Arbitrary-Scale Texture Generation from Coarse-Grained Control

Yanhai Gan, Feng Gao, Junyu Dong, and Sheng Chen

***Abstract*—Existing deep-network based texture synthesis approaches all focus on fine-grained control of texture generation by synthesizing images from exemplars. Since the networks employed by most of these methods are always tied to individual exemplar textures, a large number of individual networks have to be trained when modeling various textures. In this paper, we propose to generate textures directly from coarse-grained control or high-level guidance, such as texture categories, perceptual attributes and semantic descriptions. We fulfill the task by parsing the generation process of a texture into the three-level Bayesian hierarchical model. A coarse-grained signal first determines a distribution over Markov random fields. Then a Markov random field is used to model the distribution of the final output textures. Finally, an output texture is generated from the sampled Markov random field distribution. At the bottom level of the Bayesian hierarchy, the isotropic and ergodic characteristics of the textures favor a construction that consists of a fully convolutional network. The proposed method integrates texture creation and texture synthesis into one pipeline for real-time texture generation, and enables users to readily obtain diverse textures with arbitrary scales from high-level guidance only. Extensive experiments demonstrate that the proposed method is capable of generating plausible textures that are faithful to user-defined control, and achieving impressive texture metamorphosis by interpolation in the learned texture manifold.**

***Index Terms*—Texture synthesis, Bayesian hierarchy, Markov random field, fully convolutional network.**

I. Introduction

Natural images are generally difficult to be characterized by either deterministic or statistical models [1]–[4]. Textures, as a special type of images, are however most amenable to statistical modeling [5]–[7]. The notion of texture has a long history in visual perception, computer vision, computer graphics, computational geometry, etc. These applications all have slightly different characterizations of texture [8]. In this paper, we leverage the viewpoint of texture as an image containing repeating visual patterns with some amount of randomness [5]. More formally, a texture is a realization of a stationary, ergodic and Markovian stochastic process [8].

This work was supported by the National Key Research and Development Program of China under Grant 2018AAA0100602. (Corresponding author: Junyu Dong.)

Our implementation codes (including the proposed method and the combination method) as well as all the datasets are publicly available in https://github.com/gyh5421?tab=repositories.

Yanhai Gan, Feng Gao, and Junyu Dong are with the School of Computer Science and Technology, Ocean University of China, Qingdao, Shandong Province, 266100 China (e-mails: ganyanhai@ouc.edu.cn, gaofeng@ouc.edu.cn, dongjunyu@ouc.edu.cn).

Sheng Chen is with School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK (e-mail: sqc@ecs.soton.ac.uk).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org., provided by the author. The material includes one supplemental docement and two GIFs. Contact dongjunyu@ouc.edu.cn for further questions about this work.

Even artists for the virtual world design find it challenging to efficiently design realistic and complex textures with a high resolution [9]. Although a plethora of textures in the real world can be re-created in games or virtual worlds by applying existing texture synthesis techniques [10]–[12], the output is limited to be similar to the input sample. More specifically, for the tasks, such as texture synthesis [13], style transfer [14], colorization [15], dehaze [16], super-resolution [17], sketch/paint/segmentation to image [15], [18] and others [19]–[21], the distributions of particular characteristics of the target images need to be described in detail in the form of two-dimensional structured data, which provides 'fine-grained controls' for the tasks. By contrast, we believe that texture categories, perceptual attributes [22], semantic descriptions and other high-level cognitive conditions are 'coarse-grained controls'. In practice, texture generation under coarse-grained control is in high demand and poses a greater challenge [22].

An intuitive idea to accomplish generating textures from coarse-grained control is to first create a high-quality texture in some manners, and then apply a texture synthesis algorithm to produce textures with larger scale and similar appearance. For the first stage, direct image generation without using the traditional rendering pipeline [23]–[27] has attracted significant interest due to the promising results of deep networks [28], [29]. For the second stage, non-parametric methods often excel at generating realistic results but with limited generalization ability [6], [26], [30], whereas parametric approaches, especially deep learning based approaches, have produced promising results in recent years [28], [29]. Nevertheless, this intuitive two-stage procedure can neither provide users with flexible control on the appearance of the synthesized textures, nor is sufficiently efficient for real-time applications.

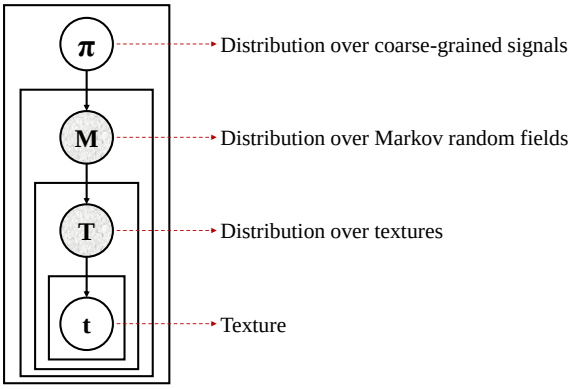


Fig. 1. The proposed hierarchical generating process. $\boldsymbol{\pi}$ represents the real distribution of the coarse-grained signals. **M** represents a distribution over the Markov random fields, which is determined by the sampled coarse-grained signal. **T** is the distribution of the texture images, which is a Markov random field sampled from **M**. t represents the texture sampled from **T**.

In this paper, we propose a framework that integrates exemplar texture creation and larger-scale texture synthesis into one pipeline, enabling coarse-grained control and real-time texture generation. This is achieved by parsing the generation of a texture into the three-level Bayesian hierarchy illustrated in Fig. 1. Specifically, a coarse-grained signal first determines a distribution over Markov random fields. Then a Markov random field is used to model the distribution of the final output textures. Finally the output texture is generated from the sampled Markov random field distribution. To the best of our knowledge, this is the first completely data-driven and controllable texture generation method. Before elaborating our contributions further, we review the related works.

A. Related Works

As one of the most powerful texture models, Markov random fields characterize texture by statistical interactions within local neighborhoods [10], [31]–[39]. However, the linear feature banks and small Markov blankets adopted by typical Markov texture models are usually insufficient for capturing complex structures [10], [40]. Thus, textures should be treated as samples of high-order Markov random fields, and complicated non-linear filters are needed to model the interplay between local regions. This however will impose high complexity. Despite of this drawback, realizations of the estimated model suggest that utilizing high-order interaction is realistic.

A Markov random field was traditionally simulated by a number of approximation techniques, with Markov chain Monte Carlo algorithm being the most popular one. With the recent reviving in deep learning [41]–[47], researchers can no longer limit to Markov random fields and take advantages of deep learning's powerful fitting ability to solve general random fields [28], [29], [48]–[52]. Various methods have been developed, which can be categorized into two main types: 1) likelihood based models, including autoregressive models [48], [53], variational autoencoders (VAEs) [28], [49], [54] and flow-based generative models [55], [56]; and 2) implicit generative models, such as generative adversarial networks (GANs) [29], [57] and moment matching based generative models [58]–[63]. These methods are generally concerned with the extremely challenging tasks of modeling all the dependencies within very high-dimensional input data, usually specified in the form of full joint probability distribution.

This formulation, however, makes it difficult to use these methods when dealing with large-scale images due to the curse of dimensionality [64], [65]. On the other hand, for many practical applications, e.g., computer graphics, virtual reality, animation, etc., there exists particular need to create large-scale textures with photo-realistic imagery. Therefore, the prevailing generative methods can hardly serve as off-the-shelf models for texture generation, although they may perform well on some natural image generation tasks [49], [52], [66], [67]. In fact, capturing all the patterns within the sample data for modeling textures is unnecessary due to the actuality that spatial ergodicity of textures is essentially satisfied as the array expands. Specifically, an ergodic texture has the same behavior averaged over locations as averaged over all the states in its phase space. Since the state of an ergodic process after a long time is nearly independent of its initial state, it is unnecessary to employ enormous models which would require huge amount of computation resources. In particular, after we capture the parameters of the Markov random field, we can arbitrarily enlarge the content of the texture by consciously controlling the boundary conditions.

Indeed, a number of efforts have been devoted into this regard recently [7], [10]–[12], [18], [68], [69], and large body of works leverage GANs [12], [18], [68]–[70] for learning texture models (some works [18], [69] also deal with natural images). However, the networks employed by these methods [11], [12], [18], [68]–[70] are always tied to individual exemplar textures, meaning that a large number of individual networks have to be trained when various textures need to be modeled. In addition, other neural texture synthesizers [7], [10], [71] either produce compromised visual quality [71] or need a long runtime for deconvolution [7], [10]. Furthermore, we may also emphasize that the current approaches all focus on fine-grained control of texture generation, i.e., synthesizing images from exemplars [7], [11], [18], [68], [71], [72], whereas texture generation based on coarse-grained control, e.g., categorical, perception [22] and semantic [73] attributes, are rarely investigated. Another limitation of the current methods is that the trained networks lack the ability to realize textures metamorphosis, i.e., they lack the ability to smoothly transform a source texture into another one so that new texture can be created.

B. Our Novel Contributions

It can be seen that most existing approaches compose texture synthesizer of specific style, where separate convolutional networks have to be trained when modeling different textures. Furthermore, the existing texture models focus on fine-grained control of texture generation. Against this background, this paper exploits GANs [22], [57], [74]–[79] in a new realm – generating diverse textures from coarse-grained signals. Differing from the previous works [7], [11], [12], [18], [68]–[72], the proposed texture model is trained on multifaceted texture datasets and therefore our model inherently supports realization of dramatic visual variations owing to coarse-grained guidance. The Markov random field employed at the bottom level of the Bayesian hierarchy depicted in Fig. 1 makes it possible to generate textures of arbitrary scale and width/height aspect ratio. Therefore, the proposed method can be seen as a unified pipeline that integrates texture creation (from coarse-grained signals) and texture synthesis.

This is the first fully tractable yet flexible texture model that provides an efficient way for generating textures directly from coarse-grained control. Our proposed method has the following characteristics which make it state-of-the-art for texture generation: 1) high quality of the generated textures, 2) high scalability with respect to the output texture size, 3) real-time capability for large-scale texture generation, 4) large variety of the generated appearances under the same coarse-grained control, and 5) hallucinogenic texture metamorphosis.

II. Proposed Method

A. Preliminary and Notations

Consider an undirected graph $G=(V,E)$ with the sets of vertices V and edges E, respectively, as well as a set of random variables $X=(X_v)_{v\in V}$ indexed by V. We denote each output of the random vector X as $x=(x_1,x_2,\ldots,x_{card(V)})$, where x_v is associated with vertex v and $card(V)$ represents the cardinality of V. Let $P(x_v|x_{-v})$ denote the conditional distribution of x_v, given all other values $x_{-v}=\{x_u : u\neq v\}$. Then, the random vector X forms a Markov random field with respect to G if the following local Markov property is satisfied:

$$P(x_v|x_{-v}) = P(x_v|x_{\partial v}), \tag{1}$$

where ∂v represents the neighbours of vertex v. The Markov property of an arbitrary probability distribution can be difficult to establish, and a commonly used class of Markov random fields are those that can be factorized according to the cliques of the graph. Specifically, if the joint density of the random variables can be factorized over the cliques of G:

$$P(X=x) = \prod_{C\in cl(G)} \phi_C(x_C), \tag{2}$$

then the random vector X forms a Markov random field with respect to G. Here, $cl(G)$ is the set of cliques of G, and x_C is a particular field configuration of the clique C. The functions ϕ_C are sometimes referred to as factor potentials or clique potentials. Although the factorization is favorable for computing purpose, not all Markov random fields can be factorized. A Markov random field can be factorized if at least one of the following two conditions is satisfied:

- The density is positive (by the Hammersley-Clifford theorem).
- The graph is chordal[1] (by equivalence to a Bayesian network).

Essentially, natural images lie in a low dimensional manifold [80], and the model distribution is also supported by a low dimensional manifold as we use deep learning methods to map low dimensional vectors to the training data space [81]. Therefore, the modeled joint density is no way to be positive. On the other hand, since we adopt fully convolutional networks (FCNs) [82] for simulating Markov random fields, the implied graph must not be chordal, details of which will be discussed in Subsection II-D. To address these two problems, we adopt two strategies: (i) we audaciously postulate that the employed deep model can assign infinitely small probabilities to samples that are unlikely to appear in the real distribution, so that the first condition is approximately met; and (ii) we add instance noise into the generated and real data samples to create a proxy distribution that is nearly positive, and the proxy distribution is optimized during training. As a result, we can think that the density of the modeled Markov random field is factorable. How to create the proxy distribution can be found in Subsection II-D, and the effects of these two solutions will be analyzed in the ablation study of Section III.

[1]A chordal graph is one in which all cycles of four or more vertices have a chord.

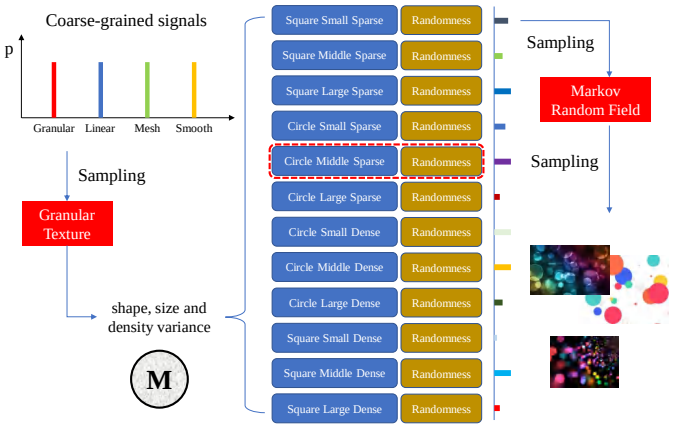


Fig. 2. Flowchart of the generating process. To obtain one texture, we do sampling three times. First, as the coarse-grained signals are fetched from training data, we are indeed sampling from the real distribution. A coarse-grained signal (e.g., 'granular texture') depicts only certain aspects of the appearance of the textures, and it determines a distribution over the Markov random fields (denoted by blue rectangles in the figure) which share some common cues about the appearance. Second, a Markov random field is sampled from the distribution. The sampled Markov random field determines how the appearance of the textures behaves as it associates distinct probabilities with different visual patterns. Third, a texture is sampled from the Markov random field, where the randomness will create abundant location variances and appearance distortions around the textons. These randomness make the generated textures look more realistic. Furthermore, arbitrary-scale texture generation can be achieved as the Markov random field formulates a full condition to guide the growth of the texture according to existing contents.

B. Resolve Texture Generation Process

Similar to the previous work [12], [68], [70], we treat textures as outputs of Markov random fields. However, we further view the Markov random fields as samples of another distribution, which is itself a sample of a stochastic process. As illustrated in Fig. 1, $\boldsymbol{\pi}$ represents the prior distribution of the coarse-grained signals and t represents one observable texture, while $\mathbf{M}$ and $\mathbf{T}$ are implicit probability models. In order to realize coarse-grained control on the appearance of the synthesized textures, our framework eventually optimizes the joint probability of the coarse-grained signals and textures, and two latent variables will be introduced to simulate samplings from $\mathbf{M}$ and $\mathbf{T}$. This will be further explained in Subsection II-C.

Fig. 2 exemplifies the generating process with four curated types of textures (coarse-grained controls). First, one coarse-grained signal is sampled from the prior distribution, and we denote it as c, i.e., $c\sim\boldsymbol{\pi}$. Each coarse-grained signal (e.g., 'granular texture') determines a distribution over a random variable m, and we denote this distribution as $\mathbf{M}$. The coarse-grained signal, e.g., 'granular textures', indeed corresponds to large variants, each of which represents a family of textures that share similar appearance and are outputs of the same Markov random field. In other words, m implicitly determines a Markov random field, and the prior distribution $\boldsymbol{\pi}$ conforms to a stochastic process whose sample orbital is the probability function over the Markov random fields. Second, a particular m is sampled from $\mathbf{M}$, and emits a Markov random field $\mathbf{T}$. Afterward, various textures can be sampled from $\mathbf{T}$. In Fig. 2, $\mathbf{T}$ represents textures composed of sparsely distributed middle circles, and the sampled textures from $\mathbf{T}$ are preattentively indistinguishable to a human observer [83], yet containing

different texton configurations and distortions.

The above analysis declares that a sample of $\boldsymbol{\pi}$ is a seed to $\mathbf{M}$, determining a distribution over Markov random fields, and a sample of $\mathbf{M}$ is a seed to $\mathbf{T}$, determining a distribution over textures. The seeds here are in a way reminiscent of the base distribution of the Dirichlet Process [84]. Therefore, the complete workflow depicted in Fig. 2 should be:

$$\boldsymbol{\pi} \rightarrow c \mapsto \mathbf{M} \rightarrow m \mapsto \mathbf{T} \rightarrow t, \tag{3}$$

where $\rightarrow$ represents sampling from the distribution and $\mapsto$ represents derivation of a distribution from the given seed. This workflow formulates a Bayesian hierarchy:

$$\begin{aligned} \mathbf{M}|\boldsymbol{\pi} &\sim \boldsymbol{\pi}(c), && (4)\\ \mathbf{T}|\mathbf{M} &\sim P(m|c,\theta_M), && (5)\\ t|\mathbf{T} &\sim P(t|m,\theta_T), && (6) \end{aligned}$$

where θ_M represents the parameters of the probabilistic model that casts the coarse-grained signals to distributions over Markov random fields and θ_T represents the parameters that formulate the Markov random field according to m.

By combining (4) to (6), we obtain the joint distribution:

$$\begin{aligned} P(t,\mathbf{T},\mathbf{M}) =& P(\mathbf{M})P(\mathbf{T}|\mathbf{M})P(t|\mathbf{T},\mathbf{M}) && (7)\\ =& P(\mathbf{M})P(\mathbf{T}|\mathbf{M})P(t|\mathbf{T}) = \boldsymbol{\pi}(c)\mathbf{M}(\mathbf{T})\mathbf{T}(t) && (8)\\ =& \boldsymbol{\pi}(c)P(m|c,\theta_M)P(t|m,\theta_T) && (9)\\ =& \boldsymbol{\pi}(c)P(m|c,\theta)P(t|m,\theta), && (10) \end{aligned}$$

where $\theta = \{\theta_M, \theta_T\}$ denote all the parameters of the model. From (7) to (10), we utilize an assumption that the parameters of the Markov random field are sufficient to model the distribution over textures, and the other information can be neglected. As $m \mapsto \mathbf{T}$ and $c \mapsto \mathbf{M}$ persist, $P(t,\mathbf{T},\mathbf{M}) = P(t,m,c)$ holds all the time. To obtain the joint probability of coarse-grained signals and textures, we integrate out the other variable:

$$P(t,c) = \int_m P(t,m,c)dm = \int_m \boldsymbol{\pi}(c)P(m|c,\theta)P(t|m,\theta)dm. \tag{11}$$

To obtain a good approximation of this joint probability, a large batch size should be applied during training.

C. Hierarchy Model Construction

Eq. (11) indicates that three samplings occur in the simulation process: firstly a coarse-grained signal is sampled from the prior distribution $\boldsymbol{\pi}$; secondly a Markov random field is sampled from $p(m|c,\theta)$; and thirdly a texture is sampled from $p(t|m,\theta)$. Because coarse-grained signals are fetched from the training data, they are naturally consistent with the prior distribution $\boldsymbol{\pi}$. As mentioned in Subsection II-B, we need to use two implicit probability models to estimate $\mathbf{M}$ and $\mathbf{T}$. For this purpose, we define two transformations:

$$\begin{aligned} m &= f_{\theta_M}(c, z_1), && (12)\\ t &= g_{\theta_T}(m, z_2), && (13) \end{aligned}$$

where z_1 and z_2 are two factorable latent variables whose values are independently and identically sampled from a Gaussian distribution $N(0,1)$. Clearly, f_{θ_M}, along with specific c, implies a distribution over m, and similarly, g_{θ_T}, along with specific m, implies a distribution over t, which is assumed to be a Markov random field. Afterwards, sampling from $\mathbf{M}$ and $\mathbf{T}$ can be converted to sampling from $p(z_1)$ and $p(z_2)$ and then applying the corresponding transformation functions.

By substituting m in (13) with $f_{\theta_M}(c, z_1)$, we obtain:

$$t = g_{\theta_T}(f_{\theta_M}(c, z_1), z_2) = h_\theta(c, z_1, z_2), \tag{14}$$

where h_θ is the composite function formed by f_{θ_M} and g_{θ_T}, with $\theta = \{\theta_M, \theta_T\}$ as the parameters. The composite function makes m implicit and its working mechanism enrolled in the model parameters, which avoids striving for the dedicated formalism of m. Because g_{θ_T} induces a Markov random field over t when m is fixed, h_θ conformably forms a Markov random field over t when c and z_1 are fixed. In fact, we can view c and m as the parameters of the two transformations from z_1 and z_2, respectively, namely, $m = f_{\theta_{M,c}}(z_1)$ and $t = g_{\theta_{T,m}}(z_2)$, and a specific combination of c and z_1 corresponds to a particular parameter configuration of $g_{\theta_{T,m}}$.

To simulate the Markov random field realized by h_θ when c and z_1 are fixed, we apply a FCN on z_2 and adopt c and z_1 to adjust the computation of the network. The reason why such a FCN substantiates a Markov random field can be found in [68]. Basically, the limited receptive field of a FCN makes remote portions of the same output independent of each other, which implies a stationary and ergodic stochastic process. In

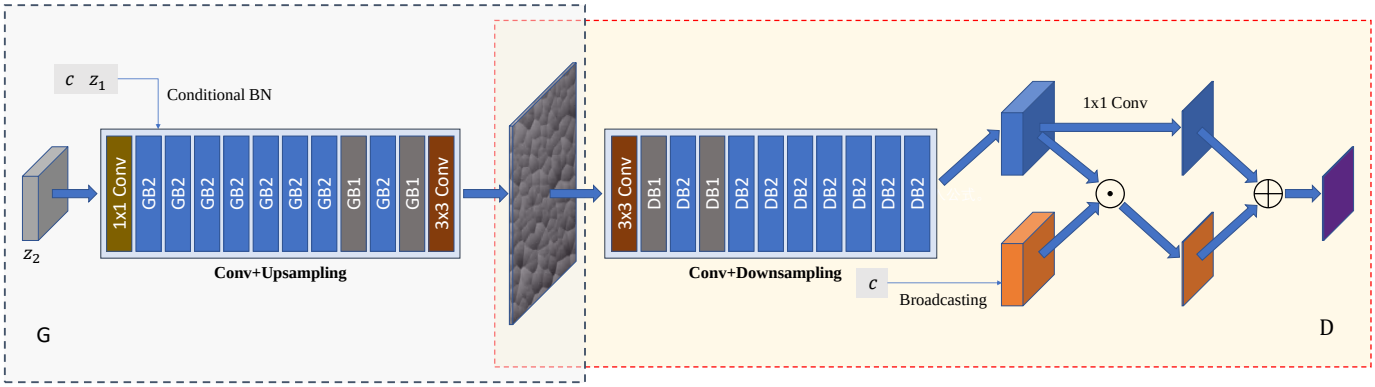


Fig. 3. Architecture of the implementation. **Conv+Upsampling** represents a FCN consisting of cascaded convolutional layers, non-linear activation functions, conditional batch normalization (CBN) layers [14], and bilinear interpolation layers. **Conv+Downsampling** represents a FCN consisting of cascaded convolutional layers, non-linear activation functions, and average pooling layers. $\odot$ and $\oplus$ represent element-wise multiplication and addition.

accordance with the FCN structure, z_2 embodies a 3-D tensor that stands for the random variance of appearance among spatial locations, while c and z_1 turn into global variables that impose systematic effects on the transformation function from z_2 to t. In the sequel, we regard the composite function h_θ and the FCN employed to realize it as interchangeable.

D. Implementation

The overall architecture of our implementation is illustrated in Fig. 3. We substantiate an implicit generative model to learn our hierarchical texture model. Specifically, SN-GAN [85] is employed as the basic scheme for training. Compared to other GAN variants [66], [74], [80], [81], [86], [87], SN-GAN affords stable training with large learning rate and improves the quality of the generated images by using spectral normalization as a novel way to realize the Lipschitz constraint [66], [81]. Besides normalizing the spectral of each layer's weights of the discriminator D, we also apply spectral normalization in our generator G according to the recommendation of [52] to achieve better generalization by reducing the generator's sensitivity to the perturbation of inputs [88], [89].

Since the scale of the generated textures can be changed, the discriminator also implements a FCN and forms a two-dimensional tensor as the output with each element representing the possibility of being a patch from the real textures (patch discriminator [15]). To embed c and z_1 into h_θ, we supply c and z_1 with conditional gains and biases in batch normalization (conditional batch normalization [14], [90]) layers to exert a global influence on the transformation from z_2 to t. In addition, we introduce a newly designed squeeze-excitation noise layer in certain building blocks of the generator to compensate for local stochasticity of the generated textures. Full details of the network structure can be found in Subsection III-B.

As discussed in Subsection II-A, the density of the modeled Markov random field is factorable, either i) the deep model can assign infinitely small probabilities to samples which are forbidden by the real distribution or ii) we add instance noise to the generated and real-data samples to create a proxy distribution for optimization. In light of this, the possibility of the whole texture being realistic can be formulated as the product of probabilities of all maximal cliques of the texture. Even so, Eq. (2) is still difficult to realize in the deep learning regime, where summation is preferred to multiplication as summation scales more easily to high dimensional data with light computational footprint. Thus, we alternatively adopt an exponential family probability of the Markov random field:

$$\begin{aligned} P(X=x) &= \prod_{C\in cl(G)} \exp(\ln(\phi_C(x_C))) \\ &= \exp\left(\sum_{C\in cl(G)} f_C(x_C)\right), \end{aligned} \tag{15}$$

where $f_C(x_C)$ is the logarithm of the potential $\phi_C(x_C)$. Eq. (15) enables us to optimize $P(X)$ through adapting the sum of all the elements in the set $\{f_C(x_C)|C \in cl(G)\}$.

Multi-scale training [91] is a common technique to enable a trained deep model to naturally generalize to images of different sizes. We emphasize that single-scale training is sufficient to ensure our designed deep model to generalize to the textures of any scale in inference. In fact, we found out in the experiments that multi-scale training is not beneficial to the generalization of our model. This is because the non-padding convolutions are used in both our generator and discriminator networks, and hence the deep model has no means to know where the border is and inevitably treats patches as samples for optimization, ultimately formulating random fields strictly conformable to Markov properties. Therefore, multi-scale training has no effect on the generalization of the trained model. All the reported results in the experiment section are obtained through training our model on a single scale, which confirm that our model is capable of generalizing the textures to different scales. In addition, because we aim to fit the joint probability of the coarse-grained signals and textures, the coarse-grained signal c is taken into consideration together with t for the calculation of the discriminator's response.

The size of the maximal clique of the simulated Markov random field is just the size of the projective field of the generator. Pixels within a projective field interact with each other through the point they receive information, on the foundation that some proximal (in terms of *Chebyshev distance*) pixels interplay through more common clues. If we denote the pixels within a projective field as C and add a pixel outside the projective field, the added pixel must be independent of the most remote (in terms of *Chebyshev distance*) pixels in C. Therefore, the projective field constitutes the maximal clique of the Markov random field. Accordingly, we set the size of the receptive field of the discriminator the same as that of the projective field of the generator to estimate logarithm potentials of the maximal cliques. It should be noted that the Markov blanket of certain pixel is twice the size of the maximal clique minus one. This results in an easy inducement of a non-chordal cycle of four vertices in the implied graph, which has been mentioned in Subsection II-A. To enlarge the Markov blanket for capturing more complex textons, we can apply a bigger projective field within the available computation resources.

In Subsection II-A, we point out that a proxy distribution can be created to make the modeled Markov random fields factorable. Even though we process all data in float point, the raw images are in uint8 format and the generated images also have to be converted into uint8 format for displaying and persistence. That is, the distribution of the real data is discrete, and the distribution of the generated data also becomes discrete after quantization. What we really want is to align these two discrete distributions, even though, for convenience, we treat all data as continuous during training (plugging quantization into the network will interrupt the gradient flow, because it produces zero gradients almost everywhere).

Reconsidering the quantization process, we know that the resulted discrete distribution consists of the integrals of the density (or cumulative masses if the distribution is inherently discrete) over evenly divided intervals of the value space. Without loss of generality, if a uniform scalar quantizer (i.e., each element is rounded to the nearest integer: $\hat{x}_v =$

$round(x_v)$) is applied, the following formula will hold:

$$P(\hat{x}) = \lim_{\varepsilon \to 0^-} \int_{\hat{x}-0.5}^{\hat{x}+0.5+\varepsilon} P(x)dx, \ \forall \hat{x} \in \mathbb{Z}^{card(V)}. \quad (16)$$

$P(x)$ may be inherently discrete or a fusion of discrete and continuous distributions, and if so, we treat it as a train of Dirac delta functions with weights equal to the probability mass function, so that we can uniformly use integral operation for probability aggregation. The same trick is utilized in the sequel, whenever the same situation is encountered.

On the other hand, if we add the independent and identically distributed (i.i.d.) uniform noise $n \sim \mathcal{U}(\boldsymbol{I})$ with $\boldsymbol{I} = (-0.5,\ 0.5]^{card(V)}$ to the data, we can obtain

$$P(\tilde{x}) = P(x) * P(n) = \int P(x)\mathcal{U}(\tilde{x} - x)dx, \quad (17)$$

where $\tilde{x} = x + n$ is the corrupted data, and $*$ denotes convolution operation. For any integer value of $\tilde{x}$,

$$\begin{aligned} P(\tilde{x} = z) &= \int P(x)\mathcal{U}(z - x)dx \\ &= \lim_{\varepsilon \to 0^-} \int_{z-0.5}^{z+0.5+\varepsilon} P(x)dx, \ \forall z \in \mathbb{Z}^{card(V)}. \end{aligned} \quad (18)$$

By combining with (16), the above result shows that the probability density of $\tilde{x}$ at any integer value is equal to the probability mass of $\hat{x}$ at the same value. For the real data, the aggregation of the probabilities over the quantization bin $(-0.5,\ 0.5]^{card(V)}$ is just the probability mass of the real data at the bin center, as the distribution of the real data is discrete. Therefore, if we create the proxy distributions for the generated data and real data by contaminating them with the uniform noise n, we can essentially align the probability mass functions of the generated data (after quantization) and real data by alternatively aligning their proxy distributions.

Adding instance noise bears some resemblance to what used to stabilize the training of generative models [92], in particular, GANs [80], [93], [94], but it differs from the latter in the constraints imposed to ensure the alignment of the discrete probability mass functions all along the training dynamics. As a result, rather than reporting an estimate of the discrete distribution, the implementation creates an ensemble considering ultimate morphology of data and pronounces performance using the actual resulted distribution, thus demonstrating the feasibility of our solution as a complete storage data generation method. It should be noted that the added uniform noise thickens the manifold, which usually owns a zero volume [81], using a unit hypercube brush, pushing the distribution one step closer (but not really) positive. The efficacy of performing optimization in the proxy distribution space will be analyzed empirically through ablation study in the experiment section.

III. Experiments

A. Datasets

Publicly available texture datasets [95]–[99] are usually constructed many years ago and they are not suitable for our purpose, i.e., generating high-quality textures from coarse-grained control. For example, textures in CuRet [95] and Outex [96] are in low resolutions and of low quality, whereas many textures in VisTex [98] and DTD [99] are neither stationary nor ergodic. To illustrate the generation ability of our hierarchical texture model, we introduce two new high-quality, large-variety texture datasets (HRTD-G and HRTD). The both datasets consist of 1000 textures from 10 categories.

In the first dataset, procedural textures are generated in a way similar with [100]. However, we only choose 10 procedural texture models: 'CA(forest fire model)', 'CA(surface tension model)', 'CA(excitable media model)', 'Cellular', 'Folding texton', 'Folding cellular', 'Folding fractal', 'Folding perlin', 'Fractal(one-over-fBeta-noise)', and 'Fractal(Fourier spetral synthesis)', to produce 100 textures for each model. All the textures in this dataset have a resolution of 512×512 and an 8 bit grey scale. We refer to this dataset as high resolution texture dataset with grey scale (HRTD-G). In the following experiments, we crop the central 448×448 part from each texture, and then resize it to 224×224 as a training sample.

Textures in the second dataset are natural images collected from websites. They are from 10 categories: 'blanket', 'brick', 'fabric', 'marble', 'metal', 'water', 'sky', 'stone', 'tile' and 'wood', with 100 textures in each category. Textures in this dataset vary in size with the shortest side as 240, the longest side as 6038, the lowest aspect ratio (width/height) as 0.56, and the highest aspect ratio as 1.88. All the textures in this dataset are 24 bit RGB images. We call this dataset high resolution texture dataset (HRTD). In the experiments, we centrally crop along the longer side of each image to form a square image, and then resize it to 224×224 as a training sample.

In addition, we use the perceptual texture database (PTD) [101] for evaluating the ability of generating textures from perceptual descriptions. PTD consists of 450 textures with each texture owning 12 perceptual attributes: contrast, repetition, granularity, randomness, roughness, feature density, directionality, structural complexity, coarseness, regularity, local orientation, and uniformity [101], [102]. These perceptual attributes are all labeled based on psychophysical experiments. Textures in this dataset have a resolution of 512×512 with 256 grey scales. Each perceptual attribute has a value in the range of 1 to 9, with the high value and low value representing the opposite properties of a perceptual feature. In our experiments, we crop the central 448×448 patch from each texture, and then resize it to 224×224 as a training sample.

For all the datasets, we randomly crop a 146×146 patch from each of the training samples in every epoch to train our texture model. All the displayed textures in the following figures (except the two large textures in Fig. 10) are of 622×622 resolution but shown in smaller spaces due to the page limit. Readers can zoom in to see more details.

B. Networks

We adopt the network structure of [67] as the backbone to construct our generator and discriminator, and apply the same hyperparameters of [67]. However, we remove all the fully connected layers from the original structure, and extend the input of the generator from a vector to a 3-D tensor, and form the output of the discriminator as a 2-D tensor. As underlined

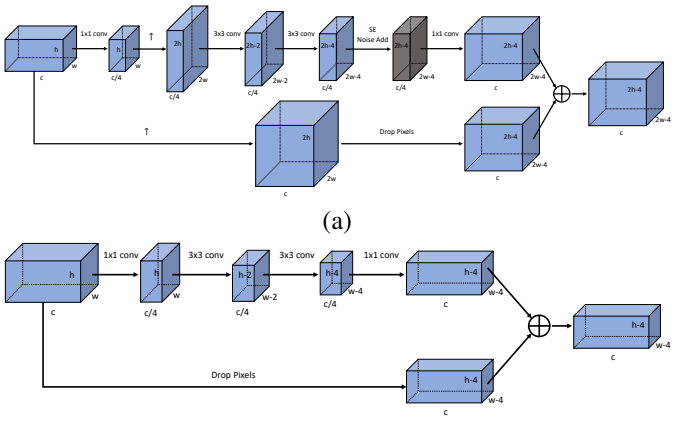


Fig. 4. Residual modules of the generator with (a) and (b) corresponding to GB1 and GB2 in Fig. 3, respectively. Therein, ↑ represents bilinear upsampling, and ⊕ represents element-wise addition.

in Subsection II-D, we alter the residual modules [41] to avoid any zero padding in the convolutional layers. Specifically, we synchronously drop pixels of the input as the convolutional layers proceed, while other operations remain the same as the original, in every residual module. Thus, the cropped input and the convolutional layers' output fit seamlessly in the spatial dimensions at the final stage of the residual operation. The overall architecture has been given in Fig. 3, where all the convolutional layers are with a stride of 1 and a kernel size of 3×3. The two building blocks of the generator are illustrated in Fig. 4 (a) and (b), while the two building blocks of the discriminator are illustrated in Fig. 5 (a) and (b).

Inspired by [103], [104], we provide the generator with a direct means to generate stochastic details by introducing explicit noise inputs to the intermediate layers. However, rather than feeding a dedicated noise image to each convolutional layer of the generator as in [103], [104], we compensate for randomness only when the resolution is doubled (as depicted in Fig. 4 (a)), because we believe that doubling the resolution creates urgent requirement for additional randomness to support plausible local variety. Moreover, the authors of [103], [104] utilize the learned per-feature scaling factors to broadcast

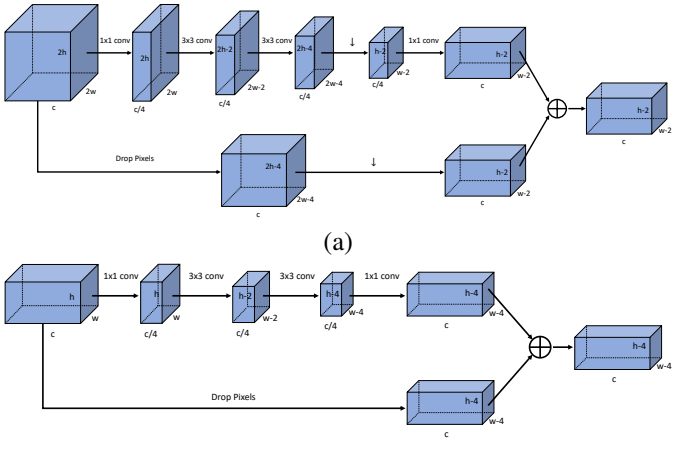


Fig. 5. Residual modules of the discriminator with (a) and (b) corresponding to DB1 and DB2 in Fig. 3, respectively. Therein, ↓ represents average pooling for downsampling, and ⊕ represents element-wise addition.

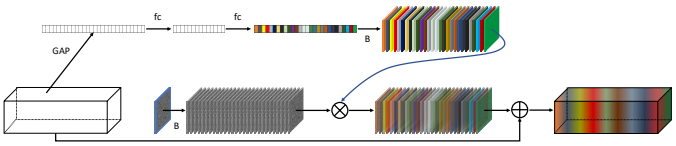


Fig. 6. Illustration of the SE noise layer used by the residual modules of the generator. Therein, GAP is global average pooling, fc represents fully connected layer, B represents broadcasting along the spatial axis. ⊗ and ⊕ represent element-wise multiplication and addition, respectively.

the single-channel noise image to all feature maps. However, these scaling factors become constant after the model is optimized. A more reasonable solution is to adaptively recalibrate the noise image for each instance, because a distinct instance predictably requires a distinguishing level of compensation for randomness. To this end, we borrow the spirit of the 'Squeeze-and-Excitation' (SE) block [46] to produce personalized per-feature scaling factors for each instance. We call this module 'SE Noise Layer', and depict it in Fig. 6.

In addition, before feeding the coarse-grained signals to the networks, if they are categorical, an embedding layer is applied to embed the discrete signals into continuous vectors (as in [67]). Otherwise when real-valued vectors are encountered, an affine transformation combined with a non-linear activation function is applied to form a mapping from the original signals to more disentangled latent space [103]. After that, the transformed coarse-grained signals are fed into the generator and discriminator, respectively. Note that we apply the same embedding or mapping function to the coarse-grained signals in all the layers of the generator, i.e., all the applications of the transformation share the same set of parameters. The embedding or mapping function in the discriminator however differs from that of the generator. As an example, Fig. 7 depicts this utilization along with CBN in the generator.

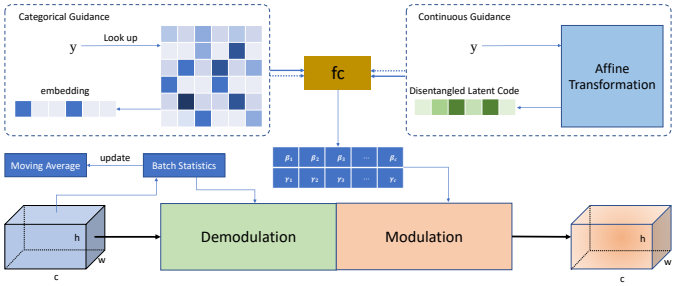
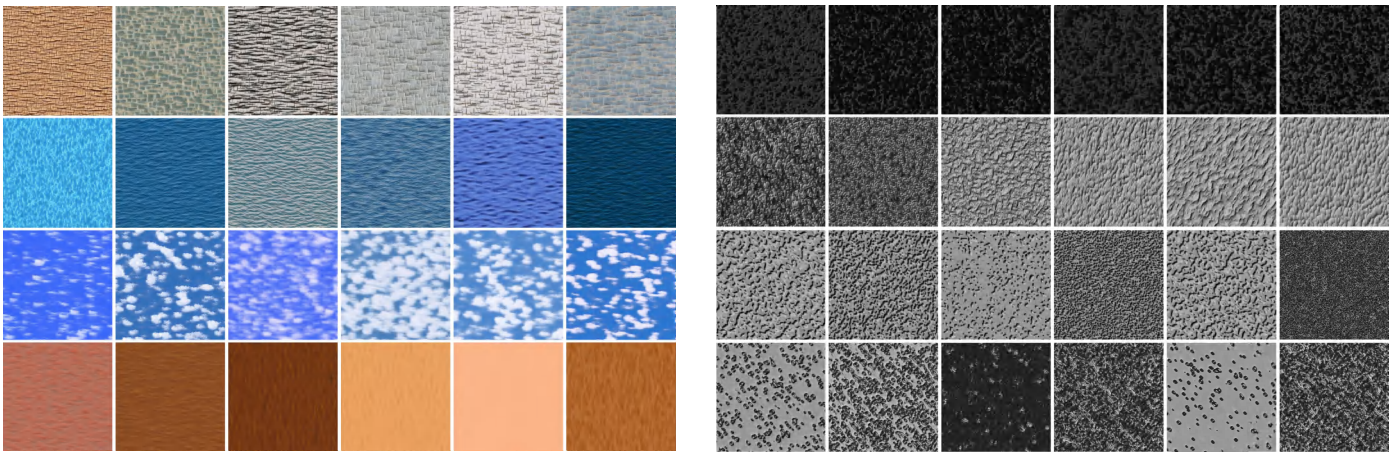


Fig. 7. Processing of the guidance signals in the CBN module of the generator. Demodulation is standard batch normalization for zero-centering and whitening of the features, while modulation shifts and scales the feature maps according to the input guidance. The dotted arrow indicates that the input guidance is either a categorical signal or a continuous signal, but not both at the same time, and only one data stream is enabled at a time.

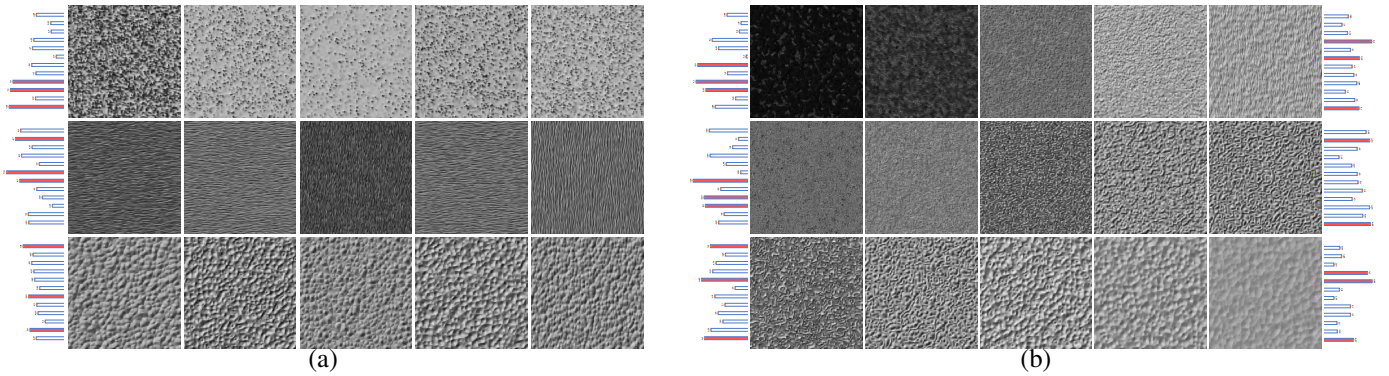
C. Generating Quality Textures from Coarse-Grained Control

The way humans perceive a texture is not easily quantifiable with statistics or metrics. In fact, in the absence of a recognized quantitative standard for image quality analysis [105], there is no standard metric to quantitatively evaluate the texture synthesis models [12], [13], [68]. Nonetheless, one can qualitatively assess whether a texture model generates the right visual characteristics according to the given guidance. Furthermore, qualitative analysis is a very common practice in the field of texture synthesis [12], [13], [68], [70], [106]. Hence,



(a) (b)

Fig. 8. Texture samples generated from categorical signals. All the textures in the figure are of 622×622 resolution (Zoom in to see more details).



(a) (b)

Fig. 9. Texture generation controlled by perceptual attributes (Zoom in to see the details). In both (a) and (b), the bar chart on the left-side of each row lists the 12 perceptual features: contrast, repetition, granularity, randomness, roughness, feature density, directionality, structural complexity, coarseness, regularity, local orientation, and uniformity in bottom-to-top order. In (b), the bar chart on the right-side of each row lists these 12 perceptual features but in top-to-bottom order. The textures in each row of (a) are generated from the same perceptual features depicted in the bar chart on the left side. In (b), the leftmost texture in each row is generated from the perceptual features depicted in the left bar chart, and the rightmost texture in each row is generated from the perceptual features depicted in the right bar chart, while the middle textures in each row are generated by interpolating the perceptual features in the latent space.

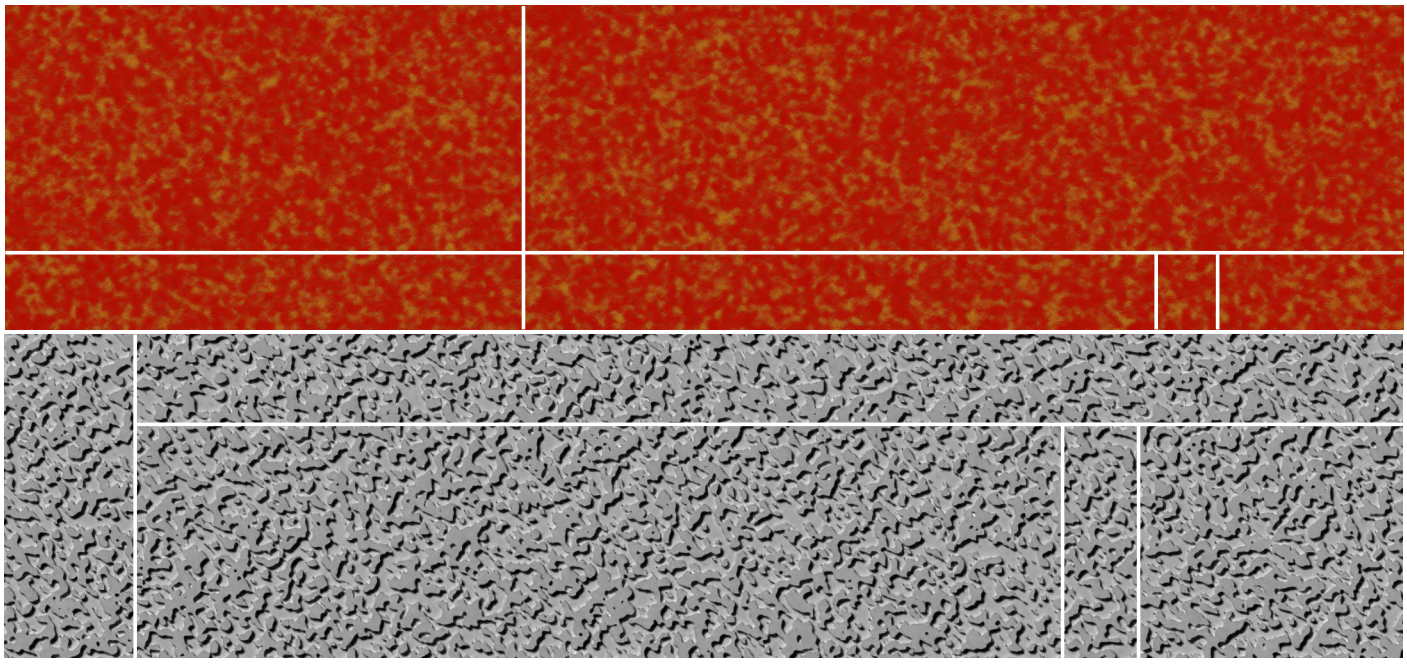


Fig. 10. Generating arbitrarily large textures. After the texture model has been trained on a particular dataset (here are HRTD at the top and HRTD-G at the bottom), it can be applied to generate textures of arbitrary scales, with coarse-grained signals as the interactive control on the appearance.

we intuitively display the generated textures to manifest the capability of the proposed method qualitatively.

We first train the texture model on HRTD and HRTD-G, respectively, and report the generated textures in Fig. 8. Textures in each row of Fig. 8 are generated from the same categorical signal, namely, 'bricks', 'water', 'sky' and 'wood', respectively, in Fig. 8 (a) and 'CA(forest fire model)', 'Folding cellular', 'CA(excitable media model)' and 'Cellular', respectively, in Fig. 8 (b). Fig. 9 (a) presents the generated textures by training the model on PTD. The left diagram in each row of Fig. 9 (a) indicates the values of the perceptual features that are used for generating the textures in this row. For clarity, the most prominent three perceptual features are colored in red. Observe that the textures in the first row of Fig. 9 (a) are predominant in contrast, granularity and randomness, and the textures in the second row of Fig. 9 (a) are significant in feature density, directionality, and local orientation, while the textures in the third row of Fig. 9 (a) show significant features in repetition, feature density and uniformity.

Note that our texture model can be applied to generate arbitrarily large textures of any shape even though it is trained on a dataset of a single scale. To illustrate this important capability, Fig. 10 displays the 'blanket' and 'CA(excitable media model)' textures generated by using our texture model after it has been trained on HRTD and HRTD-G, respectively.

D. Smooth Texture Metamorphosis

Existing texture synthesizers cannot generate customized textures, and cannot achieve plausible texture metamorphosis either. In our proposed texture model, flexible sampling in the noise space z_2 allows to create novel textures of potentially infinite output size, while high-level guidance permits to designate particular texture appearance. Furthermore, interpolation in the noise space z_1 allows smooth transition between textures of the same type, and interpolation in the disentangled latent space provides smooth transition between different types of textures. Fig. 9 (b) and Fig. 11 illustrate this advantage. Specifically, textures in Fig. 9 (b) persuades a visual translation between different perceptual descriptions. It can be seen that the textures in the first row of Fig. 9 (b) illustrate perceptual translation from granular to uniform with randomness and feature density retained, and the textures in the second row of Fig. 9 (b) demonstrate a decrease of feature density and an increase of uniformity, regularity and repetition, while the textures in the third row of Fig. 9 (b) show a decrease of structural complexity and contrast and an increase of randomness and roughness.

Fig. 11 describes several texture galleries, in which textures are generated using our generator by interpolating in the noise (z_1) or latent space. Specifically, textures in each row of Fig. 11 (a) and (b) demonstrate a visual translation within the

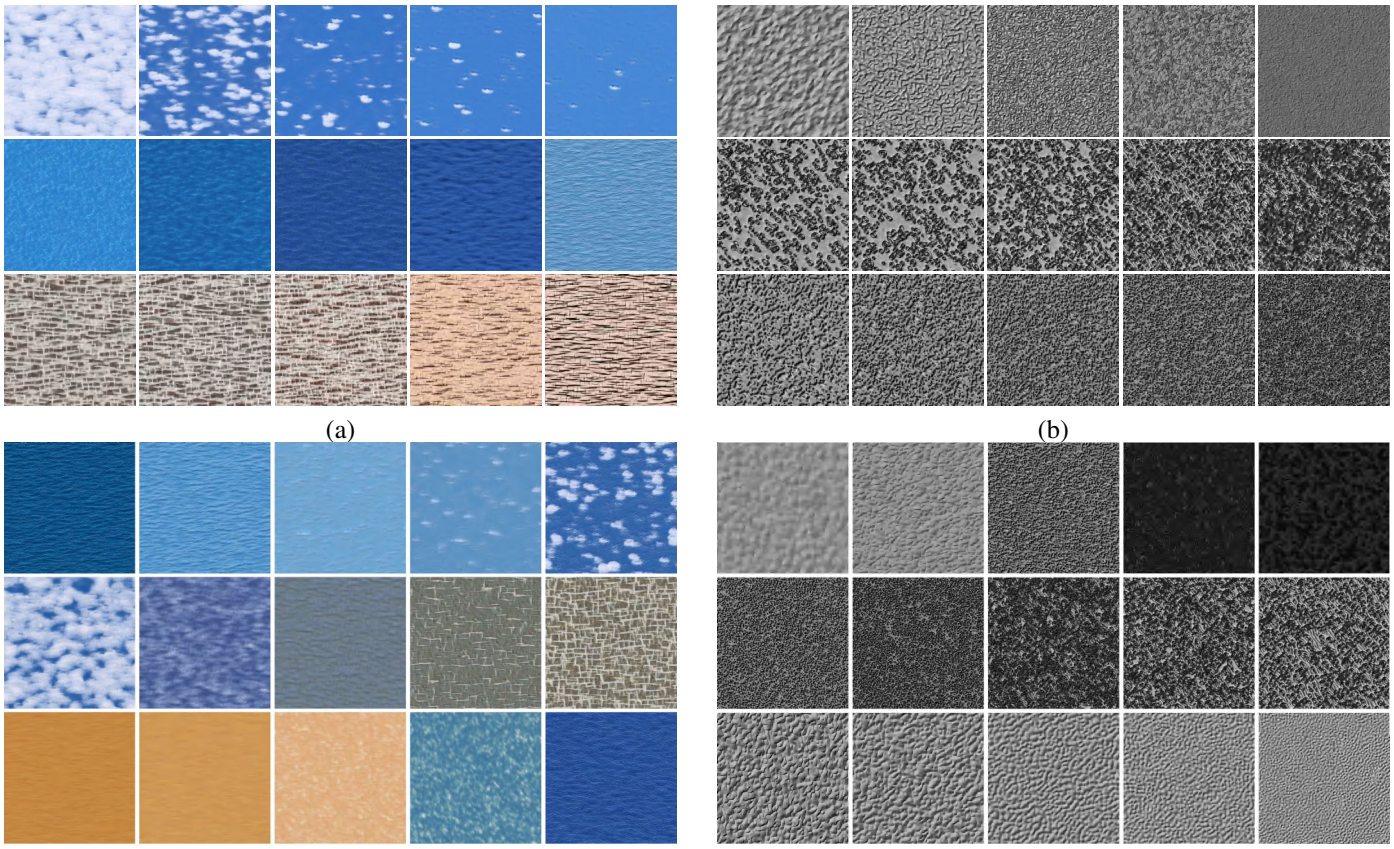


Fig. 11. Smooth texture metamorphosis (Zoom in to see the details). In each row of (a) and (b), textures are generated by applying the same categorical signal, specifically, 'sky', 'water' and 'bricks', respectively, in (a), and 'Folding perlin', 'Cellular' and 'CA(excitable media model), respectively, in (b). Observe smooth translation between textures of the same control, i.e., gradually disappeared clouds in the sky, gradually changed ripples of the water, gradually changed bricks, and gradually increased feature density of the texture. Texture galleries in (c) and (d) represent texture translation from one type to another, i.e., from water to sky, from sky to bricks, from wood to water, and from one procedural model to another.

TABLE I

ABLATION STUDY FOR THE PROPOSED TECHNIQUES. EACH CONFIGURATION IS TRAINED ON THE DATASETS WITH THE SAME SETUP. FID_S, FID_M, FID_L AND FID_H REPRESENT THE FID VALUES CALCULATED ON THE SCALES 146, 166, 186 AND 206, WHILE MSFID IS THE MEAN VALUE OF THEM.

	HRTD					HRTD-G					PTD				
	FID_S	FID_M	FID_L	FID_H	MSFID	FID_S	FID_M	FID_L	FID_H	MSFID	FID_S	FID_M	FID_L	FID_H	MSFID
Baseline	87.12	89.34	90.71	92.50	89.92	71.40	72.01	72.34	72.98	72.18	70.88	71.12	71.69	72.02	71.43
+Large Batchsize	85.42	86.46	87.65	89.66	87.30	69.34	69.36	69.43	69.51	69.41	68.15	68.20	68.23	68.40	68.25
+Proxy Distribution	83.08	85.92	87.74	90.85	86.90	68.12	68.78	69.02	69.56	68.87	66.34	66.50	66.51	66.61	66.49
+SE Noise Layer	83.10	84.25	86.50	88.43	85.57	67.10	67.23	67.40	67.58	67.33	62.98	63.11	63.27	63.35	63.18

same control, that is, different Markov random fields of the same control are sampled by interpolating in z_1. Textures in each row of Fig. 11 (c) and (d) demonstrate the visual translation between different guidance signals, that is, interpolation is performed in the disentangled latent space of c. Clearly our method achieves hallucinating texture metamorphosis and smooth texture translation in very natural manner.

E. Ablation Studies

In order to analyze the effectiveness of each proposal of our method, we use the 'plain' version of our model, in which all the bells and whistles (SE Noise Layer, Proxy Distribution) are removed, as the baseline. From this baseline, we perform control experiments by enabling the proposals one by one.

For quantitative comparison, we create a variant of the popular Fréchet inception distance (FID) [107], to measure the distribution distance between the generated textures and the ground truth textures. Specifically, to probe the content-growth ability, we calculate the FID values on progressively increasing scales (146, 166, 186, 206), and then average them. These scales are chosen because the texture models are trained on 146×146 texture patches, and the size of the training samples is 224×224 (see Subsection III-A). In terms of measuring the scalability of the texture synthesis models, it is meaningless to evaluate on a scale smaller than the training texture patches. Also it is impossible to evaluate the model on a larger scale than the training samples, since there are no ground truth textures. We refer to this variant of FID as MSFID. Indeed, MSFID is inspired by MS-SSIM [78] and c-FID [13], but MS-SSIM fails to react to image quality in terms of similarity to the training set and c-FID only operates on crops of images of the same size.

The experimental results are listed in Table I, where each configuration is separately trained on the datasets with the same hyperparameters. The baseline is trained with a batch size of 50, and Large Batchsize indicates a larger batch size of 100. From Table I, we can observe the performance improvement as each technique is enabled. This demonstrates the effectiveness of each proposal. Note that in a few cases (HRTD with scales 186 and 206, and HRTD-G with scale 206), proxy distribution does not improve the performance. This is because the proxy distribution does not always guarantee strictly positive.

F. Comparison with Other Methods

Since this is the first work aiming to synthesize textures directly from coarse-grained control, there is no target or benchmark method for comparison in the literature. To promote our method for this important application, we will illustrate the advantage of our method over 'adjusted' existing methods. With the current advances of image generation and texture synthesis, to synthesize an arbitrarily large texture from coarse-grained control, we can first use an image generation method to create a fixed-size texture from the coarse-grained control, and then apply a texture synthesis method to extend the content of the texture to a larger scale. We would like to apply the most celebrated image generation and texture synthesis methods to compose the combination rival of our method. The image generation and texture synthesis methods that catch our eyes include: BigGAN [67], StyleGAN [103], and StyleGAN2 [104] for image generation; DeepTextures [106], Texture Networks [11], SGAN [12], PSGAN [68], and Transposer [13] for texture synthesis.

Among the image generation methods, StyleGAN2 achieves the best results in most image generation tasks [104]. Hence, we choose StyleGAN2 as the implementation of the first stage. For texture synthesis, DeepTextures [106] needs to solve an optimization problem in the pixel space to synthesize a texture, while Texture Networks [11] and SGAN [12] have to train a separate deep network for each texture of interest. Therefore, these three methods are computationally prohibitive, since a quantitative evaluation of the produced image quality requires the calculation of the MSFID metric on a huge number of samples (in our experiments, each scale entails 20,000 texture patches, which is abbreviated as FID20K). Consequently, only PSGAN [68] and Transposer [13] can be employed as the synthesis model in the second stage, because they support learning multiple textures from datasets composed of one or more complex large images. However, Transposer [13] just doubles the size of the synthesized texture in each forward propagation, which does not help the implementation. Therefore, we construct the combination method by integrating StyleGAN2 and PSGAN together (abbreviated as S+PSGAN).

We compare our method with this combination method on the three datasets. It seems to be reasonable that we should first train StyleGAN2 on the datasets, and then train PSGAN on the same datasets. In this way, in the inference stage,

TABLE II

QUANTITATIVE PERFORMANCE COMPARISON FOR THE PROPOSED METHOD AND THE COMBINATION METHOD (S+PSGAN). FID_S, FID_M, FID_L AND FID_H REPRESENT THE FID VALUES CALCULATED ON THE SCALES 146, 166, 186 AND 206, WHILE MSFID IS THE MEAN VALUE OF THEM.

	HRTD					HRTD-G					PTD				
	FID_S	FID_M	FID_L	FID_H	MSFID	FID_S	FID_M	FID_L	FID_H	MSFID	FID_S	FID_M	FID_L	FID_H	MSFID
S+PSGAN	138.41	140.74	145.53	148.62	143.32	221.08	223.71	227.38	230.92	225.77	210.34	213.78	216.80	221.84	215.69
Proposed	83.10	84.25	86.50	88.43	85.57	67.10	67.23	67.40	67.58	67.33	62.98	63.11	63.27	63.35	63.18

PSGAN can successively undertake the textures generated by StyleGAN2. Nonetheless, when PSGAN is applied to a multifaceted dataset, the output can no longer be controlled in the inference stage. That is, PSGAN will randomly give a texture that it believes comes from the training data's distribution. In this case, we cannot directly utilize PSGAN to extend the contents of the textures generated by StyleGAN2. If we tie an individual PSGAN to each texture, although the above problem is solved, PSGAN immediately loses its flexibility. We adopt a roundabout way to tackle this hurdle. We first train StyleGAN2 with the default configuration (*config F*) on each dataset until convergence (totally 737k, 1,065k, and 3,522k real images shown to the discriminator on HRTD, HRTD-G and PTD, respectively). Since StyleGAN2 can only process square-shaped images with a power-of-two size and PSGAN outputs textures whose size is an integer multiple of 32, the texture crops used for training are resized from 146×146 to 256×256 on the fly, and the final outputs of the combination method are accordingly resized back to the original scale at the ratio of $146/256$. After StyleGAN2 has been trained, it is utilized to generate 20,000 textures, and the generated textures are then used to train PSGAN. When PSGAN is fully trained (100 epoches as default), it is applied to generate 1,000 (450 for PTD) textures of 384×384 (resized back to 219×219) for evaluation. The output resolution 384×384 is selected to make the restored size the closest to 224×224. We can safely assume that training the model at a higher resolution will not harm the performance, because the higher resolution only introduces redundant information but does not lose information.

Table II summarizes the quantitative comparison of our method and the combination method based on the FID metric. To obtain MSFID, on each scale, the FID20K value is calculated by looking over 20,000 crops from the synthesized textures and 20,000 crops from the dataset. In this manner, the calculated MSFID reflects an estimate of the synthesized image quality on average of tying a specific PSGAN to each texture generated by StyleGAN2. The estimate is reasonable on the condition that flexibility of the texture synthesis method needs to be considered. Furthermore, the accurate evaluation is impossible because training separate networks for 20,000 textures[2] will consume unaffordable computational resources. The experimental results of Table II demonstrate the superiority of our method over the combination method quantitatively. Surprisingly, the combination method performs very poorly on HRTD-G, which is a relatively simple dataset compared to HRTD according to the results of our method.

Qualitatively, an illustration of the textures synthesized using the combination method is given in Fig. 12 (the textures are also proportionally resized from the output resolution 1088×1088 to the benchmark resolution 622×622). It is obvious that the quality of the synthesized textures in Fig. 12 is much poorer compared to that of Fig. 8. In particular, the textures generated by the combination method tend to dully repeat the very basic structural elements. In addition to the inferior visual performance, the combination method consumes much more synthesis time (estimated by the sum of the generation time and synthesis time) than our method due to the inherent shortcoming of two-stage calculation. This efficiency issue becomes even more serious in practice. For the combination method, the heavy burden of training a neural network will probably be involved in the second stage when a particular texture is desired, because each texture thrown by the first stage is likely to be an unseen sample of the synthesis model.

[2]Although only 1,000 textures of 384×384 are sampled for evaluation, training PSGAN on 1,000 textures generated by StyleGAN2 may hamper the performance as many modalities will be absent from the training set.

Fig. 12. Texture samples generated by the combination method. The textures in the first, second, and third rows are generated by training the models (StyleGAN2 and PSGAN) within the combination method on HRTD, HRTD-G and PTD, respectively. All the textures in the figure are of 622×622 resolution.

G. Practical Application

The motivation of our work comes from practical demand. For example, designers of game companies would like an efficient method to acquire arbitrarily large and visually diverse textures by simply providing the semantic descriptions to create realistic backgrounds for game scenes. To this end, we have implemented a demonstration system that can produce high-quality textures according to the user's inputs, and provide previews, individual and batched download of the generated textures. Fig. 13 is a demonstration of the system, where textures of certain type and size are produced. This system provides a convenient way for even inexperienced designers to obtain high-quality textures, and it further verifies the effectiveness of our proposed method. Although the system only supports the basic demonstration functions currently, it can easily be extended to offer a comprehensive design toolkit. The development of a complete game software is beyond the

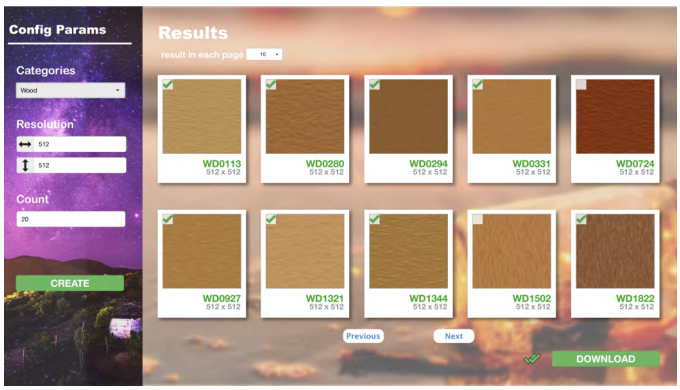


Fig. 13. Interactive user interface of the proposed texture generation system. In this diagram, 20 textures of 512x512 from the "wood" category are generated.

scope of this paper. Potential users in other related fields, wallpaper production, film post-production, animation, etc., can also use the system to experience the convenience of texture generation at any scale under coarse-grained control, and obtain inspiration for their particular applications.

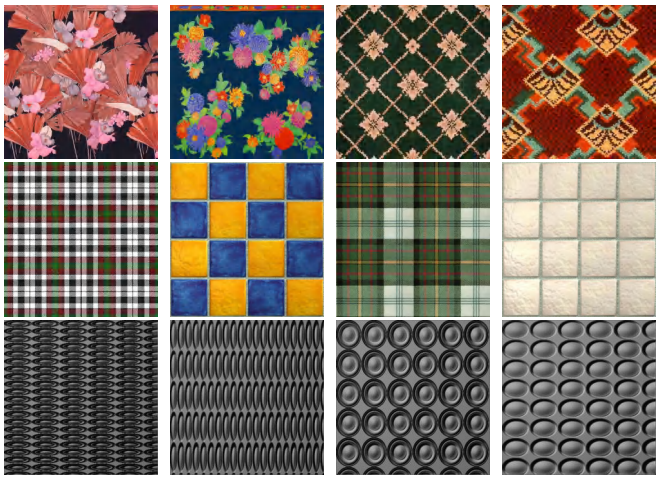


Fig. 14. Missed modalities of our texture model. These textures either have complex structures (not stationary) or regular arrangements (not stochastic). The displayed textures are all of their original resolutions in the datasets.

H. Limitations

Although we impress the conversion from z_2 to t with a global influence by adjusting the Markov random field's parameters through c and z_1, we find that the learned Markov random field lacks the ability of spatial alignment. We believe that the responsibility lies in the Markov random field itself, which models data through local properties and thus is unable to capture long-range dependencies. Fig. 14 gives the missed cases of our texture model. Textures in Fig. 14 are samples from the datasets, but absent from the generated results of our texture model. Obviously, these missed modalities are all regular or non-stationary textures whose formation requires spatially global interaction. In a nutshell, our learned texture model can only support generating random textures strictly conformable with a positive Markov random field.

We reiterate that the purpose of the proposed method is to solve the coarse-grained controllable texture synthesis, that is, to establish the mapping relationship between human-understandable coarse-grained control variables and texture images of arbitrary scales. The proposed method does not support texture synthesis under fine-grained control (the working paradigm of example-based texture synthesis), and cannot synthesize a larger-scale texture image given an exemplar texture. We further point out that texture synthesis under coarse-grained control and texture synthesis under fine-grained control can complement each other in various application scenarios to complete various texture synthesis tasks, while constructing a large unified model that supports both coarse-grained controllable texture synthesis and fine-grained controllable texture synthesis is not very necessary.

I. Latency Analysis

Since the texture model aims to learn enormous Markov random fields for different guidance signals to support generation of different types of textures, it takes a relatively long time for training. However, once the training is completed, the texture model is very efficient at synthesis. Generally, it takes about 9 hours to train on HRTD, 7 hours to train on HRTD-G, and 5 hours to train on PTD. Because the model supports generation of textures of arbitrary scale, it consumes on average 0.25 (0.22) microseconds for one color (gray) pixel in the synthesis stage. In other words, it can produce a 512×512 color (gray) texture within 0.07 (0.06) seconds[3].

IV. Concluding Remarks

In this paper, we have presented a framework for arbitrary-scale texture generation from coarse-grained control. Although deconvolution approaches allow generation of textures over a variety of natural images, they need to solve a complicated optimization problem with iterative back propagation just for creating a single output texture. Fitting a convolutional network in a feed-forward fashion can improve the efficiency of deep texture synthesis, but most existing approaches compose texture synthesizer of specific style, where separate networks have to be trained when various textures need to be modeled. The challenge is to find the right balance between tractability and flexibility. Our work has contributed to this endeavour by introducing a fully tractable yet flexible texture model. In addition, existing texture models focus on fine-grained control of texture generation (synthesis, stylization) but controllable synthesis with user interaction is even more useful. The proposed method has provided an efficient way for generating textures directly from coarse-grained controls, which makes it convenient for users to author realistic visual content according to their own desire. Extensive experiments have demonstrated that the proposed method is effective at generating high-quality and large-scale textures from high-level guidance.

However, since Markov random fields are employed at the bottom of the Bayesian hierarchy for our texture modeling, the learned texture model can only generate stochastic textures. In our future work, we will investigate more universal texture models for both random and regular texture synthesis.

References

[1] D. Kersten, "Predictability and redundancy of natural images," *JOSA A*, vol. 4, no. 12, pp. 2395–2400, 1987.

[2] D. J. Field, "Relations between the statistics of natural images and the response properties of cortical cells," *JOSA A*, vol. 4, no. 12, pp. 2379–2394, 1987.

[3] J. G. Daugman, "Entropy reduction and decorrelation in visual coding by oriented neural receptive fields," *IEEE Trans. Biomedical Engineering*, vol. 36, no. 1, pp. 107–114, Jan. 1989.

[4] D. L. Ruderman and W. Bialek, "Statistics of natural images: Scaling in the woods," *Phys. Rev. Lett.*, vol. 73, no. 6, pp. 814–817, 1994.

[5] J. Portilla and E. P. Simoncelli, "A parametric texture model based on joint statistics of complex wavelet coefficients," *Int. J. Computer Vision*, vol. 40, no. 1, pp. 49–70, 2000.

[6] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *Proc. ICCV 1999* (Kerkyra, Greece), Sep. 20-27, 1999, pp. 1033–1038.

[7] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proc. CVPR 2016* (Las Vegas, NV, USA), Jun. 27-30, 2016, pp. 2414–2423.

[3] All the training and inference are performed on the computer with Ubuntu 16.04.6 LTS, Pytorch 1.1.0, 2 Intel(R) Xeon(R) CPU E5-2678 v3, 2 NVIDIA TITAN RTX GPUs and 64 GB memory.

[8] G. Georgiadis, A. Chiuso, and S. Soatto, "Texture compression," in *Proc. DCC 2013*, (Snowbird, UT, USA), Mar. 20-22, 2013, pp. 221–230.

[9] W. Xian, *et al.*, "TextureGAN: Controlling deep image synthesis with texture patches," in *Proc. CVPR 2018* (Salt Lake City, UT, USA), Jun. 18-23, 2018, pp. 8456–8465.

[10] C. Li and M. Wand, "Combining Markov random fields and convolutional neural networks for image synthesis," in *Proc. CVPR 2016* (Las Vegas, NV, USA), Jun. 26-Jul. 1, 2016, pp. 2479–2486.

[11] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky, "Texture networks: Feed-forward synthesis of textures and stylized images," in *Proc. ICML 2016* (New York City, NY, USA), Jun. 19-24, 2016, pp. 1349–1357.

[12] N. Jetchev, U. Bergmann, and R. Vollgraf, "Texture synthesis with spatial generative adversarial networks," *arXiv:1611.08207*, 2016. [Online]. Available: http://arxiv.org/abs/1611.08207

[13] G. Liu, *et al.*, "Transposer: Universal texture synthesis using feature maps as transposed convolution filter," *arXiv:2007.07243*, 2020. [Online]. Available: https://arxiv.org/abs/2007.07243

[14] V. Dumoulin, J. Shlens, and M. Kudlur, "A learned representation for artistic style," in *Proc. ICLR 2017* (Toulon, France), Apr. 24-26, 2017, pp. 1–26.

[15] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proc. CVPR 2017* (Honolulu, HI, USA), Jul. 21-26, 2017, pp. 1125–1134.

[16] W. Chen, H. Fang, J. Ding, and S. Kuo, "PMHLD: Patch map-based hybrid learning dehazeNet for single image haze removal," *IEEE Trans. Image Process.*, vol. 29, pp. 6773–6788, 2020.

[17] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Deep image prior," *Int. J. Computer Vision*, vol. 128, no. 7, pp. 1867–1888, 2020.

[18] T. R. Shaham, T. Dekel, and T. Michaeli, "SinGAN: Learning a generative model from a single natural image." in *Proc. ICCV 2019* (Seoul, Korea), Oct. 27-Nov. 2, 2019, pp. 4569–4580.

[19] G. Wang, C. Sun, and A. Sowmya, "Cascaded attention guidance network for single rainy image restoration," *IEEE Trans. Image Process.*, vol. 29, pp. 9190–9203, 2020.

[20] S. Guo, *et al.*, "Toward convolutional blind denoising of real photographs," in *Proc. CVPR 2019* (Long Beach, CA, USA), Jun. 16-20, 2019, pp. 1712–1722.

[21] T. Li, Y. Chan, and D. P. Lun, "Improved multiple-image-based reflection removal algorithm using deep neural networks," *IEEE Trans. Image Process.*, vol. 30, pp. 68–79, 2021.

[22] Y. Gan, *et al.*, "Perception driven texture generation," in *Proc. ICME 2017* (Hong Kong, China), Jul. 10-14, 2017, pp. 889–894.

[23] S. Wolfram, "Statistical mechanics of cellular automata," *Reviews of Modern Physics*, vol. 55, no. 3, pp. 601–644, 1983.

[24] J.-M. Dischler and D. Ghazanfarpour, "A survey of 3D texturing," *Computers & Graphics*, vol. 25, no. 1, pp. 135–151, 2001.

[25] J. Liu, *et al.*, "Perception-driven procedural texture generation from examples," *Neurocomputing*, vol. 291, pp. 21–34, 2018.

[26] L. McMillan and G. Bishop, "Plenoptic modeling: An image-based rendering system," in *Proc. SIGGRAPH 1995* (Los Angeles, CA, USA), Aug. 6-11, 1995, pp. 39–46.

[27] J. Lalonde, *et al.*, "Photo clip art," *ACM Trans. Graphics*, vol. 26, no. 3, p. 3-1–3-10, 2007.

[28] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. ICLR 2014* (Banff, AB, Canada), Apr. 14-16, 2014, pp. 1–14.

[29] I. J. Goodfellow, *et al.*, "Generative adversarial nets," in *Proc. NIPS 2014* (Montreal, Quebec, Canada), Dec. 8-13, 2014, pp. 2672–2680.

[30] J. Hays and A. A. Efros, "Scene completion using millions of photographs," *ACM Trans. Graphics*, vol. 26, no. 3, p. 4-1–4-8, 2007.

[31] J. M. Hammersley and P. Clifford, "Markov fields on finite graphs and lattices," *Unpublished manuscript*, pp. 1–26, 1971.

[32] P. Clifford, "Markov random fields in statistics," in: G. Grimmett and D. Welsh (eds.) *Disorder in Physical Systems: A Volume in Honour of John M. Hammersley*, Oxford University Press, 1990, pp. 19–32.

[33] M. Hassner and J. Sklansky, "The use of Markov random fields as models of texture," *Computer Graphics and Image Modeling*, vol. 12, no. 4, pp. 357–370, 1980.

[34] G. R. Cross and A. K. Jain, "Markov random field texture models," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-5, no. 1, pp. 25–39, 1983.

[35] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721–741, 1984.

[36] H. Derin and H. Elliott, "Modeling and segmentation of noisy and textured images using Gibbs random fields," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 1, pp. 39–55, 1987.

[37] D. Geman, S. Geman, C. Graffigne, and P. Dong, "Boundary detection by constrained optimization," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, no. 7, pp. 609–628, 1990.

[38] S. Z. Li, *Markov Random Field Modeling in Image Analysis*. Springer-Verlag: London, 2009.

[39] Z. Liu, *et al.*, "Deep learning Markov random field for semantic segmentation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 40, no. 8, pp. 1814–1828, 2018.

[40] H. Tjelmeland and J. Besag, "Markov random fields with higher-order interactions," *Scandinavian J. Statistics: Theory and Applications*, vol. 25, no. 3, pp. 415–433, 1998.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR 2016* (Las Vegas, NV, USA), Jun. 26-Jul. 1, 2016, pp. 770–778.

[42] C. Szegedy, *et al.*, "Rethinking the inception architecture for computer vision," in *Proc. CVPR 2016* (Las Vegas, NV, USA), Jun. 26-Jul. 1, 2016, pp. 2818–2826.

[43] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. CVPR 2017* (Honolulu, HI, USA), Jul. 21-26, 2017, pp. 4700–4708.

[44] Y. Chen, *et al.*, "Dual path networks," in *Proc. NIPS 2017* (Long Beach, CA, USA), Dec. 4-9, 2017, pp. 4467–4475.

[45] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. CVPR 2018* (Salt Lake City, UT, USA), Jun. 18-23, 2018, pp. 6848–6856.

[46] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. CVPR 2018* (Salt Lake City, UT, USA), Jun. 18-23, 2018, pp. 7132–7141.

[47] A. G. Howard, *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[48] A. van Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *Proc. ICML 2016* (New York City, NY, USA), Jun. 19-24, 2016, pp. 1747–1756.

[49] A. Razavi, A. van den Oord, and O. Vinyals, "Generating diverse high-fidelity images with VQ-VAE-2," in *Proc. NeurIPS 2019* (Vancouver, Canada), Dec. 8-14,2019, pp. 1–11.

[50] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, 2016. [Online]. Available: http://distill.pub/2016/deconv-checkerboard

[51] M. Lucic, *et al.*, "High-fidelity image generation with fewer labels," in *Proc. ICML 2019* (Long Beach, CA, USA), Jun. 9-15, 2019, pp. 4183–4192.

[52] H. Zhang, I. J. Goodfellow, D. N. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *Proc. ICML 2019* (Long Beach, CA, USA), Jun. 9-15, 2019, pp. 7354–7363.

[53] L. Dinh, D. Krueger, and Y. Bengio, "NICE: Non-linear independent components estimation," in *Proc. ICLR 2015* (San Diego, CA, USA), May 7-9, 2015, pp. 1–13.

[54] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proc. ICML 2014* (Beijing, China), Jun. 21-26, 2014, pp. 1278–1286.

[55] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *Proc. ICLR 2017* (Toulon, France), Apr. 24-26, 2017, pp. 1–32.

[56] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Proc. NeurIPS 2018* (Montréal, Canada), Dec. 2-8, 2018, pp. 10236–10245.

[57] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv:1411.1784*, 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[58] Y. Li, K. Swersky, and R. S. Zemel, "Generative moment matching networks," in *Proc. ICML 2015* (Lille, France), Jul. 6-11, 2015, pp. 1718–1727.

[59] G. K. Dziugaite, D. M. Roy, and Z. Ghahramani, "Training generative neural networks via maximum mean discrepancy optimization," in *Proc. UAI 2015* (Amsterdam, Netherlands), Jul. 12-16, 2015, pp. 258–267.

[60] D. J. Sutherland, *et al.*, "Generative models and model criticism via optimized maximum mean discrepancy," in *Proc. ICLR 2017* (Toulon, France), Apr. 24-26, 2017, pp. 1–13.

[61] C. Li, *et al.*, "MMD GAN: Towards deeper understanding of moment matching network," in *Proc. NIPS 2017* (Long Beach, CA, USA), Dec. 4-9, 2017, pp. 2203–2213.

[62] M. Binkowski, D. J. Sutherland, M. Arbel, and A. Gretton, "Demystifying MMD GANs," in *Proc. ICLR 2018* (Vancouver, BC, Canada), Apr. 30-May 3, 2018, pp. 1–36.

[63] C. N. dos Santos, Y. Mroueh, I. Padhi, and P. Dognin, "Learning implicit generative models by matching perceptual features," in *Proc. ICCV 2019* (Seoul, Korea), Oct. 27-Nov. 2, 2019, pp. 4461–4470.

[64] R. M. Neal, "Annealed importance sampling," *Statistics and Computing*, vol. 11, no. 2, pp. 125–139, 2001.

[65] L. Wasserman, *All of Statistics: A Concise Course in Statistical Inference*. Springer Science & Business Media, 2013.

[66] I. Gulrajani, *et al.*, "Improved training of Wasserstein GANs," in *Proc. NIPS 2017* (Long Beach, CA, USA), Dec. 4-9, 2017, pp. 5769–5779.

[67] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," in *Proc. ICLR 2019* (New Orleans, LA, USA), May 6-9, 2019, pp. 1–35.

[68] U. Bergmann, N. Jetchev, and R. Vollgraf, "Learning texture manifolds with the periodic spatial GAN," in *Proc. ICML 2017* (Sydney, NSW, Australia), Aug. 6-11, 2017, pp. 469–477.

[69] A. Shocher, S. Bagon, P. Isola, and M. Irani, "InGAN: Capturing and remapping the 'DNA' of a natural image," in *Proc. ICCV 2019* (Seoul, Korea), Oct. 27-Nov. 2, 2019, pp. 1–10.

[70] C. Li and M. Wand, "Precomputed real-time texture synthesis with Markovian generative adversarial networks," in *Proc. ECCV 2016* (Amsterdam, Netherlands), Oct. 11-14, 2016, pp. 702–716.

[71] Y. Li, *et al.*, "Universal style transfer via feature transforms," in *Proc. NISP 2017* (Long Beach, CA, USA), Dec. 4-9, 2017, pp. 386–396.

[72] T.-Y. Chiu, "Understanding generalized whitening and coloring transform for universal style transfer," in *Proc. ICCV 2019* (Seoul, Korea), Oct. 27-Nov. 2, 2019, pp. 4451–4459.

[73] H. Chi, *et al.*, "Semantic attributes based texture generation," in *Proc. ICGIP 2017* (Qingdao, China), Oct. 13-15, 2018, pp. 1–10.

[74] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *Proc. ICLR 2016* (San Juan, Puerto Rico), May 2-4, 2016, pp. 1–16.

[75] X. Mao, *et al.*, "Least squares generative adversarial networks," in *Proc. ICCV 2017* (Venice, Italy), Oct. 22-29, 2017, pp. 2813–2821.

[76] J. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, "Generative visual manipulation on the natural image manifold," in *Proc. ECCV 2016* (Amsterdam, Netherlands), Oct. 11-14, 2016, pp. 597–613.

[77] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proc. CVPR 2017* (Honolulu, HI, USA), Jul. 21-26, 2017, pp. 5967–5976.

[78] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *Proc. ICML 2017* (Sydney, NSW, Australia), Aug. 6-11, 2017, pp. 2642–2651.

[79] M. Zhu, P. Pan, W. Chen, and Y. Yang, "DM-GAN: Dynamic memory generative adversarial networks for text-to-image synthesis," in *Proc. CVPR 2019* (Long Beach, CA, USA), Jun. 16-20, 2019, pp. 5802–5810.

[80] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," in *Proc. ICLR 2017* (Toulon, France), Apr. 24-26, 2017, pp. 1–17.

[81] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv:1701.07875*, 2017. [Online]. Available: http://arxiv.org/abs/1701.07875

[82] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. CVPR 2015* (Boston, MA, USA), Jun. 7-12, 2015, pp. 3431–3440.

[83] B. Julesz, "Visual pattern discrimination," *IRE Trans. Information Theory*, vol. 8, no. 2, pp. 84–92, 1962.

[84] W. Roth and F. Pernkopf, "Bayesian neural networks with weight sharing using Dirichlet processes," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 42, no. 1, pp. 246–252, 2020.

[85] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *Proc. ICLR 2018* (Vancouver, BC, Canada), Apr. 30-May 3, 2018, pp. 1–26.

[86] T. Salimans, *et al.*, "Improved techniques for training GANs," in *Proc. NIPS 2016* (Barcelona, Spain), Dec. 5-10, 2016, pp. 2234–2242.

[87] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Proc. NIPS 2016* (Barcelona, Spain), Dec. 5-10, 2016, pp. 901–909.

[88] A. Odena, *et al.*, "Is generator conditioning causally related to GAN performance?" in *Proc. ICML 2018* (Stockholm, Sweden), Jul. 10-15, 2018, pp. 3846–3855.

[89] Y. Yoshida and T. Miyato, "Spectral norm regularization for improving the generalizability of deep learning," *arXiv:1705.10941*, 2017. [Online]. Available: http://arxiv.org/abs/1705.10941

[90] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. ICML 2015* (Lille, France), Jul. 6-11, 2015, pp. 448–456.

[91] B. Singh, M. Najibi, and L. S. Davis, "SNIPER: Efficient multi-scale training," in *Proc. NIPS 2018* (Montreal, Canada), Dec. 2-8, 2018, pp. 1–11.

[92] Y. Wu, Y. Burda, R. Salakhutdinov, and R. B. Grosse, "On the quantitative analysis of decoder-based generative models," in *Proc. ICLR 2017* (Toulon, France), Apr. 24-26, 2017, pp. 1–17.

[93] C. K. Sønderby, *et al.*, "Amortised MAP inference for image super-resolution," in *Proc. ICLR 2017* (Toulon, France), Apr. 24-26, 2017, pp. 1–17.

[94] L. M. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for GANs do actually converge?" in *Proc. ICML 2018* (Stockholm, Sweden), Jul. 10-15, 2018, pp. 3478–3487.

[95] K. J. Dana, B. van Ginneken, S. K. Nayar, and J. J. Koenderink, "Reflectance and texture of real-world surfaces," *ACM Trans. Graphics*, vol. 18, no. 1, pp. 1–34, 1999.

[96] T. Ojala, *et al.*, "Outex – new framework for empirical evaluation of texture analysis algorithms," in *Proc. ICPR 2002* (Quebec, Canada), Aug. 11-15, 2002, pp. 701–706.

[97] G. Kylberg, "The kylberg texture dataset v. 1.0," *Report 35*, Centre for Image Analysis, Swedish University of Agricultural Sciences and Uppsala University, Uppsala, Sweden, September 2011.

[98] https://vismod.media.mit.edu/vismod/imagery/VisionTexture/vistex.html.

[99] M. Cimpoi, *et al.*, "Describing textures in the wild," in *Proc. CVPR 2014* (Columbus, OH, USA), Jun. 23-28, 2014, pp. 3606–3613.

[100] J. Liu, J. Dong, L. Qi, and M. Chantler, "Identifying perceptual features of procedural textures," *Perception*, vol. 42, pp. 221–221, 2013.

[101] J. Liu, *et al.*, "Visual perception of procedural textures: Identifying perceptual dimensions and predicting generation models," *PloS one*, vol. 10, no. 6, pp. 1–22, 2015.

[102] A. R. Rao and G. L. Lohse, "Towards a texture naming system: Identifying relevant dimensions of texture," *Vision Research*, vol. 36, no. 11, pp. 1649–1669, 1996.

[103] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proc. CVPR 2019* (Long Beach, CA, USA), Jun. 16-20, 2019, pp. 4401–4410.

[104] T. Karras, *et al.*, "Analyzing and improving the image quality of styleGAN," in *Proc. CVPR 2020*, Jun. 14-19, 2020, pp. 8110–8119.

[105] L. Theis, A. van den Oord, and M. Bethge, "A note on the evaluation of generative models," in *Proc. ICLR 2016* (San Juan, Puerto Rico), May 2-4, 2016, pp. 1–10.

[106] L. A. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," in *Proc. NIPS 2015* (Montreal, Quebec, Canada), Dec. 7-10, 2015, pp. 262–270.

[107] M. Heusel, *et al.*, "GANs trained by a two time-scale update rule converge to a local nash equilibrium," in *Proc. NIPS 2017* (Long Beach, CA, USA), De. 4-9, 2017, pp. 6629–6640.