Received April 6, 2020, accepted April 17, 2020, date of publication April 23, 2020, date of current version May 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2989898

Fast Texture Synthesis for Discrete Example-Based Elements

ZHENGRUI HUANG[1,2], XIAOHONG LIN[3], AND CHONGCHENG CHEN[1,2], (Member, IEEE)
[1]Key Laboratory of Spatial Data Mining and Information Sharing of Ministry Education, Fuzhou University, Fuzhou 350108, China
[2]National Engineering Research Center of Geospatial Information Technology, Fuzhou 350108, China
[3]Fuzhou Technology and Business College, Fuzhou 350007, China

Corresponding author: Chongcheng Chen (chencc@fzu.edu.cn)

This work was supported in part by The National Key Research and Development Project under Grant 2017YFB0504202.

ABSTRACT Considering the problem of discrete texture synthesis and the time for texturing, this paper proposes a novel framework for synthesizing texture images based on discrete example-based elements. We start with extracting texture feature distribution from exemplars and then produce discrete elements based on the cluster algorithm. After initializing a texture image, we propose a texture optimization algorithm based on heuristic searching to improve the quality of the texture image. Final, we use a texture transfer method based on Convolutional Neural Network (CNN) to stylize the optimized texture image. Our results show that the proposed texture synthesis method can significantly improve the quality of discrete texture synthesis and effectively shorten the time for texture generation.

INDEX TERMS Texture synthesis, discrete elements, cluster algorithm, heuristic searching, CNN.

I. INTRODUCTION

The extraction and analysis of texture is the key issue in the technical system of computer vision [1]. However, the technologies for further applying extracted texture need to be explored. Such applications include texture mapping, rendering and synthesis [2]. Nowadays, many studies and methods for texture mapping and rendering have been discussed (see [17] to [31]), but it is still worthy to discuss how to realize effective texture synthesis based on different texture properties that can be divided into two classes: discrete texture and continuous texture. For this purpose, it is necessary to propose a method that can integrate the process of feature extraction, optimization, synthesis and even stylization [3]. To our best knowledge, there are many pixel-based or patch-based methods for texturing based on mathematical statistical models [4], such as Markov Random Field (MRF) model. However, these pixel-based or patch-based methods are more suitable for synthesizing continuous texture [2], as shown in Fig. 1, and most of them are quite time-consuming because such methods need to grow texture pixel by pixel or patch by patch. Indeed, methods for synthesizing discrete texture are still limited.

The associate editor coordinating the review of this manuscript and approving it for publication was Gianluigi Ciocca.

In this paper, to achieve fast texture synthesis for discrete texture, we propose an integrative method to achieve this goal, including five subsections. First, we obtain the feature distribution based on images gradient information. Second, discrete example-based elements can be extracted based on image feature distribution. Here, we adopt the mean shift algorithm to determine the number of discrete elements and extract them from the exemplar discretely. Next, we initialize texture images in a stochastic or a regular way and store their properties (such as respective position, angle and structure) with a special method to speed up the synthesis process in following subsections. Then, we propose an optimization algorithm to improve the quality of texture images based on heuristic searching. Final, we use the CNN-based texture transfer technology to stylize the optimized texture images. Compared with some existing methods (such as [18], [20] and [28]) for texturing that may produce some image seams and just optimize the local texture distribution (see Fig. 12), our method starts with preserving the integrity of elements in the exemplar, then optimizes the elements distribution globally, and can save more time for texturing process. Moreover, the overall synthesis process is automatic (users only need to prepare their own materials), and it is quite easy to transfer our model to a 3D surface according to user needs. The final stylized texture images (see Fig. 13) that produce some strong visual effects are also interesting.

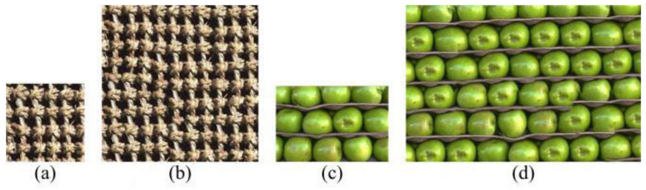


FIGURE 1. The patch-based method for texture synthesis [19]. (b) is the output image of continuous texture from (a). (d) is the output image of discrete texture from (c) that may present some image seams.

The rest of this paper is structured as follows. In Section II, we present an introduction of related work. Section III introduces the proposed process of texture synthesis. The results are presented in Section IV, and the conclusions are drawn in Section V.

II. RELATED WORK

A. FEATURE EXTRACTION

Common feature extraction methods mainly include: statistical methods, transform-based methods and model-based methods.

For statistical methods, the feature distribution of texture images is extracted by some statistical models, such as a gray-level model, a gradient histogram model and a mapped pattern-based model. In [5], the gray level co-occurrence matrix (GLCM) is used to extract second or higher order feature from RGB images. Following [5], GLCM is further applied to other scenarios, such as extracting structural information from seismic attributes [6] and monitoring changes in baguettes [7]. Furthermore, the authors in [8] propose a texture descriptor for image segmentation based on the histogram of image gradient, and the research in [9] designs a method to recognize buildings according to the histogram of oriented gradient (HOG). In this paper, we also extract image feature based on image gradient changes. Further, users can adopt some methods, such as Log-Polar Transformation (LPT), to reduce the rotation effects.

For transform-based methods, each texture image can be transformed into a frequency space or a scale space to interpret its texture. In [10], a texture classification method is proposed by applying 1D or 2D Fourier filter to extract local frequency information. Following [10], the authors propose an effective method for texture representation based on wedge filters and local descriptors [11]. To extract more image information, the author in [12] uses a copula model to decompose texture information produced by Circularly Symmetric Gabor Wavelet (CSGW).

For model-based methods, some mathematical models are adopted. Among these models, the complex network model has better effect than others, such as a gravitational model [13] or an autoregressive model [14]. In [15], a vocabulary of words based on Complex Network (CN) is built to analyze texture. Following [15], a local spatial pattern mapping (LSPM) is proposed to classify texture images [16].

B. TEXTURE SYNTHESIS

Common methods for texture synthesis mainly include: pixel-based methods, patch-based methods and example-based methods.

For pixel-based methods, the basic idea stems from [17] that designs a non-parameter synthesis method by extending a noise image pixel by pixel. Following [17], the authors in [18] propose a synthesis method based on MRF and adopt a tree-structured vector to speed up such process. In [19], a method called "image quilting" is proposed by minimizing the cost path through the overlapped areas horizontally and vertically to stitch input texture images. Such method is also effective for resizing images by reduction and expansion operation [20].

For patch-based methods, the main idea is to speed up the synthesis process by overlapping patches instead of growing pixels. In [21], the authors first extract the texture patches from exemplars and then copy them into a 3D surface seamlessly by a liner model. The research in [22] presents an analytical method for texture synthesis by decomposing an exemplar into patches and recomposing them on a 3D surface directly. Following [22], a graph-cut method is developed to compute the optimal patch size. The extracted patches can be copied into the output image dynamically [23], and the method in [23] is also suitable for video synthesis. Moreover, a search-based algorithm for texturing is proposed by constructing a image pyramid and using a cache reuse technology [24]. Compared with pixel-based methods, patch-based methods are more efficient.

For example-based methods, an input image is usually regarded as a complete and small element, and the synthesis process is similar to assemble these elements. In [25], a data-driven method for discrete texture synthesis is proposed, but such method requires users to determine their materials structures in advance. Following [25], the authors propose a global optimization algorithm to generate large building models based on small ones [26], and a tile-based method is proposed in [27] to generate facade building images. With the advent of neural networks, a method based on the long-short term memory (LSTM) that stems from Recurrent Neural Network (RNN) is proposed to produce regular texture [28], a feed-forward network (FFN) is used to synthesize diverse texture by interpolating texture images generated by different FNN layers and maximizing stylized texture quality [29], [30], and the authors in [31] propose a conditional generative CNN-based (CGCNN) algorithm for synthesizing example-based texture with non-local structures. Moreover, the research in [1] designs a novel method for growing texture over a 3D surface, and a camera-aided texturing method is proposed by integrating artistic tools into the texture synthesis system [32]. In this paper, the main idea for texture synthesis also stems from example-based methods.

III. METHODOLOGY

Our method mainly consists of five parts, as shown in Fig. 2.

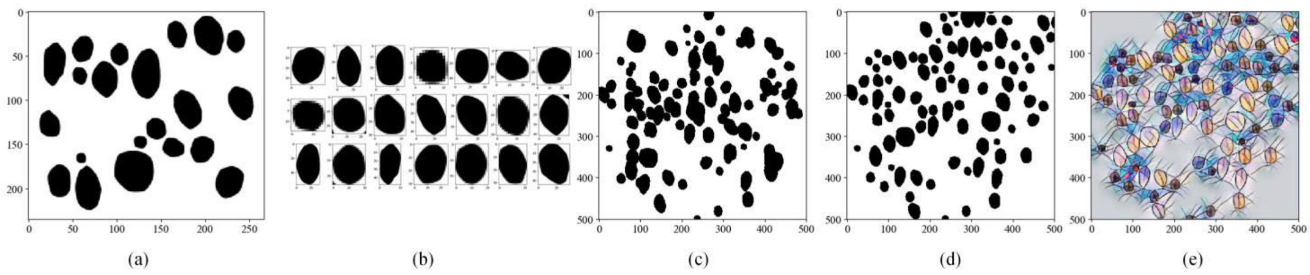
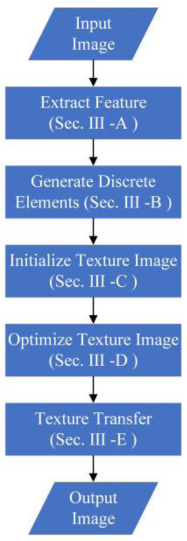


FIGURE 2. Texture synthesis flow chart. (a) is an exemplar. A feature extraction method is used to obtain discrete example-based elements in (b). In (c), we initialize a texture image in a stochastic way. The regular initialization method can be also used, as shown in Fig. 8. Then, we propose a heuristic algorithm to optimize the texture image (d). Final, a stylized image is obtained by texture transfer (e).



Input Image
Extract Feature (Sec. III -A)
Generate Discrete Elements (Sec. III -B)
Initialize Texture Image (Sec. III -C)
Optimize Texture Image (Sec. III -D)
Texture Transfer (Sec. III -E)
Output Image

FIGURE 3. Texture synthesis processing chain.

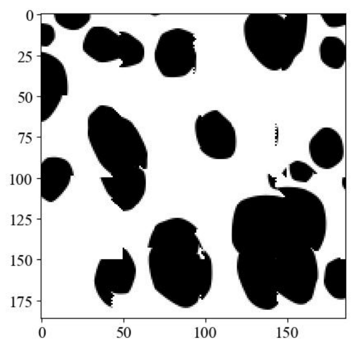


FIGURE 4. Patch-based method [23] for texture synthesis based on Fig. 3(a).

A. EXTRACT FEATURE FROM INPUT EXEMPLARS

Given an exemplar $I(x, y)$, such as Fig. 3(a), if we directly use some patch-based algorithms (such as [20]) to synthesize texture, the final texture image may present some image seams, as shown in Fig. 4, and the texturing process is time-consuming. To preserve the integrity of elements in $I(x, y)$ and speed up the synthesis process, we need to extract texture elements from $I(x, y)$ discretely.

First, we obtain the blurred image $g(x, y)$ from $I(x, y)$ by Gaussian filter [33] to reduce pattern noise:

$$g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} * I(x, y) \tag{1}$$

where $*$ means the convolution operation and σ^2 is the variance.

Next, the gradient value $G(x, y)$ and the gradient direction $\theta(x, y)$ of $g(x, y)$ can be obtained by Sobel operator [8]:

$$G(x, y) = \sqrt{g_x(x, y)^2 + g_y(x, y)^2} \tag{2}$$

$$\theta(x, y) = \arctan\left(\frac{g_y(x, y)}{g_x(x, y)}\right) \tag{3}$$

where $g_x = \frac{\partial g(x,y)}{\partial x}$ is the horizontal gradient component and $g_y = \frac{\partial g(x,y)}{\partial y}$ is the vertical gradient component.

Here, we remove some pixels that are near an element edge by threshold T_S to reduce the edge width:

$$G(x, y) = \begin{cases} G(x, y) & \text{if } G(x, y) > T_S \\ 0 & \text{if } G(x, y) \leq T_S \end{cases} \tag{4}$$

Then, two thresholds, T_L and T_H, are used to recognize the feature points based on the result of (4):

$$\begin{cases} G(x, y) = 255 & \text{if } G(x, y) > T_H \\ G(x, y) = t & \text{if } T_L \leq G(x, y) \leq T_H \\ G(x, y) = 0 & \text{if } G(x, y) < T_L \end{cases} \tag{5}$$

where $t \in \{0, 255\}$. $G(x, y) = 255$ means that (x, y) is a feature point. $G(x, y) = t$ means that (x, y) is an uncertain point. If an uncertain point is adjacent to known feature points, its value is equal to 255; otherwise, its value is equal to 0. $G(x, y) = 0$ means that (x, y) is a non-feature point.

Final, we can extract a pixel coordinate set of feature points $P = \{(x_i, y_i)\}, i = 1, \ldots, p$ based on gradient information by a border following algorithm, where p is the number of feature points, as shown in Fig. 5. Moreover, if necessary, users can add feature points to the set P manually to improve the accuracy of edge extraction when they are facing with some complex patterns, such as flower patterns.

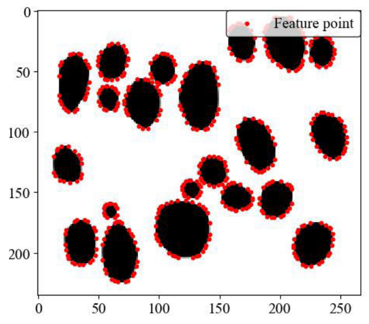


FIGURE 5. Extract feature points.

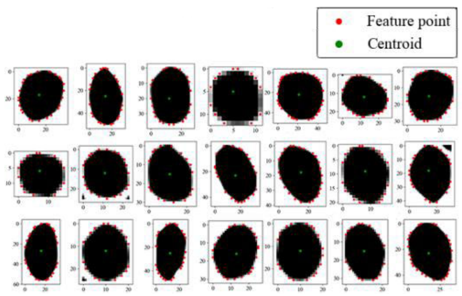


FIGURE 6. Discrete example-based elements.

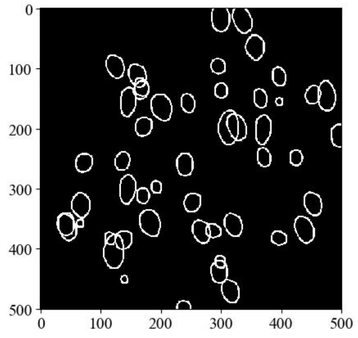


FIGURE 7. Initial texture image in a stochastic way.

B. GENERATE DISCRETE EXAMPLE-BASED ELEMENTS

Based on the result of set P, we need to cluster these feature points (see Fig. 5) to determine the approximate number of elements. Therefore, some clustering algorithms, such as K-means clustering or mean shift clustering [34], can be used to determine the number of centroids (the pixel coordinates need to be integerized) in an exemplar that are equal to the number of discrete elements. If users want to use cluster algorithms, such as K-means clustering, the number of centroids should be determined in advance. Here, we adopt the mean shift algorithm to cluster feature points because such algorithm can search centroids based on the distribution of feature points automatically, and the main steps are listed as follows:

Step1: Select one random point as a center point c_i from the unclassified points (feature points).

Step2: Add all points around c_i within r_i to the set M_i, where r_i is the radius.

Step3: Compute each vector $\left(c_i, m_j\right)$, where $m_j \in M_i$, and add all $\left(c_i, m_j\right)$ together to obtain $\vec{s}$.

Step4: Let c_i move along the direction of $\vec{s}$, and the moving distance is $\|\vec{s}\|$.

Step5: Repeat **Step2-Step4** until $\|\vec{s}\| \leq \varepsilon$, where ε is the threshold to stop the loop, and record the current position of c_i.

Step6: Repeat **Step1-Step5** until all points are classified.

Step7: Compute an access frequency f_{c_i} for each point based on different center points c_i, and let the current point belong to c_i by $\arg\max\left(\{f_{c_i}\}\right)$.

Then, we divide the set P into single discrete elements $\{p_1, \ldots, p_j\}$, as shown in Fig. 6, where j is the number of centroids. Each p_j includes its pixel coordinate, its relative angle and distance between the centroid j and corresponding feature points, and the RGB intensities, which can speed up the process for texture synthesis in following subsections.

C. INITIALIZE TEXTURE IMAGES

There are many methods that can be used to initialize texture images, such as patch-based methods. Here, we adopt a stochastic pattern for initialization, as shown in Fig. 7. However, a regular pattern can also be used to initialize texture images, as shown in Fig. 8, and we will present the final texture images based on both two initialization methods.

In this paper, the Poisson cluster process (PCP) is chosen to model the stochastic position distribution of elements that are randomly selected from $\{p_1, \ldots, p_j\}$, and we always copy selected elements into a output image completely [35]:

$$\Phi = \bigcup_{i \in n} \Phi_i + x_i \tag{6}$$

where Φ is the parent point process, n is the number of the parent point process, and Φ_i, $i \in n$ is a finite set of the child point process. The modeling process is listed as follows and shown in Algorithm 1:

Step1: Generate $n \sim \text{Poisson}\left(\pi r^2 \lambda\right)$, where λ is the density function, r is the radius, and $u_1 \sim \text{U}(0, 1), \ldots, u_n \sim \text{U}(0, 1)$.

Step2: Let $R_1 = r\sqrt{u_1}, \ldots, R_n = r\sqrt{u_n}$ and sort $R_1, \ldots, R_n$ to get $R_{(1)}, \ldots, R_{(n)}$.

Step3: Generate $u_{n+1} \sim \text{U}(0, 1), \ldots, u_{n+m} \sim \text{U}(0, 1)$ discretely.

Step4: Let $\theta_1 = 2\pi u_{n+1}, \ldots, \theta_m = 2\pi u_{n+m}$, and generate initial pixel coordinate pairs $\left(R_{(n)}, \theta_1\right), \ldots, \left(R_{(n)}, \theta_m\right)$.

In addition, to speed up the initialization process, we align the centroid of selected element with the node generated by PCP or the regular pattern and reconstruct the element based on the relative angle and distance and the RGB intensities restored in p_j.

D. OPTIMIZE TEXTURE BASED ON HEURISTIC SEARCHING

From Fig. 7 or Fig. 8, we find that some elements overlap over others, which cannot preserve the integrity of all elements.

Algorithm 1 Algorithm for Texture Initialization

```
1:  Input: Size of Texture Image, Distribution Density $\lambda$,
    Number of Child Process Points $n$, and Distribution
    Radius $r$
2:  Output: Initial Texture Image
3:  Initialize: Number of iteration $m \leftarrow \lambda$
4:  for $i = 1, 2, \ldots, m$ do
5:      Generate $u_i \sim \mathrm{U}(0, 1)$ and $R_i = r\sqrt{u_i}$
6:      for $j = 1, 2, \ldots, n$ do
7:          Generate $u_{i,j} \sim \mathrm{U}(0, 1)$
8:          Compute $\theta_{i,j} = 2\pi u_{i,j}$ and generate $(R_i, \theta_{i,j})$
9:      end for
10: end for
11: return texture image
```

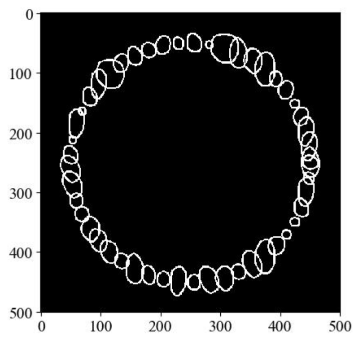


FIGURE 8. Initial texture image in a regular way.

To improve the quality of a texture image, we propose a heuristic algorithm to optimize a texture image in this subsection.

First, we formulate our optimization problem via the following energy function:

$$E = \sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{k=1,k\neq j}^{N} O\left(S_j, S_k\right) \tag{7}$$

where M is the number of parent nodes, N is the number of child nodes in each parent cluster, $O\left(S_j, S_k\right)$ is the average overlap rate (AOR) between element j and element k, and the expression of $O\left(S_j, S_k\right)$ can be expressed as:

$$O\left(S_j, S_k\right) = \frac{S_j \cap S_k}{S_j \cup S_k} \tag{8}$$

where S_j and S_k are the relative area of element j and element k.

Second, we define the relative area S_j or S_k in (8) as the bounding box of element j or element k. Here, Graham algorithm is used to determine such bounding box, and its program interface can be found in CV2, as shown in Fig. 9.

Then, we can define our optimization problem as:

$$\min E = \sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{k=1,k\neq j}^{N} O\left(S_j, S_k\right) \tag{9}$$

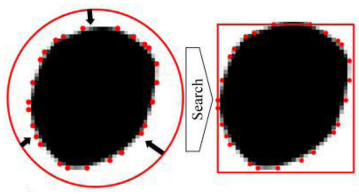


FIGURE 9. The bounding box of single element.

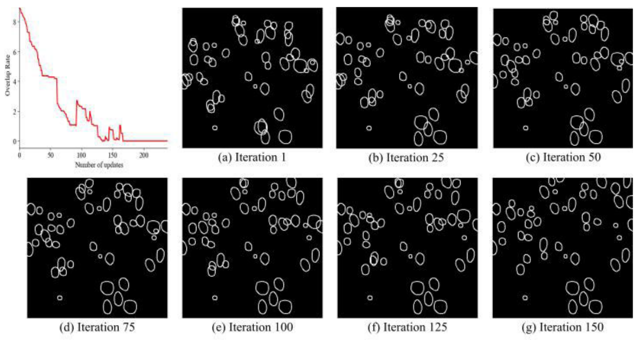


FIGURE 10. Iteration process.

Final, it is hard to solve (9) directly because its gradient cannot be obtained. Therefore, we propose a heuristic algorithm to optimize (9) based on Coordinate Descent Optimization (CDO) that mainly consists of five steps, as shown in Algorithm 2, with a time complexity of $O(MNN)$.

Step1: Set each centroid j as an origin, divide the searching area for S_j into four quadrants, and set initial parameters for the searching radius, the searching length in pixel coordinate frame and the AOR threshold to stop the searching process (set by users).

Step2: Before each iteration, we need to determine whether element j and element k are overlapped based on (8). If the value of (8) is greater than the AOR threshold, we will add the number pair (j, k) to the set $\{(j, k)\}$.

Step3: Before each searching process, we select (j, k) from the set $\{(j, k)\}$, then fix the element j position and choose one random direction to begin our searching process. Here, we adopt three transform methods to optimize the global texture distribution, including translation, scale and rotation. For example, we first fix the value of y_k and search the possible value $x_{k'}$ for the position $(x_{k'}, y_k)$. Then, based on the new position of x_k, we will search the possible value for y_k.

Step4: After the searching process, we add the element k' position to a set $\{(x_{k'}, y_{k'})\}$, where $(x_{k'}, y_{k'})$ is the possible position for element k, and determine the optimal position for element k by finding $\left(x_k^*, y_k^*\right)$ that can minimize (8) in the set $\{(x_{k'}, y_{k'})\}$. If so, (j, k) is removed from the set $\{(j, k)\}$.

Step5: Repeat **Step2-Step4** until the set $\{(j, k)\}$ is empty or the number of loop reaches its threshold.

By Algorithm 2, we can obtain optimized texture images, as shown in Fig. 10.

Algorithm 2 Algorithm for Texture Optimization

```
1:  **Input:** Initial Texture Image, Number of Parent Process Points $m$, Number of Child Process Points $n$
2:  **Output:** Optimized Texture Image
3:  **Initialize:** Maximum number of iteration $T \leftarrow \kappa$, accuracy threshold $\leftarrow \varepsilon$, AOR threshold $\leftarrow \sigma$
4:  Compute initial $E$ from (7) - (8)
5:  **while** $T \neq \kappa$ or $E \geq \varepsilon$ **do**
6:      **for** $i = 1, 2, \ldots, m$ **do**
7:          **for** $j = 1, 2, \ldots, n$ **do**
8:              **for** $k = 1, 2, \ldots, n$ and $j \neq k$ **do**
9:                  Compute $O\left(S_j, S_k\right)$ from (8)
10:                 **if** $O\left(S_j, S_k\right) \leq \sigma$ **do**
11:                     Add $(j, k)$ to the set $\{(j, k)\}$
12:                 **end if**
13:             **end for**
14:         **end for**
15:     **end for**
16: **while** $\{(j, k)\} \neq \varnothing$ **do**
17:         Select $(j, k)$, compute $(x_{k'}, y_{k'})$ based on heuristic searching, and add $(x_{k'}, y_{k'})$ to a set $\{(x_{k'}, y_{k'})\}$
18:         Let $\left(x_k^*, y_k^*\right) \leftarrow \arg\min(\{(x_{k'}, y_{k'})\})$, update the element $k$ position and remove $(j, k)$ from $\{(j, k)\}$
19: **end while**
20:     $T+=1$, and compute current $E$
21: **end while**
22: return texture image
```

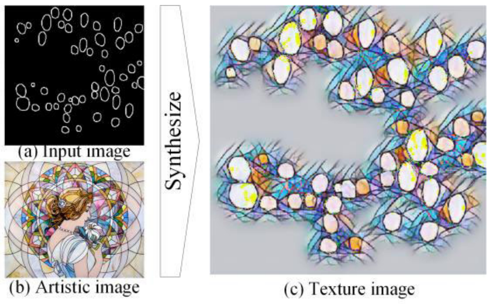


FIGURE 11. Stylization process.

TABLE 1. Simulation parameters of texture synthesis.

Parameter	Description	Value
M	Number of parent point process	1
N	Number of child point process	50
r	Distribution radius	250
T	Number of maximum iteration	100
ε	Accuracy threshold	0.01
σ	AOR threshold	0.01

E. TEXTURE TRANSFER

In this subsection, our aim is to stylize optimized texture images, and we adopt a neural network method to stylize texture images. Such method can provide the feature space

TABLE 2. System parameters of texture synthesis.

Parameter	Type
Processor	Intel i7-5500U
Memory	16 G
OS	Window

TABLE 3. Results of texturing time by different methods.

Pattern	Our method	Pixel-based method [18]	Patch-based method [20]
Circle pattern	10 s	2 h and 10 min	1 h
Wave pattern	9 s	1 h and 30 min	50 min
Leopard print	15 s	2 h and 30 min	1 h and 40 min

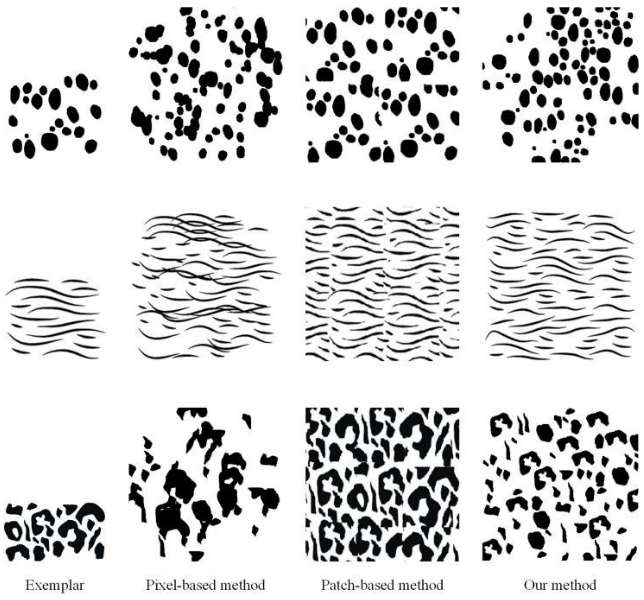


FIGURE 12. The experiment results of texture synthesis. For each discrete texture (left), the output images based on the pixel-based method in [18] and the patch-based method in [20] are on the middle, and the output image based on our method is on the right.

produced by CNN layers that are very useful for feature representation.

Given an input image $A(x, y)$ (Fig. 10(g)) and an artistic image $B(x, y)$ (Fig. 11(b)), we use 16 convolutional layers and 5 average pooling layers from VGG-Network [36] to extract feature representations from $A(x, y)$ and $B(x, y)$ in different layers and define the squared-error loss between different feature representations in layer l as:

$$\ell(A, B, l) = \frac{1}{2} \sum_{i=1}^{N_l} \sum_{j=1}^{M_l} \left(F_{i,j}^l - P_{i,j}^l\right)^2 \tag{10}$$

where N_l is the number of filters in layer l, M_l is the size of feature map, $F_{i,j}^l$ is the content loss produced by the filter i at position j in $A(x, y)$, and $P_{i,j}^l$ is the style loss produced by the filter i at position j in $B(x, y)$.

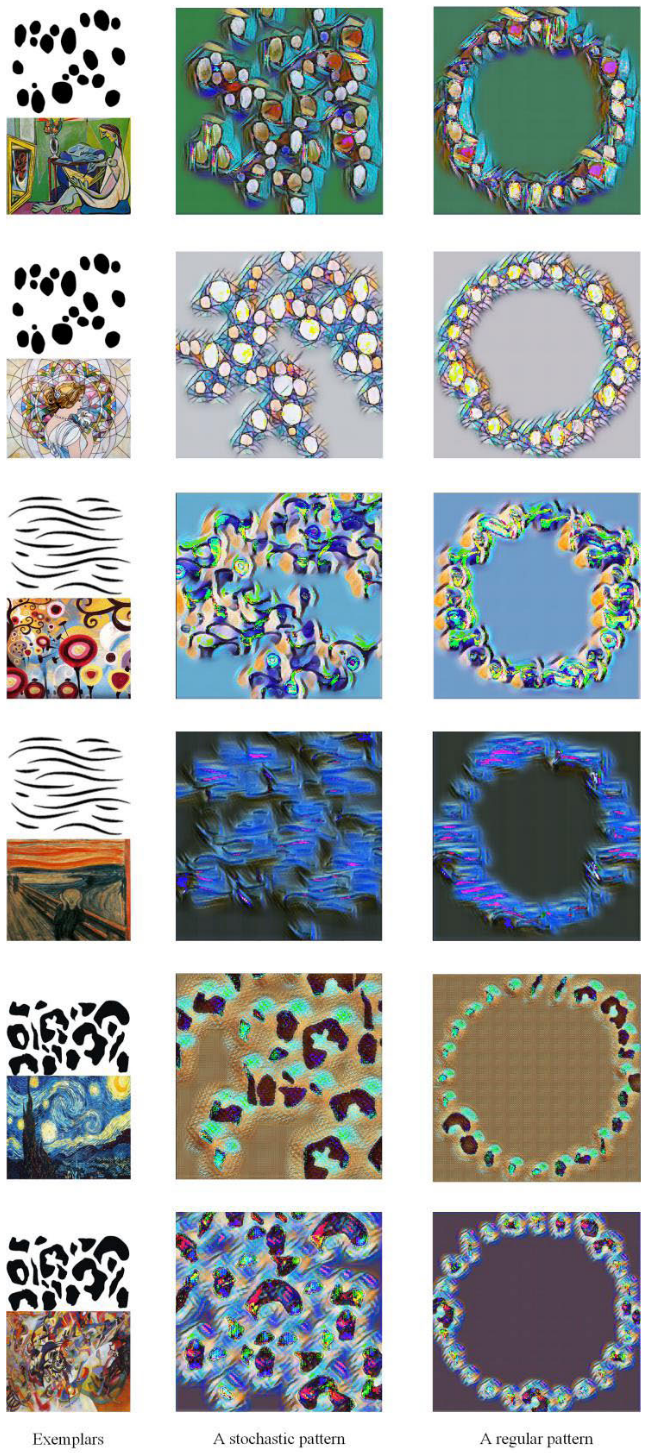


FIGURE 13. **The experiment results of texture stylization. The input images are on the left, including an exemplar and an artistic image, the output image with a stochastic pattern is on the middle, and the output image with a regular pattern is on the right.**

The derivative of (10) can be expressed as [37]:

$$\frac{\partial \ell (A, B, l)}{\partial F^l_{i,j}} = \begin{cases} \left(F^l_{i,j} - P^l_{i,j}\right) & \text{if } F^l_{i,j} > 0 \\ 0 & \text{if } F^l_{i,j} \leq 0 \end{cases} \tag{11}$$

Therefore, a gradient descent algorithm (GDA) can be an effective method to optimize $A(x, y)$ until it has the same feature representation as $B(x, y)$, i.e., minimize (10) based on the convergence of (11), and the final texture image is shown in Fig. 11.

IV. SIMULATION RESULTS

For our simulations, the parent nodes are distributed in an area 500 × 500 (the size of output image), and the distribution of parent nodes satisfies PCP, where the distribution density λ is 1, the number of subprocess points n is 50, and the distribution radius r is 250 (usually half of the size of an output image). Table 1 lists the simulation parameters of texture synthesis and Table 2 lists the system parameters of texture synthesis.

All results in this section are independent experiments through Python (including NUMPY, CV2, SKLEARN and KERAS). The results of texture synthesis are presented in Fig. 12 and the corresponding results of texture stylization are presented in Fig. 13. Moreover, the results of time for texture synthesis are summarized in Table 3. As illustrated in Table 3, our proposed method can save nearly 1 hour and 2 hours compared with the pixel-based method in [18] and the patch-based method in [20], respectively. Here, the main steps in [18] are summarized as follows, with a time complexity that is proportional to the size of a sample image and an initial noise image:

Step1: Input a sample image I_s and generate a random noise image I_n.

Step2: Set the searching radius $r = 2$ and the searching window

$$w = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

Step3: Select a patch in I_s that is similar to the local region in I_n based on the convolution operation.

Step4: Repeat **Step3** until all pixels in I_n are updated.

The main steps in [20], with a time complexity that is proportional to the size of an output image, a block and an overlap region, include:

Step1: Input a sample image I_s, initialize an image I_0 and set the size for an overlap region S and a block B.

Step2: Copy one block B_1 from I_s into I_0 randomly to begin.

Step3: Compute the error $e = (B_{current} - B_{next})^2$ at the overlap region S_e to minimize $E_{i,j} = e + \min\left(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1}\right)$, where i and j are the size of the overlap region S_e and E is the cumulative error.

Step4: Repeat **Step3** until I_0 are filled with blocks.

Compared with our method, the methods in [18] and [20] are more time-consuming obviously.

V. CONCLUSION

With a focus on fast texture synthesis for discrete texture, this paper proposes a novel framework for texturing that mainly consists of five steps. We first extract texture feature from input exemplars and generate discrete elements based on the feature distribution and the mean shift algorithm. After initializing the texture image in a stochastic or regular way, we propose a texture optimization algorithm base on heuristic searching to improve the quality of optimized texture images. Final, a CNN-based method is used to stylize our texture images. The results indicate that our method can effectively optimize and speed up the process of discrete texture synthesis.

In the future, we will explore methods for extracting more complex patterns, like some feature matching methods, and take into account factors that may improve the extraction accuracy, such as illumination variation, viewpoint angles, color constancy, and contrast balancing. Moreover, it is important to design more robust optimization algorithms and apply our texturing method to other scenarios, such as remote sensing images fusion and artistic or industrial design.

REFERENCES

[1] Y. Gui and Y. Liu, "Discrete texture elements synthesis on surfaces using elements distribution," *Int. J. Comput. Vis. Robot.*, vol. 10, no. 1, p. 61, 2020.

[2] L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk, "State of the art in example-based texture synthesis," in *Proc. Eurographics*, Munich, Germany, 2009, pp. 93–117.

[3] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra, "Texture optimization for example-based synthesis," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 795–802, Jul. 2005.

[4] Y. Peng and H. Yin, "Markov random field based convolutional neural networks for image classification," in *Proc. Int. Conf. Intell. Data Eng. Automated Learn.* Cham, Switzerland: Springer, 2017.

[5] P. Mohanaiah, P. Sathyanarayana, and L. Gurukumar, "Image texture feature extraction using GLCM approach," *Int. J. Sci. Res. Publications*, vol. 3, no. 5, pp. 1–5, 2013.

[6] R. Mohebian, M. A. Riahi, and O. Yousefi, "Detection of channel by seismic texture analysis using grey level co-occurrence matrix based attributes," *J. Geophys. Eng.*, vol. 15, no. 5, pp. 1953–1962, Oct. 2018.

[7] M. Nouri, B. Nasehi, M. Goudarzi, and S. A. Mehdizadeh, "Non-destructive evaluation of bread staling using gray level co-occurrence matrices," *Food Anal. Methods*, vol. 11, no. 12, pp. 3391–3395, Dec. 2018.

[8] M. Sharma and H. Ghosh, "Histogram of gradient magnitudes: A rotation invariant texture-descriptor," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2015, pp. 4614–4618.

[9] B. Li, K. Cheng, and Z. Yu, "Histogram of oriented gradient based gist feature for building recognition," *Comput. Intell. Neurosci.*, vol. 2016, Oct. 2016, Art. no. Histogram of oriented gradient based gist feature for building recognition.

[10] R. Maani, S. Kalra, and Y.-H. Yang, "Noise robust rotation invariant features for texture classification," *Pattern Recognit.*, vol. 46, no. 8, pp. 2103–2116, Aug. 2013.

[11] J. Zhang, J. Liang, C. Zhang, and H. Zhao, "Scale invariant texture representation based on frequency decomposition and gradient orientation," *Pattern Recognit. Lett.*, vol. 51, pp. 57–62, Jan. 2015.

[12] C. Li and Y. Huang, "Deep decomposition of circularly symmetric Gabor wavelet for rotation-invariant texture image classification," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 2702–2706.

[13] J. J. de Mesquita Sá Junior and A. Ricardo Backes, "A simplified gravitational model to analyze texture roughness," *Pattern Recognit.*, vol. 45, no. 2, pp. 732–741, Feb. 2012.

[14] D. Vaishali, R. Ramesh, and J. Anita Christaline, "2 d autoregressive model for texture analysis and synthesis," in *Proc. Int. Conf. Commun. Signal Process.*, Apr. 2014, pp. 1135–1139.

[15] L. F. Scabini, W. N. Gonçalves, and A. A. Castro, "Texture analysis by bag-of-visual-words of complex networks," in *Proc. Iberoamerican Congr. Pattern Recognit.* Cham, Switzerland: Springer, 2015.

[16] S. Thewsuwan and K. Horio, "Texture classification by local spatial pattern mapping based on complex network model," *Int. J. Innov. Comput. Inf. Control*, vol. 14, no. 3, pp. 1113–1121, 2018.

[17] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, vol. 2, Sep. 1999, pp. 1033–1038.

[18] L.-Y. Wei and M. Levoy, "Fast texture synthesis using tree-structured vector quantization," in *Proc. 27th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, 2000, pp. 479–488.

[19] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in *Proc. 28th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, 2001, pp. 341–346.

[20] S. Avidan and A. Shamir, "Seam carving for content-aware image resizing," *ACM Trans. Graph.*, vol. 26, no. 3, p. 10, Jul. 2007.

[21] E. Praun, A. Finkelstein, and H. Hoppe, "Lapped textures," in *Proc. 27th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, 2000, pp. 465–470.

[22] J.-M. Dischler, K. Maritaud, B. Lévy, and D. Ghazanfarpour, "Texture particles," *Comput. Graph. Forum*, vol. 21, no. 3, pp. 401–410, Sep. 2002.

[23] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: Image and video synthesis using graph cuts," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 277–286, 2003.

[24] L.-Y. Wei and M. Levoy, "Order-independent texture synthesis," 2014, *arXiv:1406.7338*. [Online]. Available: http://arxiv.org/abs/1406.7338

[25] C. Ma, L.-Y. Wei, and X. Tong, "Discrete element textures," *ACM Trans. Graph.*, vol. 30, no. 4, p. 62, Jul. 2011.

[26] P. Merrell, "Example-based model synthesis," in *Proc. Symp. Interact. 3D Graph. Games*. Seattle, WA, USA: Association for Computing Machinery, 2007, pp. 105–112.

[27] D. Dai, H. Riemenschneider, G. Schmitt, and L. Van, "Example-based facade texture synthesis," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2013, pp. 1065–1072.

[28] X. Cai, B. Song, and Z. Fang, "Exemplar based regular texture synthesis using LSTM," *Pattern Recognit. Lett.*, vol. 128, pp. 226–230, Dec. 2019.

[29] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Diversified texture synthesis with feed-forward networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3920–3928.

[30] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6924–6932.

[31] Z.-M. Wang, M.-H. Li, and G.-S. Xia, "Conditional generative ConvNets for exemplar-based texture synthesis," 2019, *arXiv:1912.07971*. [Online]. Available: http://arxiv.org/abs/1912.07971

[32] T. Sethapakdi and J. McCann, "Painting with CATS: Camera-aided texture synthesis," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2019, pp. 1–9.

[33] A. Humeau-Heurtier, "Texture feature extraction methods: A survey," *IEEE Access*, vol. 7, pp. 8975–9000, 2019.

[34] K.-L. Wu and M.-S. Yang, "Mean shift-based clustering," *Pattern Recognit.*, vol. 40, no. 11, pp. 3035–3052, Nov. 2007.

[35] S. N. Chiu, D. Stoyan, W. S. Kendall, and J. Mecke, *Stochastic Geometry and Its Applications*. Hoboken, NJ, USA: Wiley, 2013.

[36] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," 2015, *arXiv:1508.06576*. [Online]. Available: http://arxiv.org/abs/1508.06576

[37] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," *Int. J. Comput. Vis.*, vol. 128, no. 2, pp. 336–359, Feb. 2020.

ZHENGRUI HUANG was born in Fujian, China, in 1994. He received the B.S. degree from the College of Environment and Resource, Fuzhou University, Fuzhou, China, in 2017, where he is currently pursuing the M.S. degree with the Academy of Digital China, Fujian. His major research interests include the Internet of Things and spatial information integration technology, wireless communication and UAV-aided communication, UAV path planning, multiobjective optimization, machine learning, and computer vision.

XIAOHONG LIN received the B.S. degree from the School of Design, Hunan University, Hunan, China, in 1989. She is currently working with the Fuzhou Technology and Business College, Fuzhou, China. Her major research interests include industrial design, product design and 3D modeling, machine learning, and computer vision.

CHONGCHENG CHEN (Member, IEEE) received the M.A. degree in hydro and engineering geology from the Chengdu University of Technology, Chengdu, China, in 1992, and the Ph.D. degree in cartography and GIS from the Institute of Geographical Science and Resource, Chinese Academy of Science, Beijing, China, in 2000. He initiated the geographical information system development and application at Fuzhou University. He was the Co-Founder of the Spatial Information Research Center, Fuzhou, China, in 2001. He is currently dedicated to bridging the gap between innovative IT and earth science research under the umbrella of Digital Earth, and promote geospatial technology to inoculate with applications in broad fields of nature resource, environment, ecology, and government management since then.