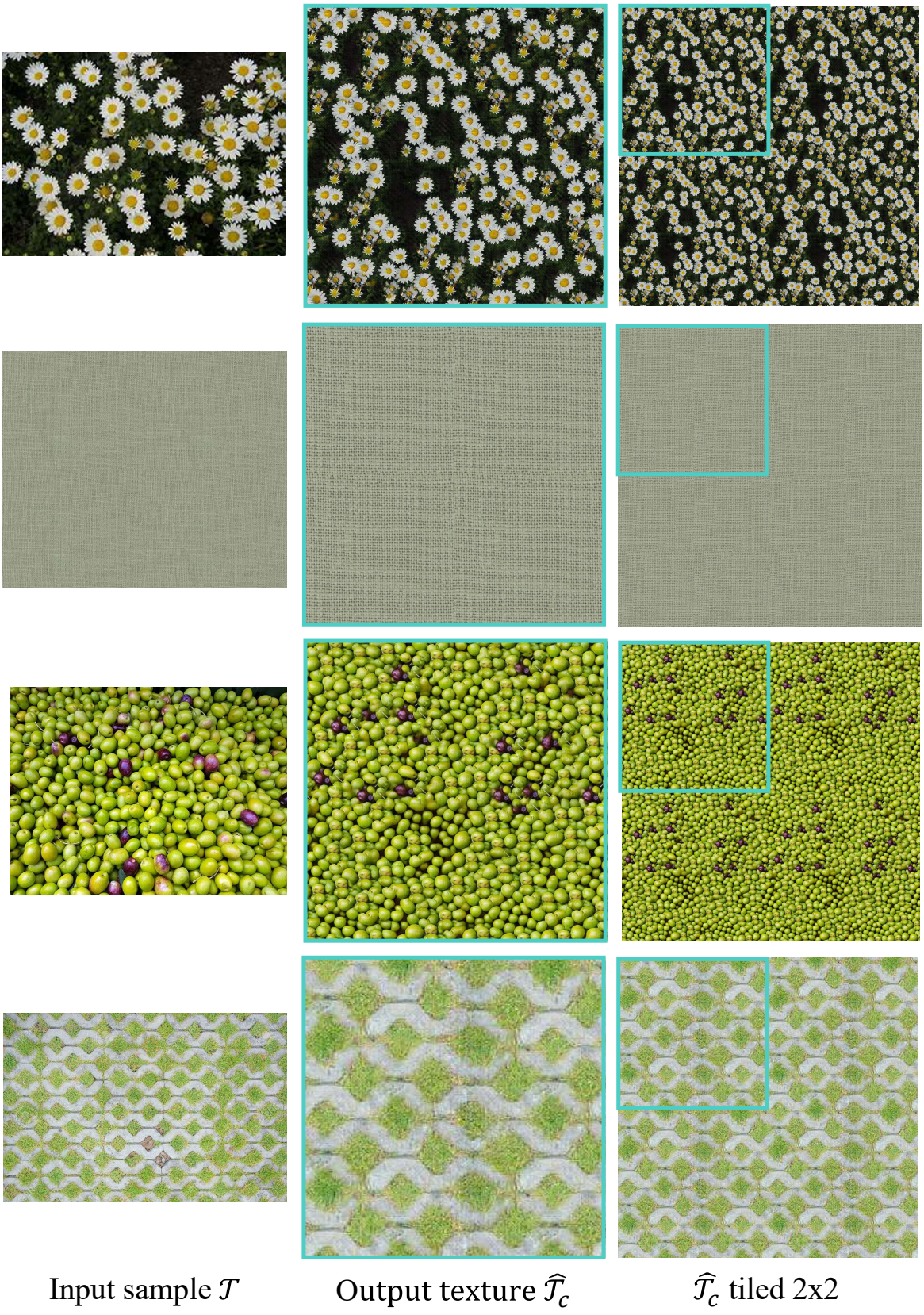


Input sample $\mathcal{T}$ | Output texture $\widehat{\mathcal{T}}_c$ | $\widehat{\mathcal{T}}_c$ tiled 2x2

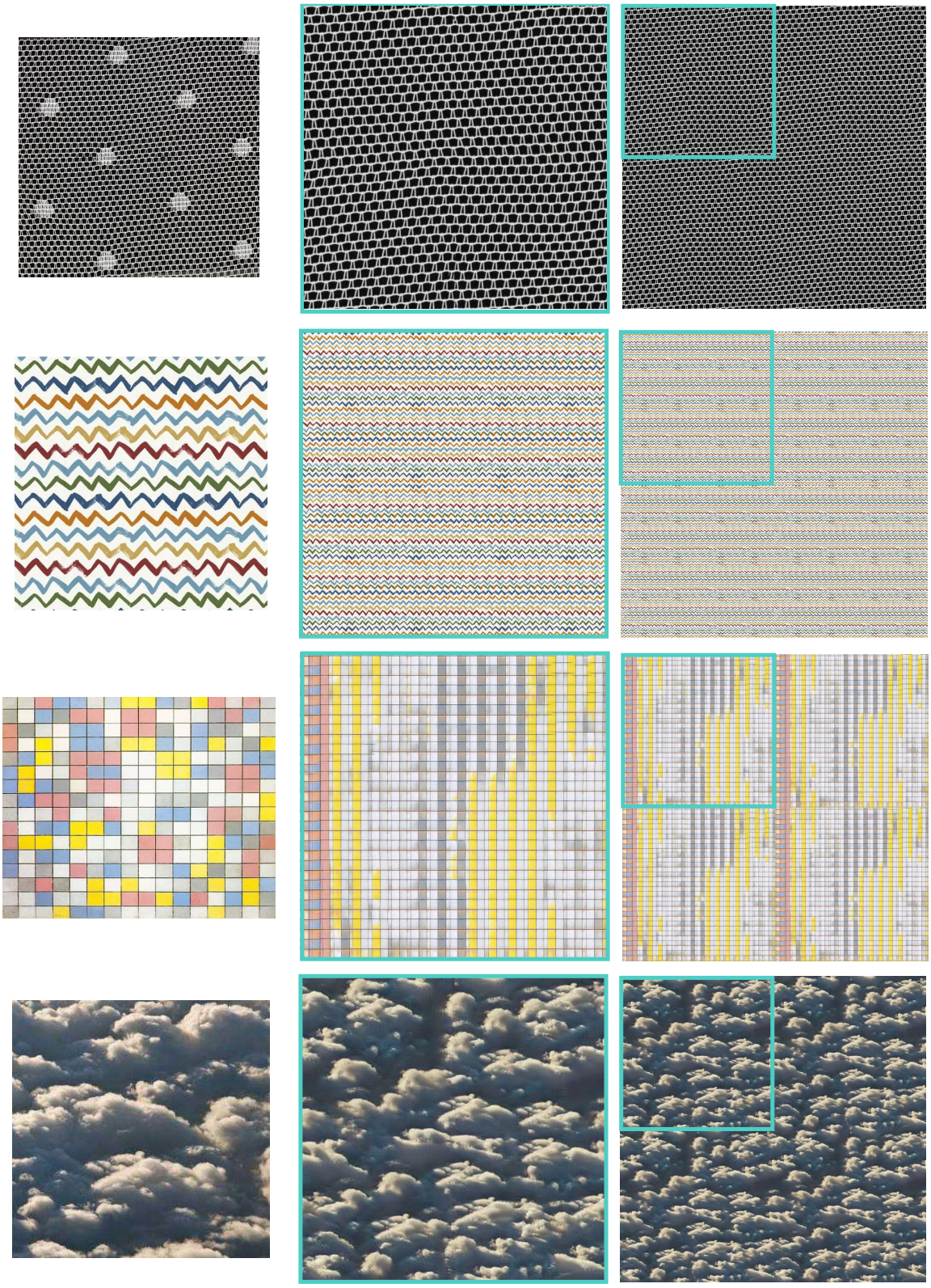

Input sample $\mathcal{T}$
Output texture $\widehat{\mathcal{T}}_c$
$\widehat{\mathcal{T}}_c$ tiled 2x2


Input sample $\mathcal{T}$
Output texture $\widehat{\mathcal{T}}_c$
$\widehat{\mathcal{T}}_c$ tiled 2x2


Input sample $\mathcal{T}$
Output texture $\widehat{\mathcal{T}}_c$
$\widehat{\mathcal{T}}_c$ tiled 2x2



Input sample $\mathcal{T}$ | Output texture $\widehat{\mathcal{T}}_c$ | $\widehat{\mathcal{T}}_c$ tiled 2x2



Input sample $\mathcal{T}$ Output texture $\widehat{\mathcal{T}}_c$ $\widehat{\mathcal{T}}_c$ tiled 2x2

REFERENCES

[1] J. Moritz, S. James, T. S. Haines, T. Ritschel, and T. Weyrich, "Texture stationarization: Turning photos into tileable textures," in *Eurographics Symposium on Geometry Processing*, vol. 36, no. 2, 2017, pp. 177–188.

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," 12 2019.

[3] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," Tech. Rep., 2015.

[4] K. Kurach, M. Lučić, X. Zhai, M. Michalski, and S. Gelly, "A large-scale study on regularization and normalization in gans," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3581–3590.

[5] J. Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, vol. 2017-Octob, pp. 2242–2251, 3 2017.

[6] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," in *International Conference on Learning Representations*, 2018.

[7] Y. Zhou, Z. Zhu, X. Bai, D. Lischinski, D. Cohen-Or, and H. Huang, "Non-stationary texture synthesis by adversarial expansion," *ACM Transactions on Graphics (ToG)*, vol. 37, no. 4, Jul. 2018.

[8] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2016-Decem, 2016, pp. 770–778.

[10] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[11] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5967–5976.

[12] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2017-Janua, pp. 105–114, 9 2017.

[13] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings on the International Conference on Machine Learning*, vol. 30, no. 1. Citeseer, 2013, p. 3.

[14] T. Deliot and E. Heitz, "Procedural stochastic textures by tiling and blending," *GPU Zen*, vol. 2, 2019.

[15] Z. Li, M. Shafiei, R. Ramamoorthi, K. Sunkavalli, and M. Chandraker, "Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and svbrdf from a single image," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 2475–2484.

[16] C. Rodriguez-Pardo, S. Suja, D. Pascual, J. Lopez-Moreno, and E. Garces, "Automatic extraction and synthesis of regular repeatable patterns," *Computers and Graphics (Pergamon)*, vol. 83, pp. 33–41, 10 2019.

[17] U. Bergmann, N. Jetchev, and R. Vollgraf, "Learning texture manifolds with the periodic spatial gan," pp. 469–477, 2017.

[18] E. Niklasson, A. Mordvintsev, E. Randazzo, and M. Levin, "Self-organising textures," *Distill*, 2021, https://distill.pub/selforg/2021/textures.

[19] L. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 262–270.

[20] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky *et al.*, "Theano: A python framework for fast computation of mathematical expressions," *arXiv e-prints*, pp. arXiv–1605, 2016.

[21] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[22] K. Ikeuchi, "Determining surface orientations of specular surfaces by using the photometric stereo method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 6, pp. 661–669, 1981.