Supplementary Material of Scraping Textures from Natural Images for Synthesis and Editing

Xueting Li[1], Xiaolong Wang[2],
Ming-Hsuan Yang[1], Alexei A. Efros[3], and Sifei Liu[4]

[1] UC Merced [2] UC San Diego [3] UC Berkeley [4] NVIDIA

1 Overview

In this supplementary, we provide more details about the implementation and experiental results of the proposed method. We start by describing implementation details including network architecture, objective details, data augmentation mechanism and hyperparameters in Section 2. We then validate our design choices by carrying out ablation studies on different components of our method in Section 3. In Section 4, we present a detailed comparison and differentiation with other state-of-the-arts texture synthesis methods. More qualitative results on arbitrary-sized texture synthesis and texture editing are demonstrated in Section 6. Finally, we discuss limitations of our work in Section 7.

2 Implementation Details

2.1 Network Architecture

We show the detailed framework of our method in Fig. 1.

Encoder Our encoder shown in Fig. 1(a) is inspired by the encoder in [7], which includes six residual blocks. It takes an image $I \in \mathcal{R}^{3\times H\times W}$ as input and first encodes it into a feature map $F \in \mathcal{R}^{3\times \frac{H}{8}\times \frac{W}{8}}$. The grouping module shown in Fig. 1(c) takes F as input and outputs the grouping mask $M \in \mathcal{R}^{K\times H\times W}$ and the texture code $t^k \in \mathcal{R}^{256}, k = 1, \ldots, K$ for each group.

Decoder The decoder shown in Fig. 1(b) takes each texture code t^k as input and predicts the frequency of the sine wave S^k for each group k. Then we concatenate the sine wave with a random Gaussian noise image of equal size and feed the concatenation into a stack of SPADE residual blocks [11] for either reconstruction or texture synthesis, as discussed in Section 3.3 and 3.4 in the submission, respectively. We note that the sine wave is a rigorous periodic function, thus we concatenate it with a noise image before feeding into the SPADE blocks such that the network can better fit for irregular textures (e.g., grass, water, etc.)

2.2 Adversarial Loss with Masks

We provide a detailed mathematical formulation of integrating masks in the adversarial loss in Section 3.5 in the manuscript. Given a synthesized texture image and the texture segment in the input image, we use adversarial training to encourage our model to synthesize more realistic texture images. We denote the synthesized texture image as $\hat{I}_t$, the foreground mask of the texture segment as M_t and a VGG network [13] pretrained on the ImageNet dataset [4,9] as Φ. We sum the Gram matrix matching loss on feature maps from multiple layers (i.e., *relu3_1*, *relu4_1*, *relu5_1*) of the VGG together. At each layer l, we resize the mask M_t to match the size of the feature map at layer l and denote the resized mask as M_t^l. The Gram matrix matching objective at layer l is computed as:

$$\mathcal{L}_{Gram}^{l} = \left| \frac{\Phi_l^T(\hat{I}_t) \cdot \Phi_l(\hat{I}_t)}{H_l W_l} - \frac{\Phi_l^T(I, M_t^l)\Phi_l(I, M_t^l)}{\sum M_t^l} \right| \tag{1}$$

where $\Phi_l(\hat{I}_t) \in \mathcal{R}^{C_l \times H_l \times W_l}$ is the feature map of the synthesized texture at layer l, $\Phi_l(I, M_t^l)$ represents extracting the features of the input image from only the foreground region defined by M_t^l, avoiding introducing irrelevant texture features into the Gram matrix computation. The final Gram matrix matching loss is computed as:

$$\mathcal{L}_{Gram} = \sum_{l=0}^{3} L_{Gram}^{l} \tag{2}$$

2.3 Training Details

Data Augmentation For each training image, we utilize random scaling and cropping as data augmentation. During training, we randomly scale the image such that its shorter edge is within the range of $224 \sim 268$. We then randomly crop a 224×224 image as a training sample. During inference, our network is able to synthesize arbitrary-sized texture image by extending the periodic sine wave, as shown in Fig. 6, Fig. 7 and Fig. 8.

Hyperparameters For each input image, we use the SLIC [1] method to obtain 196 superpixels and cluster them into 10 groups. We represent the texture in each group with a 256-dimensional texture code and a 32-channel parametric sine wave. All objectives, including the L1, LPIPS, Gram Matrix matching, have equal weight, i.e., 1.0. The temperature in the SoftMax is set to 23 to encourage less ambiguous assignments. We start the training with a batch size of twelve and decrease the batch size to four after adding the texture synthesis task to fit the GPU memory. The training process takes about two days on four 12GB GPUs. For each unseen image, we fine-tune the model for 5000 iterations with the same objectives (i.e., L1, LPIPS, focal frequency loss, Gram Matrix matching and patch GAN loss).

3 Ablation Studies

3.1 Different Group Number

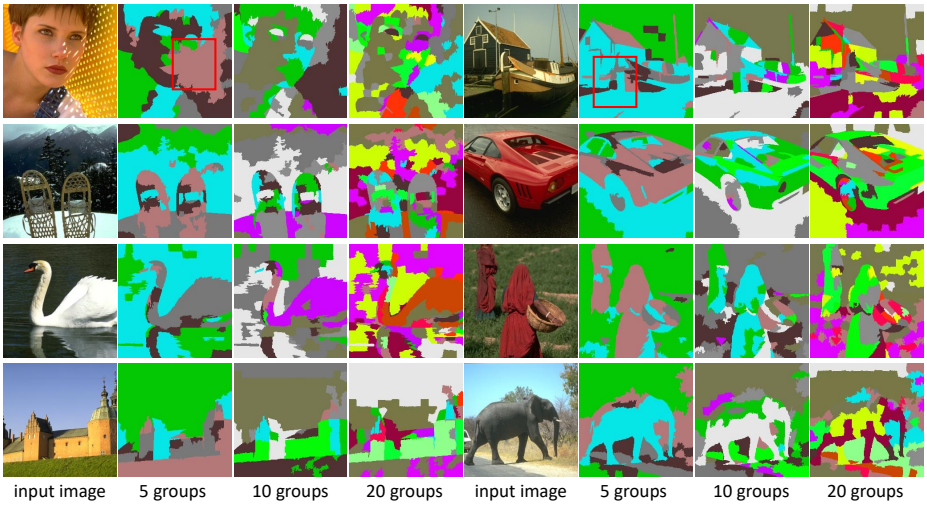

Fig. 2: **Grouping mask visualization for different group numbers.**

We compare models trained with different group number and show qualitative and quantitative results in Fig. 2 and Table 1, respectively. With less group number, our model aggressively cluster similar pixels together. This produces more clean an compact grouping masks. However, it may cluster pixels of different textures together, as shown in the red boxes in Fig. 2. This is also validated by the Boundary Recall (BR) and Boundary Precision (BP) and Achievable Segmentation Accuracy (ASA) score in Table 1. Given more groups, our model tends to "over-segment" the input image. Pixels of the same texture may be clustered into different groups due to lighting or color change. While this potentially increases the BR, BP and ASA score as shown in Table 1, this may reduce the size of texture segment and provide inferior supervision for the texture synthesis learning.

Groups	BR ↑	BP ↑	ASA ↑
5	0.61	0.28	0.61
10	0.68	0.28	0.67
20	0.72	0.24	0.71

Table 1: **Ablation study on group number.** "BR" stands for Boundary Recall, "BP" stands for Boundary Precision and "ASA" stands for Achievable Segmentation Accuracy.

3.2 Training on a Single Image from Scratch

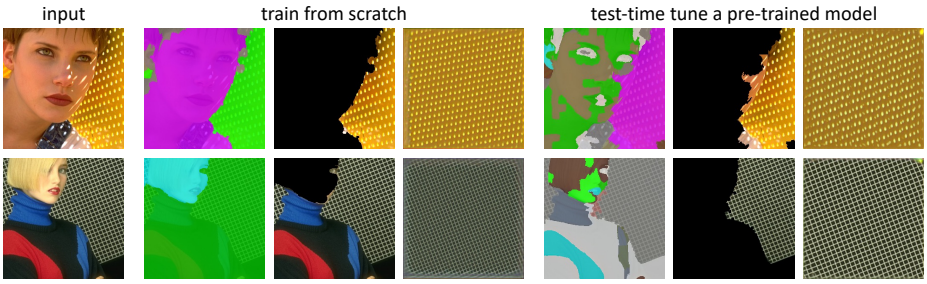

Fig. 3: **Ablation study on model pre-training.**

To resolve the capacity limitation of neural networks, few texture synthesis works [12,8,14] train a separate network for different texture image. However, we show in Fig. 3 that a single image does not provide sufficient prior knowledge to learn a reasonable grouping model. As shown in Fig. 3, when we train the model on a single image from scratch, the model fails to produce decent texture segments for texture learning. Thus, we first train our model on a set of natural images and then test-time adapt it to an unseen image for texture synthesis. We note that the test-time adaptation process in our method is straightforward and efficient to carry out since it does not require any form of supervision from the unseen images.

4 Detailed Comparison with State-of-the-art Methods

Besides Fig.3 in the manuscript, we show more texture synthesis results by our method and the baseline methods [10,8,6,2,12,14] in Fig. 4. The most *significant difference* between the proposed method and all the baseline methods is that our method directly learns texture synthesis from cluttered natural images without any form of supervision, while all baseline methods require clean rectangular texture patches to synthesize high-quality texture images. We discuss and compare to each baseline method in details in the following.

PSGAN The PSGAN [2] is a generative texture synthesis method. It takes the concatenation of a global random noise vector, a spatial random noise image and a sine wave as input to synthesize a texture image. Though PSGAN is able to capture a handful of textures in a single image without supervision, it relies on the strong assumption that the image is ultra-resolution and each randomly cropped training patch only includes a single texture pattern. However, as shown in Fig. 4, such strong assumption easily breaks for normal-sized images. Besides, since PSGAN is a generative model, it is not fit for controlled texture synthesis and the model may fail to capture all texture patterns in the image. On the contrary, our model explicitly decomposes the image into K groups and model

the texture pattern in each group. We note that the PSGAN[5] does not converge when trained on cropped texture patches in Fig. 4 (e). Thus, we replicate the texture patches to form a large image for training. This operation stabilizes training but introduces repeated pattern artifacts as shown in Fig. 4(i).

Image Quilting As a non-parametric texture synthesis method, Image Quilting [6] sequentially synthesizes a texture image by searching and copying a patch that best blends in the current local context. Such a mechanism bypasses the challenging task of modeling inherent texture statistics and synthesizes realistic texture images. However, as shown in Fig. 4(h), by only relying on the local context, the method is venerable to the sampling process and may not be able to recover from a badly sampled patch. On the contrary, our method captures the statistics of a texture pattern by a texture code and a parametric sine wave, thus it is more robust and stable for the texture synthesis task. We use a Python implementation[6] of the Image Quilting method to get all texture synthesis results.

WCT and DeepTexture The WCT [10] and DeepTexture [8] method maps a random noise image to a texture image either by a closed form feature statistic matching or iterative optimization. The WCT method produces inferior texture synthesis results (see Fig. 4(f)) because the covariance matrix used in the whitening and coloring process cannot fully capture the texture pattern. The DeepTexture method simply repeats the given texture patch randomly, as shown in column five and six in Fig. 4(g). Both methods fail to capture regular texture patterns (e.g., column three in Fig. 4) since the input noise image inherently lacks structure information. We use PyTorch implementations of the WCT[7] and the DeepTexture[8] to get the baseline results.

SinGAN The SinGAN [12] is a generative model trained on a single image. Starting from a small random noise image, the model progressively learns to synthesize an image that resembles the input. It essentially performs texture synthesis when the input image is a texture patch. To compare with SinGAN, we use the texture patches in Fig. 4(e) as training image and train a separate model for each texture patch using the official code[9]. As shown by the third image in Fig.2 4(e), SinGAN fails to model regular texture well since it takes the structure-less noise image as input. Furthermore, since it only works with rectangular texture patches, thus it cannot ignore irrelevant textures in the texture patch during synthesis. Quantitatively, it achieves a c-FID, c-FID (mask), LPIPS, LPIPS (mask) score of 145.65, 114.84, 0.2981, 0.1776, which is worse than the proposed method. Please see Table 1 in the manuscript for details.

[5] https://github.com/zalandoresearch/famos
[6] https://github.com/rohitrango/Image-Quilting-for-Texture-Synthesis
[7] https://github.com/sunshineatnoon/PytorchWCT
[8] https://github.com/honzukka/texture-synthesis-pytorch
[9] https://github.com/tamarott/SinGAN

Non-stationary texture synthesis The non-stationary texture synthesis (NonStat) method [14] takes a texture patch as input and expands it twice using transposed convolutions. To obtain its best performance, we train a separate model for each texture patch in Fig. 4(e) using its official code[10]. We note that the model can only expand the input texture patch twice each network forward, thus we recurrently feed the expanded image back to the model to obtain larger texture images. Specifically, we start from a 64×64 texture patch cropped from Fig. 4(e) and recurrently feed it into the model three times to obtain a 512×512 synthesis result. However, as shown in Fig. 4(k), the model fails to synthesize reasonable images. One possible reason is that the model requires high-resolution texture images for training while the texture patches in natural images are naturally small. The training patch we provide to NonStat have a size of 100×100. Providing larger training patches is possible but may introduce more irrelevant textures.

5 Visualization

5.1 GCN Edge Weight Visualization

To demonstrate that similar nearby nodes (i.e., superpixels) share similar features in the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (See Fig.2 of the manuscript), we visualize edges in the graph neural network in Fig. 5. For each edge connecting two adjacent superpixels, we compute its weight as the Cosine similarity of the adjacent superpixel features. In Fig. 5 (b) and (f), we draw edges with weights less than 0.2, i.e., edges that connect dissimilar adjacent superpixels. As we expected, these edges mainly exist on the boundaries of different texture segments. In Fig. 5 (c) and (g), we show edges with weights greater than 0.8, i.e., edges that connect similar adjacent superpixels. These edges are mostly within a texture segment. Furthermore, if we contract edges with weights greater than 0.85, we can already get a decent grouping of the image as shown in Fig. 5 (d) and (h). These demonstrations show that our graph neural network effectively learns similar features for similar adjacent superpixels and vice versa. Thus, it provides good features for the following convolution layer to predict the grouping of the input image.

6 More Qualitative Results

6.1 Arbitrary-sized Texture Synthesis

Our model captures a texture pattern by a compact vector texture code and a parametric sine wave. Thanks to the inherent periodicity of the sine wave, we can synthesize an arbitrary large texture image by simply extending the sine wave. In Fig. 6, Fig. 7 and Fig. 8, the sizes of the synthesized texture images has are 1000×1000.

6.2 More Texture Editing Results

We show more texture editing results in Fig. 9. For each pair of mask and image, we fill each region in the mask with textures from the reference image defined by a user. More details can be found in Section 4.2 of the manuscript.

[10] https://github.com/jessemelpolio/non-stationary_texture_syn

7 Limitations

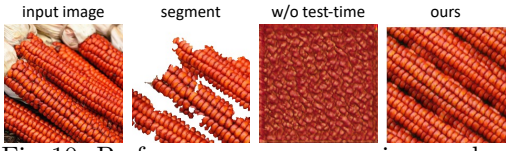

Fig. 10: Performance on unseen images before and after test-time adaptation.

In this work, we target at a challenging task of scraping texture from cluttered natural images without any supervision. Though our method performs comparable if not better compared to state-of-the-art methods, it has limitations. As shown in Fig. 10, our method requires test-time tuning on unseen images to synthesize high quality texture images. This is because we model the texture pattern compactly via a vector texture code and a sine wave, but mapping the distribution of the texture code and sine wave to a diverse texture distribution is non-trivial. Similar issues have been observed in employing generative neural networks to synthesize category agnostic images [3,5]. We leave this limitation into future works.

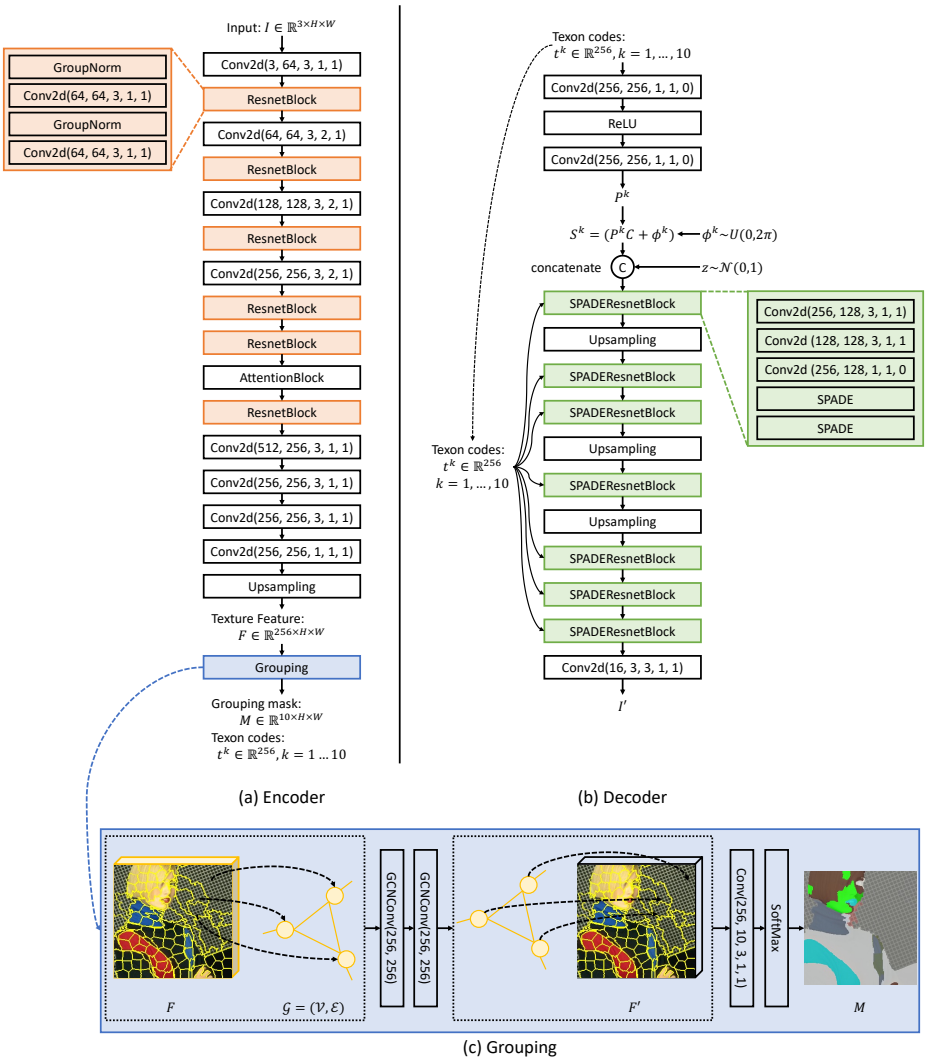

Fig. 1: **Detailed network architecture.**

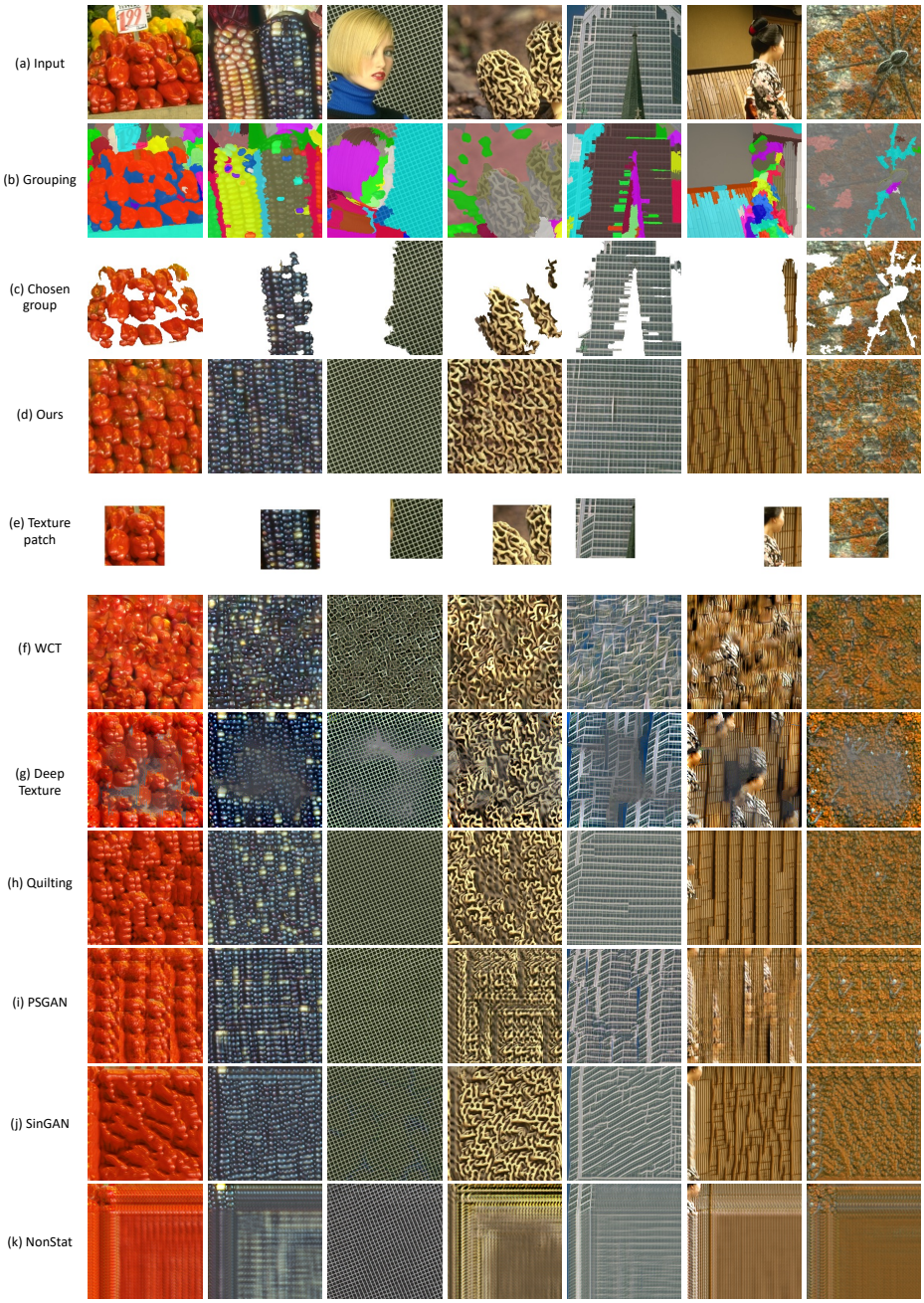

Fig. 4: **Texture synthesis results and comparison with baseline models.** (a) Input Images. (b) Grouping mask produced by our method. (c) Chosen texture segment for texture synthesis. (d) Texture synthesis by our method using the texture segment in (c). (e) Texture patch cropped from the center of the segment in (c). (f) $\sim$ (k) Textuer synthesis results by WCT [10], DeepTexture [8], Image Quilting [6], PSGAN [2], SinGAN [12] and NonStat [14] using the texture patch in (e).

Fig. 5: **GCN edge weights visualization.** (a)(e): Image. (b)(f): Edges with weights less than 0.2. (c)(g): Edges with weights greater than 0.8. (d)(h): Grouping obtained by contracting edges with weights greater than 0.85. Light grey lines in (b)(c)(f)(g) outline the SLIC superpixels [1].

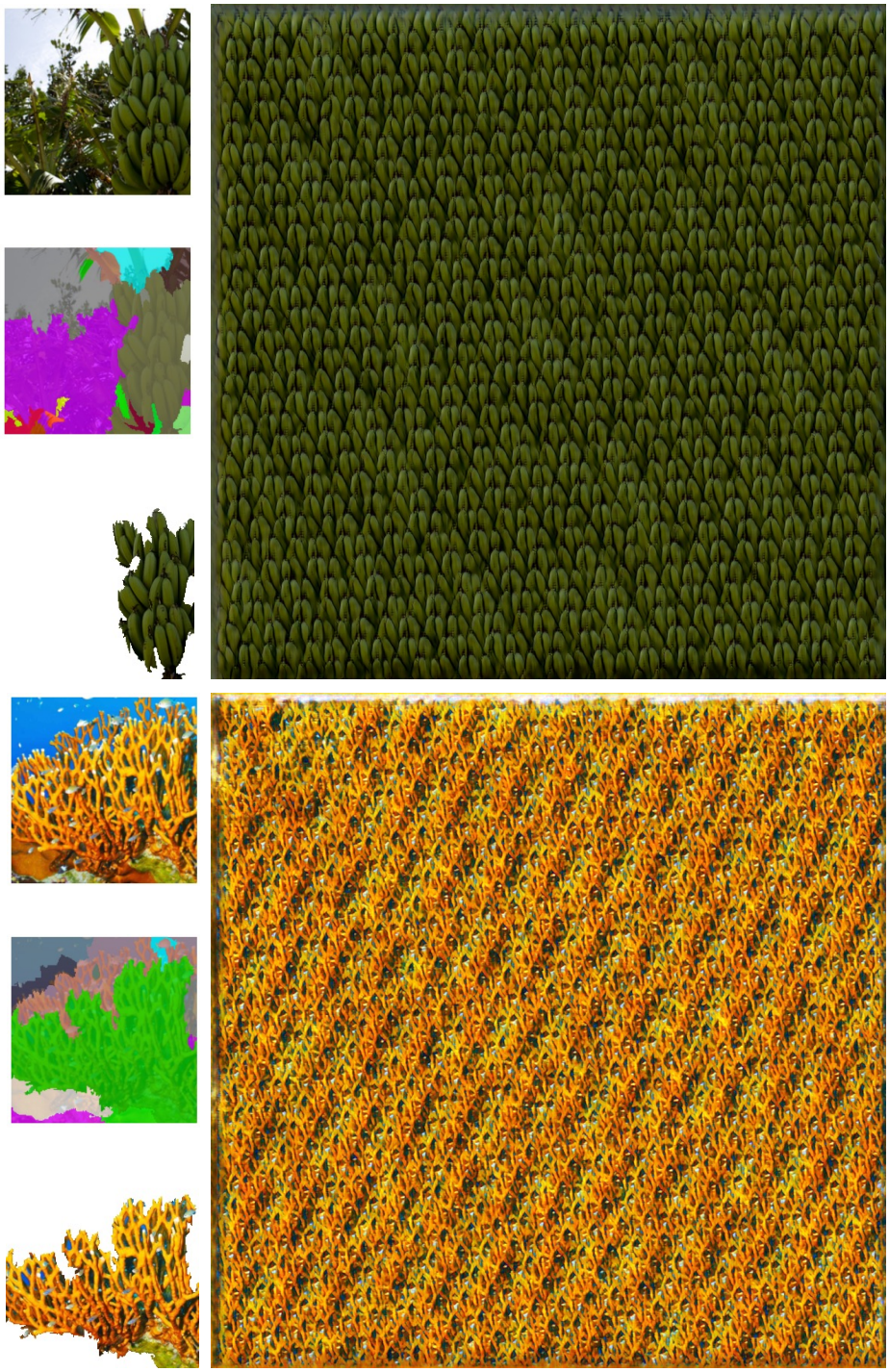

Fig. 6: **Arbitrarily large texture image synthesis.** We synthesize a 1000×1000 texture image resembling the chosen texture segment.

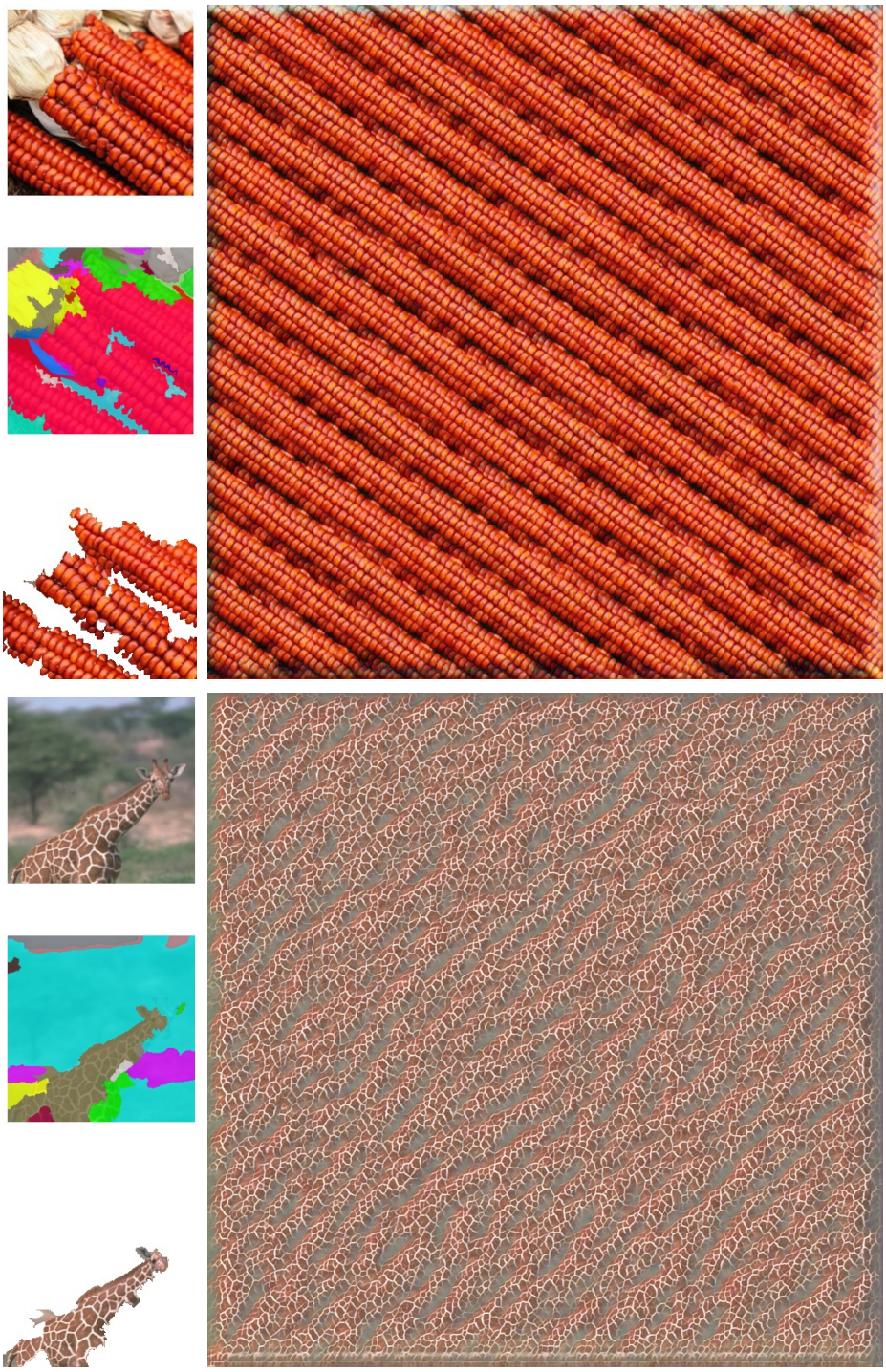

Fig. 7: **Arbitrarily large texture image synthesis.** We synthesize a 1000×1000 texture image resembling the chosen texture segment.

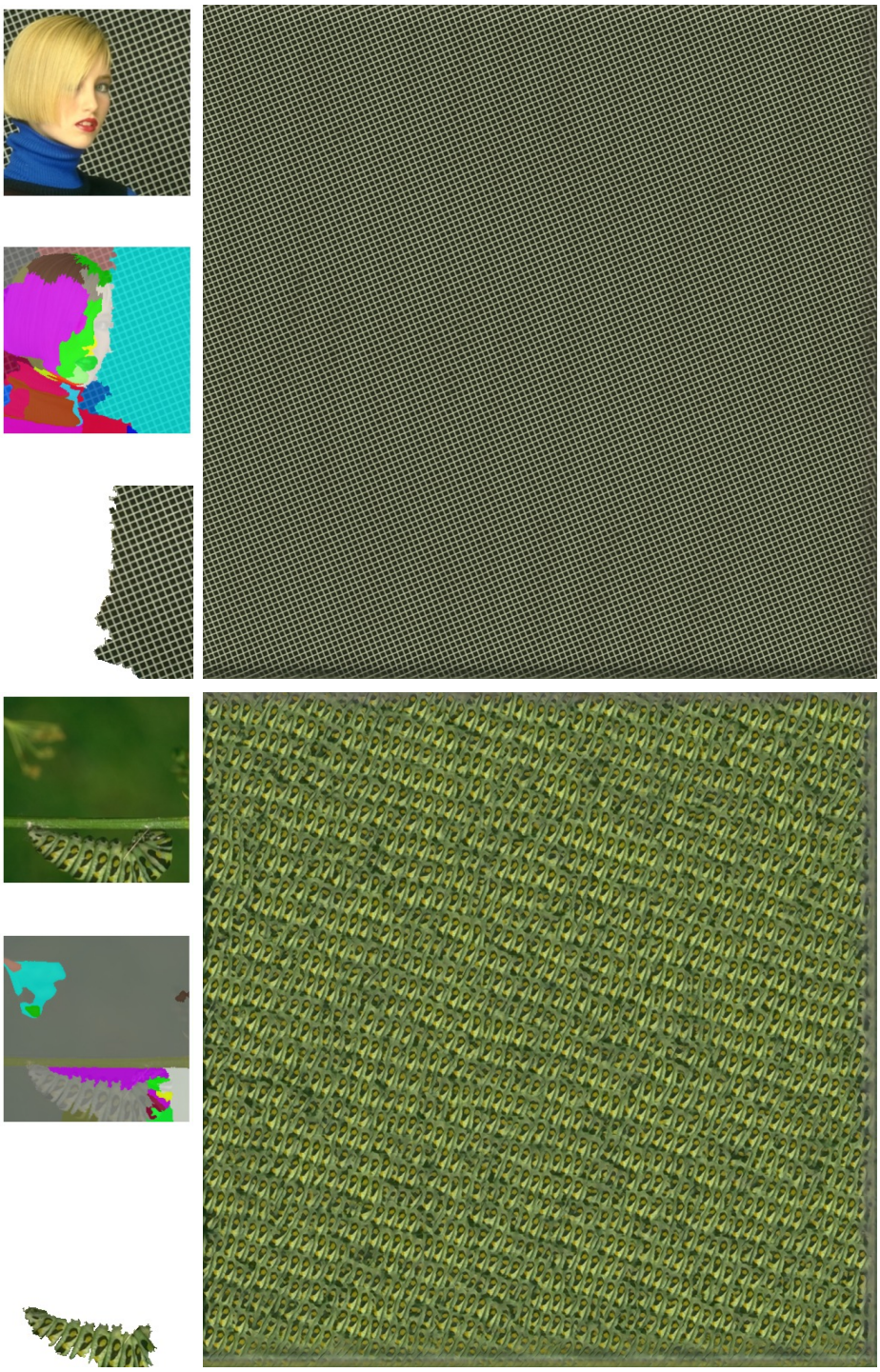

Fig. 8: **Arbitrarily large texture image synthesis.** We synthesize a 1000×1000 texture image resembling the chosen texture segment.

Fig. 9: **Texture editing.**

References

1. Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., Süsstrunk, S.: Slic superpixels compared to state-of-the-art superpixel methods. TPAMI (2012) 2, 10
2. Bergmann, U., Jetchev, N., Vollgraf, R.: Learning texture manifolds with the periodic spatial gan. arXiv preprint arXiv:1705.06566 (2017) 4, 9
3. Brock, A., Donahue, J., Simonyan, K.: Large scale gan training for high fidelity natural image synthesis. arXiv preprint arXiv:1809.11096 (2018) 7
4. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR (2009) 2
5. Dhariwal, P., Nichol, A.: Diffusion models beat gans on image synthesis. NeurIPS (2021) 7
6. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. SIGGRAPH (2001) 4, 5, 9
7. Esser, P., Rombach, R., Ommer, B.: Taming transformers for high-resolution image synthesis. In: CVPR (2021) 1
8. Gatys, L., Ecker, A.S., Bethge, M.: Texture synthesis using convolutional neural networks. NeurIPS (2015) 4, 5, 9
9. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. NeurIPS (2012) 2
10. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., Yang, M.H.: Universal style transfer via feature transforms. In: NeurIPS (2017) 4, 5, 9
11. Park, T., Liu, M.Y., Wang, T.C., Zhu, J.Y.: Semantic image synthesis with spatially-adaptive normalization. In: CVPR (2019) 1
12. Shaham, T.R., Dekel, T., Michaeli, T.: Singan: Learning a generative model from a single natural image. In: ICCV (2019) 4, 5, 9
13. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: CVPR (2015) 2
14. Zhou, Y., Zhu, Z., Bai, X., Lischinski, D., Cohen-Or, D., Huang, H.: Non-stationary texture synthesis by adversarial expansion. arXiv preprint arXiv:1805.04487 (2018) 4, 6, 9