A SAMPLING-BASED GEM ALGORITHM WITH CLASSIFICATION FOR TEXTURE SYNTHESIS

Liu-yuan Lai, Wen-Liang Hwang

Academia Sinica
Institute of Information Science

Silong Peng

The Chinese Academy of Sciences
Institute of Automation

ABSTRACT

Research on texture synthesis has made substantial progress in recent years, and many patch-based sampling algorithms now produce quality results in an acceptable computation time. However, when such algorithms are applied, whether they provide good results for specific textures, and why they do so, are questions that have yet to be answered. In this article, we deal specifically with the second question by modeling the synthesis problem as one of learning from incomplete data, and propose an algorithm that is a generalization of patch-work approach. Through this algorithm, we demonstrate that the solution of patch-based sampling approaches is an approximation of finding the maximum-likelihood optimum by the generalized expectation and maximization (GEM) algorithm.

1. INTRODUCTION

In recent years, research into synthesis by sampling, such as that in [1, 2, 3], has lead to the development of algorithms for texture synthesis that produce good quality results rapidly [4, 5, 6]. Among these methods, the work presented in [5] (referred to as patch-work hereafter) is of special interest to us. Despite its simplicity, the algorithm produces good quality results. However, two questions crucial to understanding of the texture synthesis problem remain unanswered. First, for what kind of textures does the patch-work algorithm yield perceptually acceptable results? Second, what is the solution quality of the patch-work approach? The answer to the first question requires a more concise texture model, which - to our knowledge - has not been developed yet. The goal of this article is to answer the second question. We try to explain the results of patch-work algorithm as the outcome of an optimization process by presenting the task of texture synthesis as a problem of obtaining maximum-likelihood estimates from incomplete data. The input texture image is the incomplete data that we observe, while the texture image to be synthesized is part of the unobserved data. Our proposed algorithm performs texture synthesis by solving the estimation problem with the GEM algorithm. We show that the proposed algorithm is a generalization of the patch-work approach. Furthermore, by analyzing our algorithm, we can explain the solution quality of patch-work algorithm.

The remainder of the paper is organized as follows. In Section 2, we propose our algorithm. Section 3 presents texture synthesis as a process of computing maximum likelihood estimates from incomplete data. We also show that our algorithm is in fact a GEM method. The results of running our algorithm are given in Section 4. We then present our conclusions in Section 5.

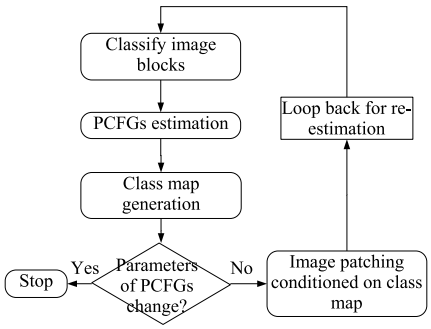

Fig. 1. Flowchart of our algorithm

2. SAMPLING-BASED ALGORITHM WITH CLASSIFICATION

Before describing each step in detail, we explain some key points about the design of our approach. The proposed algorithm introduces a new variable to the problem, namely, the classification of an image's sub-blocks, which allows us to take an abstract view of the composition of a texture's structure. This helps us analyze and extract the global structure of textures, instead of only local correlations in a small neighborhood. We use probability context free grammar (PCFG) to capture the intra-relations within each class and the inter-relations between classes. An overall flowchart of our algorithm is shown in Figure 1.

Step 1 – Classification

The first step of the algorithm classifies the constituent elements of an image. Here, we consider that textures comprise two classes: foreground and background. For textures with global structures, the placements of the foreground and background elements provide important information. We use a square block of fixed size as the unit of elements. An image is thus divided into fixed-sized blocks and represented by the mean of its pixel values. These blocks are then classified as either foreground or background, denoted as class 0 and 1, respectively. The classification method used here is the k-means algorithm running on RGB channels, with k equal to 2. A k-means classifier is trained with all possible blocks in the texture images as training data. The trained classifier is then used to classify the fixed image blocks that have been divided into texture images. Figure 2 shows the classification results (class map) of a sample texture image using the classifier thus trained.

1	0	1	0	1	1	0	0
0	0	0	0	1	1	0	0
0	1	1	0	0	0	1	1
0	1	1	0	0	0	1	1
0	1	1	0	0	0	1	1
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
1	0	1	1	0	0	0	1

Fig. 2. Classification results of a sample texture image (128x128) using 16x16 block size.

Step 2 – PCFG estimation

We now have a class map C of a texture image I. C can be seen as a two-dimensional code. Scanning the code row by row, we treat C as one-dimensional code sequences generated by some PCFG. The same technique is applied to C column-wise. We denote the languages of the two PCFGs as L_H and L_V, respectively; and define the grammar for the languages as follows:

$$S \longrightarrow P|Q$$
$$P \longrightarrow AQ|A$$
$$Q \longrightarrow BP|B$$
$$A \longrightarrow A^1|A^2|\ldots|A^n, A^i = \overbrace{0\cdots0}^{i \text{ times}},\ 1 \le i \le n$$
$$B \longrightarrow B^1|B^2|\ldots|B^n, B^j = \overbrace{1\cdots1}^{j \text{ times}},\ 1 \le j \le n$$

where n is the maximum size of the run-length code. In our algorithm, we set n equal to the number of blocks per column or row, whichever is larger.

Beginning with the start symbol S, the language forks into two branches: one ($S \rightarrow P$) generates code sequences that start with the terminal symbol 0, while the other ($S \rightarrow Q$) generates sequences starting with 1. Rules $P \rightarrow AQ$ and $Q \rightarrow BP$ define the alternations between sequences of 0s and sequences of 1s respectively. Rules $P \rightarrow A$ and $Q \rightarrow B$ stop the infinite alternation created by the previous two rules. In the following sections, we use R to denote rule probabilities. The probabilities of rule r are denoted by $P^h(r)$ for horizontal PCFG and $P^v(r)$ for vertical PCFG. Suitable subscripts will be added to specify which class map the probabilities apply to. The probabilities are calculated by counting the numbers of applications of the rules in the class map and dividing those numbers by the total number of substitutions applied.

Step 3 – Structure Generation

Based on the previously estimated probabilities R, we generate a class map for the texture image to be synthesized. We assume that if two images are realizations of the same texture, then their PCFGs should be close to each other. Let $\hat{R}$ be the probabilities of the PCFGs derived from class map $\hat{C}$ of a synthesized texture image. We use the function, $d(R\|\hat{R})$, as a measure of the discrepancy between PCFG C and PCFG $\hat{C}$:

$$d(R\|\hat{R}) = \sum_r P^h(r) \log \frac{P^h(r)}{\hat{P}^h(r)} + \sum_r P^v(r) \log \frac{P^v(r)}{\hat{P}^v(r)}. \quad (1)$$

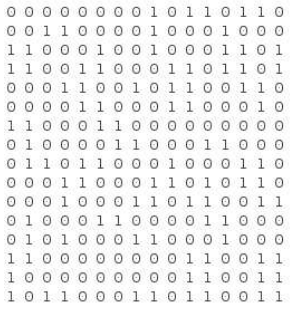

Fig. 3. The generated class map. The target image size is twice the input image in Fig. 2

The d thus specified is the sum of two Kullback-Liebler divergences. We want to find a class map $\hat{C}^*$ with probabilities $\hat{R}^*$ that gives the smallest $d(R\|\hat{R})$; however, as there is no closed form solution to this minimization, we take the following descending approach. Given an initial class map C, we modify it one block at a time so that the $d(R\|\hat{R})$ decreases with each iteration. At each iteration, we choose the block whose reassignment of 0 or 1 reduces d the most, and then make the modification. This step is repeated until $d(R\|\hat{R})$ can not be reduced any further. Following the example given in Figure 2, we show the generated class map in Figure 3.

Step 4 – Image patching

Having generated the texture structure using the class map, we begin to patch the image block by block from the upper-left corner to the bottom-right. To determine which patch should be pasted, we select the patch whose pixel values at the boundary regions best match those of the existing image in terms of the Euclidean distance. Unlike other methods, our approach classifies image patches. Therefore, instead of one large pool of candidates to search through, we have two smaller pools. For a target block to be pasted, we only search for candidates in the pool of the same class, which reduces search time. We use the graphcut method to avoid boundary artifacts when pasting the image patches. For details of the graphcut method, please refer to [7].

Step 5 – Iterate

After the previous four steps, we have an estimation of the parameters of the texture PCFGs and a synthesized texture image based on those parameters. The newly estimated parameters are then compared with those from the previous iteration. If the changes between the two sets of estimations are small, the algorithm terminates. Otherwise, it loops back to step one and starts another iteration.

Relation to patch-work

The main difference between our algorithm and the patch-work approach is that we separates image blocks into two classes. The lack of class differentiation in the patch-work method can be regarded as a special case, where all the image blocks belong to a single class. As a result, the generated class map is a constant and

the search space contains all possible image blocks. In this case, like the patch-work approach, our algorithm would perform only one iteration and stop.

3. GEM FOR TEXTURE SYNTHESIS

Let D be the data that includes the input texture I, and the synthesized texture $\hat{I}$; then, $P(D|\hat{R}) = P(I, \hat{I}|\hat{R})$. The objective of our texture synthesis algorithm is to maximize $\log P(I, \hat{I}|\hat{R})$ given the input texture I. The log probability is

$$\log P(I, \hat{I}|\hat{R}) = \log P(I, \hat{I}, \hat{C}^{t+1}|\hat{R}) - \log P(\hat{C}^{t+1}|I, \hat{I}, \hat{R}), \text{ for all } t.$$

Taking the expectation on both sides of the above equation over the probability distribution $P(\hat{C}^{t+1}|I, \hat{I}^t, \hat{R}^{*^t})$ and using the fact that

$$E\{\log P(I, \hat{I}|\hat{R})\} = \log P(I, \hat{I}|\hat{R}),$$

we have

$$\log P(I, \hat{I}|\hat{R}) = E\left\{\log P(I, \hat{I}, \hat{C}^{t+1}|\hat{R})\right\} - E\left\{\log P(\hat{C}^{t+1}|I, \hat{I}, \hat{R})\right\}. \quad (2)$$

The first and second term of the right-hand side of Equation (2) can be re-written as:

$$\begin{aligned} & E\{\log P(I, \hat{I}, \hat{C}^{t+1}|\hat{R})\}) \\ &= \sum_{\hat{C}^{t+1}} P(\hat{C}^{t+1}|I, \hat{I}^t, \hat{R}^{*^t}) \log P(I, \hat{I}, \hat{C}^{t+1}|\hat{R}) \\ &= Q(\hat{I}, \hat{R}|\hat{I}^t, \hat{R}^{*^t}) \quad (3) \\ & E\{\log P(\hat{C}^{t+1}|I, \hat{I}, \hat{R})\} \\ &= \sum_{\hat{C}^{t+1}} P(\hat{C}^{t+1}|I, \hat{I}^t, \hat{R}^{*^t}) \log P(\hat{C}^{t+1}|I, \hat{I}, \hat{R}) \\ &= H(\hat{I}, \hat{R}|\hat{I}^t, \hat{R}^{*^t}). \quad (4) \end{aligned}$$

Substituting Equations (3) and (4) into Equation (2), we have

$$\log P(I, \hat{I}|\hat{R}) = Q(\hat{I}, \hat{R}|\hat{I}^t, \hat{R}^{*^t}) - H(\hat{I}, \hat{R}|\hat{I}^t, \hat{R}^{*^t}). \quad (5)$$

Because $H(\hat{I}^t, \hat{R}^{*^t}|\hat{I}^t, \hat{R}^{*^t}) - H(\hat{I}^{t+1}, \hat{R}^{t+1}|\hat{I}^t, \hat{R}^{*^t})$ is the KL-divergence of $P(\hat{C}^{t+1}|I, \hat{I}^{t+1}, \hat{R}^{t+1})$ against $P(\hat{C}^{t+1}|I, \hat{I}^t, \hat{R}^{*^t})$, it is always the case that

$$H(\hat{I}^t, \hat{R}^{*^t}|I^t, \hat{R}^{*^t}) \geq H(\hat{I}^{t+1}, \hat{R}^{t+1}|\hat{I}^t, \hat{R}^{*^t}).$$

Consequently, if we have $\hat{I}^{t+1}$, $\hat{R}^{t+1}$ holds the property

$$Q(\hat{I}^{t+1}, \hat{R}^{t+1}|\hat{I}^t, \hat{R}^{*^t}) \geq Q(\hat{I}^t, \hat{R}^t|\hat{I}^t, \hat{R}^{*^t}),$$

we can conclude that, at each iteration,

$$\log P(I, \hat{I}^{t+1}|\hat{R}^{t+1}) \geq \log P(I, \hat{I}^t|\hat{R}^{*^t}).$$

To calculate the Q function, we define the distribution $P(\hat{C}^{t+1}|I, \hat{I}^t, \hat{R}^{*^t})$ to be proportional to the measure of the discrepancy given in Equation (1). We thus have

$$P(\hat{C}^{t+1}|I, \hat{I}^t, \hat{R}^{*^t}) \propto \exp^{-\frac{1}{\sigma^2} d(\hat{R}^{*^t} \| \hat{R}^{t+1})}, \quad (6)$$

where $\hat{R}^{t+1}$ represents the rule probablilities of the PCFGs of $\hat{C}^{t+1}$; and σ^2 is the parameter that determines the sharpness of the distribution over the optimal class map $\hat{C}^{*^{t+1}}$, whose PCFG probabilities, $\hat{R}^{*^{t+1}}$, give the minimum $d(\hat{R}^{*^t} \| \hat{R}^{*^{t+1}})$. If σ^2 is chosen to be a very small value, then Equation (6) can be approximated as

$$P(\hat{C}^{t+1}|I, \hat{I}^t, \hat{R}^{*^t}) \approx \begin{cases} 1 & \text{if } \hat{C}^{t+1} = \hat{C}^{*^{t+1}} \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

In this case, the Q function can be calculated as

$$\begin{aligned} Q(\hat{I}, \hat{R}|\hat{I}^t, \hat{R}^{*^t}) &\approx P(\hat{C}^{*^{t+1}}|I, \hat{I}^t, \hat{R}^{*^t}) \log P(I, \hat{I}, \hat{C}^{*^{t+1}}|\hat{R}) \\ &\approx \log P(I, \hat{I}, \hat{C}^{*^{t+1}}|\hat{R}). \quad (8) \end{aligned}$$

Let us set $\hat{I} = \hat{I}^t$ in $Q(\hat{I}, \hat{R}|\hat{I}^t, \hat{R}^{*^t})$ of Equation (8). We then find

$$\hat{R}^{*^{t+1}} = \arg\max_{\hat{R}} \log P(I, \hat{I}^t, \hat{C}^{*^{t+1}}|\hat{R}), \quad (9)$$

which gives us

$$\begin{aligned} Q(\hat{I}^t, \hat{R}^{*^{t+1}}|\hat{I}^t, \hat{R}^{*^t}) &= \log P(I, \hat{I}^t, \hat{C}^{*^{t+1}}|\hat{R}^{*^{t+1}}) \\ &\geq Q(\hat{I}^t, \hat{R}^{*^t}|\hat{I}^t, \hat{R}^{*^t}). \quad (10) \end{aligned}$$

Next, we set $\hat{R} = \hat{R}^{*^{t+1}}$ in $Q(\hat{I}, \hat{R}|\hat{I}^t, \hat{R}^{*^t})$ of Equation (8). We then find the next $\hat{I}^{t+1}$ is

$$\hat{I}^{t+1} = \arg\max_{\hat{I}} \log P(I, \hat{I}, \hat{C}^{*^{t+1}}|\hat{R}^{*^{t+1}}). \quad (11)$$

Having found $\hat{I}^{t+1}$ and $\hat{R}^{*^{t+1}}$, we have the property

$$\begin{aligned} Q(\hat{I}^{t+1}, \hat{R}^{*^{t+1}}|\hat{I}^t, \hat{R}^{*^t}) &\geq Q(\hat{I}^t, \hat{R}^{*^{t+1}}|\hat{I}^t, \hat{R}^{*^t}) \\ &\geq Q(\hat{I}^t, \hat{R}^{*^t}|\hat{I}^t, \hat{R}^{*^t}). \quad (12) \end{aligned}$$

Therefore, our objective log-likelihood function increases at each iteration as follows:

$$\log P(I, \hat{I}^{t+1}|\hat{R}^{*^{t+1}}) \geq \log P(I, \hat{I}^t|\hat{R}^{*^t}).$$

To obtain a synthesized image at each iteration, Equation (11) can be rewritten as

$$\begin{aligned} \hat{I}^{t+1} &= \arg\max_{\hat{I}} \log P(I, \hat{I}, \hat{C}^{*^{t+1}}|\hat{R}^{*^{t+1}}) \\ &= \arg\max_{\hat{I}} \log\left(P(I, \hat{I}|\hat{C}^{*^{t+1}}, \hat{R}^{*^{t+1}}) P(\hat{C}^{*^{t+1}}|\hat{R}^{*^{t+1}})\right). \quad (13) \end{aligned}$$

The probability $P(\hat{C}^{*^{t+1}}|\hat{R}^{*^{t+1}})$ does not depend on $\hat{I}$. Therefore, the maximization of Equation (13) can be simplified as

$$\hat{I}^{t+1} = \arg\max_{\hat{I}} \log\left(P(I, \hat{I}|\hat{C}^{*^{t+1}}, \hat{R}^{*^{t+1}})\right). \quad (14)$$

We deem $P(I, \hat{I}|\hat{C}^{*^{t+1}}, \hat{R}^{*^{t+1}})$ to be negatively proportional to a cost function of the total pasting errors induced by image pasting. Given a synthesized image $\hat{I}$, we denote the total pasting error as $\|\partial\hat{I}\|$ and the cost function as $K(\|\partial\hat{I}\|)$. We find $\hat{I}^{t+1}$ by

$$\hat{I}^{t+1} = \arg\min_{\hat{I}} K(\|\partial\hat{I}\|). \quad (15)$$

We introduce the sampling procedure used in sampling-based texture synthesizing algorithms, such as the patch-work approach, by defining the cost function K as

$$K(\|\partial\hat{I}\|) = \begin{cases} \|\partial\hat{I}^{t+1}\| & \text{if } |\|\partial\hat{I}\| - \|\partial\hat{I}^{t+1}\|| < \tilde{\epsilon}, \\ \|\partial\hat{I}\| & \text{otherwise.} \end{cases} \quad (16)$$

With the defined K function, we can introduce random sampling into our analysis. The cost of any synthesized image $\hat{I}$, whose total pasting error is $\tilde{\epsilon}$ distance within the image $\hat{I}^{t+1}$, is the same as that of $\hat{I}^{t+1}$. Therefore, the solution of Equation (15) is not unique for a sufficiently large $\tilde{\epsilon}$, which means we can randomly choose an image satisfying the equation as $\hat{I}^{+1}$.

3.1. Correspondence with our algorithm

We now explain how our algorithm relates to the above analysis. We initialize $\hat{I}^0$ to be an empty set $\emptyset$. $\hat{R}^{*^0}$ is obtained by applying the classification to I and estimating the PCFGs of the classified result. At iteration t, we start our calculation with I, $\hat{I}^t$ and $\hat{R}^{*^t}$. We classify the image blocks of $\hat{I}^t$ and acquire $\hat{C}^{t+1}$ in Step 1. Steps 2 and 3 then modify $\hat{C}^{t+1}$ according to the probabilities of the grammar rules, $\hat{R}^t$, to obtain $\hat{C}^{*^{t+1}}$ and $\hat{R}^{*^{t+1}}$, which corresponds to the estimation in Equation (7) and the M-step in Equation (9). At this stage, we check if the difference between $\hat{R}^{*^t}$ and $\hat{R}^{*^{t+1}}$ is small enough to terminate our algorithm. If it is not, the next step is to find $\hat{I}^{t+1}$ according to $\hat{C}^{*^{t+1}}$. Step 4 performs this M-step, as in Equation (15).

Remarks The class map $\hat{C}^{*^{t+1}}$ derived by Step 3 is a local minimum, because it is computationally inefficient to derive the global optimal solution. Also, pasting image blocks, as described in Step 4, to obtain $\hat{I}^{t+1}$ is a greedy approach, which does not necessarily yield the minimum total pasting error.

4. EXPERIMENT RESULTS

We show some results of our algorithm in Figure 4, along with the results of patch-work algorithm for comparison. All the input images are of size 128x128, and the output images are of size 256x256. The block size used in these demonstrations is 16x16, and the overlapping area at the boundary is 4 pixels wide. The same set of parameters is used for the patch-work algorithm, with the relative matching error $\epsilon = 0.2$. We use a sliding window moving in 4-pixel steps to sample an image block's data; thus one, 16x16 image block is sampled and its mean RGB values calculated every 4 pixels. The collected data is used to train the *2*-means classifier and paste the image.

5. CONCLUSION

In this article, we determine the kind of results derived by applying patch-work algorithms to texture synthesis. We view such synthesis as a parameter estimation problem with incomplete data, and design a GEM-based algorithm for the estimation task. Analysing the problem through the GEM algorithm, we show that the synthesis process of our algorithm is a search for the texture image that has the maximum likelihood of being observed. Because our algorithm is a generalization of the patch-work approach, we show that the outputs of the two approaches are approximations of the maximum-likelihood optimum of the GEM algorithm.

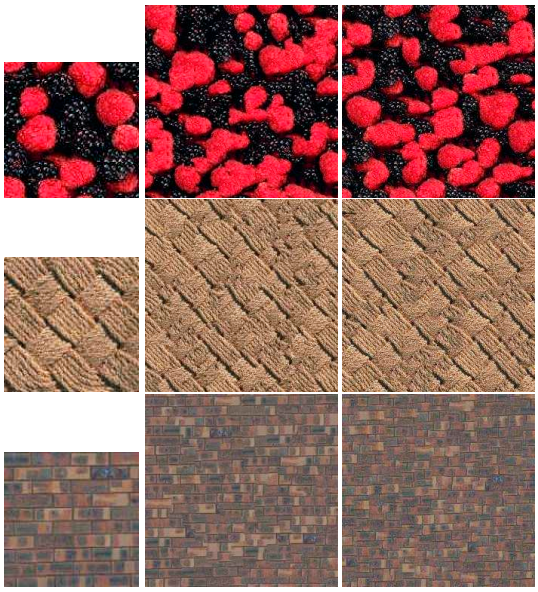

Fig. 4. Texture synthesis results. For each test image, the input image is on the left, the result of our algorithm is in the middle, and the patch-work result is on the right.

References

[1] A. Efros and T. Leung, "Texture synthesis by non-parametric sampling," in *International Conference on Computer Vision*, September 1999, vol. 2, pp. 1033–1038.

[2] Y. Wu S. Zhu and D. Mumford, "Filters, random fields and maximum entropy (frame) - towards a unified theory for texture modeling," *International Journal of Computer Vision*, vol. 27, no. 2, pp. 107–126, 1998.

[3] Michael Ashikhmin, "Synthesizing natural textures," in *The proceedings of 2001 ACM Symposium on Interactive 3D Graphics*, March 2001, pp. 217–226.

[4] Li-Yi Wei and Marc Levoy, "Fast texture synthesis using tree-structured vector quantization," in *SIGGRAPH*, 2000, pp. 479–488.

[5] Lin Liang, Ce Liu, Ying qing Xu, Baining Guo, and Heung yeung Shum, "Real-time texture synthesis by patch-based sampling," *ACM Transactions on Graphics*, vol. 20, no. 3, pp. 127–150, July 2001.

[6] Steve Zelinka and Michael Garland, "Towards real-time texture synthesis with the jump map," in *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, 2002, pp. 99–104.

[7] Vivek Kwatra, Arno Schdl, Irfan Essa, Greg Turk, and Aaron Bobick, "Graphcut textures: Image and video synthesis using graph cuts," *ACM Transactions on Graphics, SIGGRAPH 2003*, vol. 22, no. 3, pp. 277–286, July 2003.