Neural Comput & Applic (2009) 18:223–235
DOI 10.1007/s00521-008-0174-6

ORIGINAL ARTICLE

Robust tile-based texture synthesis using artificial immune system

Weiming Dong · Ning Zhou · Jean-Claude Paul

Received: 5 June 2007 / Accepted: 22 January 2008 / Published online: 16 April 2008

Abstract One significant problem in tile-based texture synthesis is the presence of conspicuous seams in the tiles. The reason is that sample patches employed as primary patterns of the tile set may not be well stitched if carelessly picked. In this paper, we introduce a robust approach that can stably generate an ω-tile set of high quality and pattern diversity. First, an extendable rule is introduced to increase the number of sample patches to vary the patterns in an ω-tile set. Second, in contrast to other concurrent techniques that randomly choose sample patches for tile construction, ours uses artificial immune system (AIS) to select the feasible patches from the input example. This operation ensures the quality of the whole tile set. Experimental results verify the high quality and efficiency of the proposed algorithm.

Keywords Texture synthesis · ω-tile · Sample patches selection · Clonal selection · Artificial immune system

W. Dong (✉)
LIAMA-NLPR, CAS Institute of Automation, Beijing, China
e-mail: wmdong@liama.ia.ac.cn; wmlake@gmail.com

W. Dong
Project ALICE, INRIA Lorraine/Loria, Nancy, France

N. Zhou
Department of Computer Science and Technology, Tsinghua University, Beijing, China
e-mail: zhoun03@mails.tsinghua.edu.cn

J.-C. Paul
INRIA/Tsinghua University, Beijing, China
e-mail: paul@tsinghua.edu.cn

1 Introduction

Generating novel photo-realistic imagery from smaller examples has been widely recognized as a significant problem in computer graphics. A wide number of applications require realistic textures to be synthesized for object decoration in virtual scenes. Texture refers to the class of imagery which is usually defined as an infinite pattern consisting of stochastically stationary repeating elements. The global repeatability within texture images is essential to texture synthesis techniques. This inherent property also makes it possible to express adequate texture information with limited portions.

Texture synthesis is an alternative way to create textures because synthetic textures can be made any size, visual repetition is avoided. Texture synthesis can also produce tileable images by properly handling the boundary conditions. The objective of texture synthesis can be stated as follows. Given an example texture (Fig. 1), synthesize a new texture that, when perceived by a human observer, appears to be generated by the same underlying process (Fig. 1).

Non real-time texture synthesis techniques can be roughly categorized into local region-growing methods and global optimization-based methods. Local methods generate the texture by growing one pixel or patch at a time with the constraint of maintaining coherence of neighboring pixels in the grown region [1–4]. Such approaches always suffer the time-consuming neighbor searching in the example, so they do not sufficiently meet real-time applications. On the other hand, global methods use some criteria to evaluate the similarity of the input, then the entire texture can be evolved as a whole. Most existing global approaches either model only pixel-to-pixel interactions which are insufficient to capture large-scale structures of the sample texture [5, 6], or introduce too

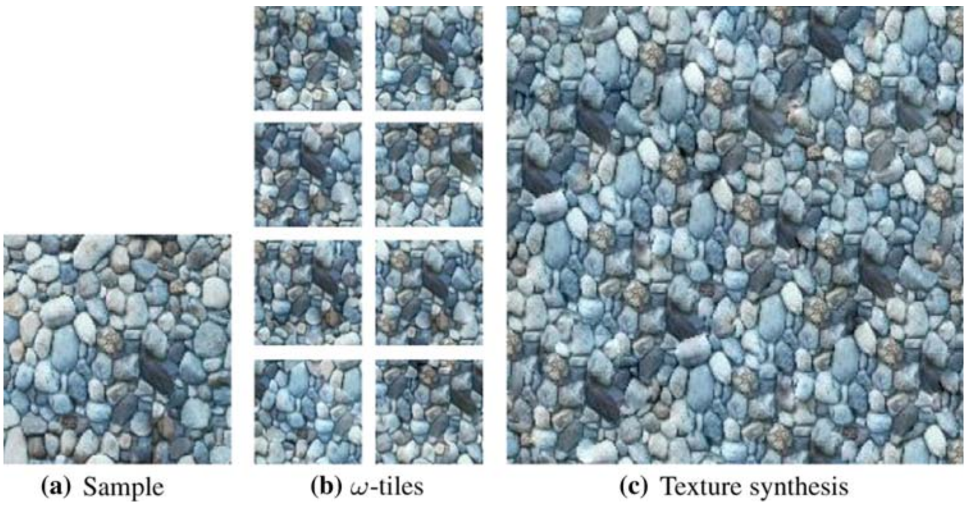


Fig. 1 Texture synthesis using our algorithm. The size of the input example in (**a**) is 128 × 128. We construct the ω-tiles (same sizes of 80 × 80) in (**b**) with our robust tile construction algorithm. The output texture in (**c**) has 256 × 256 pixels and is generated in real time using the ω-tiles in (**b**)

complex formulations to optimize [7, 8]. Kwatra et al. [9] defined a texture energy function to quantitatively measure the quality of the synthesized texture, unfortunately the synthesizing speed is still quite slow.

An alternative approach is to use texture synthesis to pre-compute a set of small tiles (textures) and use these tiles to generate arbitrary size of non-periodic images at run time [10, 11–14]. The tile-based method usually employs a set of *sample patches* which are extracted from the input example as texturing primitive. Tiles are then constructed by stitching sample patches together following some given rules. The technique requires only a small amount of memory and is very useful in many real-time applications, although sometimes achieving low-quality results or dull patterns for the lack of optimization on the tile set.

In this paper, we present an approach for tile-based image synthesis, based on the optimization of tile set quality with respect to a *clonal selection* operator. This operator is motivated by the *artificial immune system* (AIS)-driven clonal selection algorithm which is frequently used in solving complex numerical optimization problems [15–18]. Our main contribution is to merge some locally defined optimization measures into a global objective function that can jointly optimize the quality of the entire tile set. This objective function balances the qualities among tiles and can be optimized using the clonal selection operator within a simple AIS framework with reasonable computational cost.

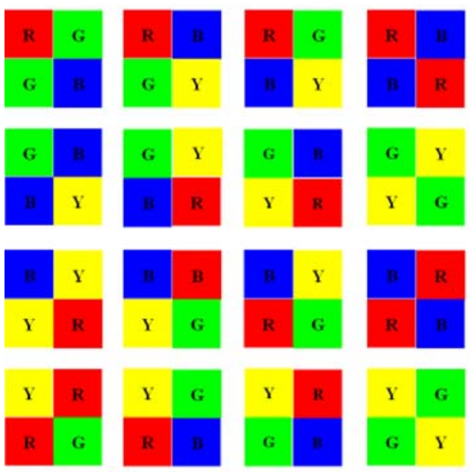
As shown in Fig. 2, an ω-tile is a square block with a specific color at each corner. A given number of small patches (*sample patches*) are extracted from an input texture to form different texturing blocks. As shown in Fig. 2 (the left-most column), a tile is cut from the center of each block to obtain the *intermediate tile*, which is the combination of four sample patch quarters. The seams in each intermediate tile are removed by replacing the interior of the tile with other pattern (*matching patch*) from the input example to generate an ω-tile. The color at each corner of an ω-tile indicates the color of the patch that contributes to the corner. We propose a global optimization algorithm to search for a feasible set of sample patches. The algorithm insures the intermediate tiles formed by these patches to satisfy both local and global optimal conditions. The local condition implies that each intermediate tile in the set could find an adequately well *matching patch* (the texture patch picked from the input example to merge into the intermediate tile for erasing junctions) from the input, while the global condition means that the tile qualities are balanced according to their matching errors with closest matching patches. The two conditions are interpreted together as an *evaluation measure*. This function is defined as the linear combination of the sum of the *matching errors* (the distance between the intermediate tile and the candidate matching patch) between intermediate tiles and their *closest matching patches* (the candidate matching patch with the smallest distance), and the standard variance of all the errors. Sample patches selection proceeds by optimizing this evaluation function using AIS. We use the graph-cut [4] method to merge the matching patches into the intermediate tiles.

The rest of the paper is organized as follows. In Sect. 2, we review some related work. An extendable rule for deriving new tile sets is described in Sect. 3. We present the AIS-based sample patches selection algorithm in Sect. 4. Finally, we show results and conclude the paper in Sects. 5 and 6.

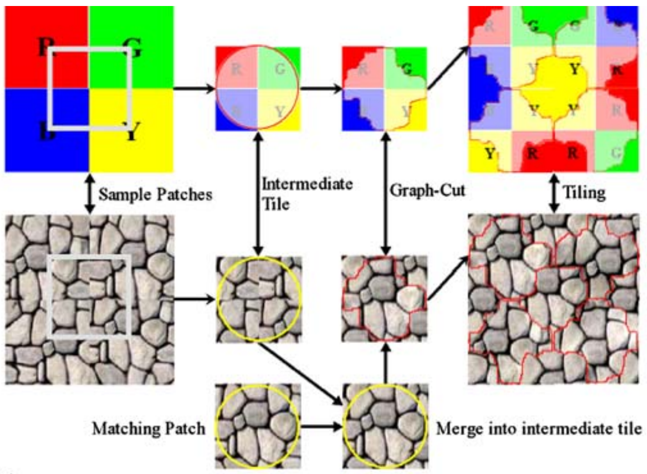
2 Related work

2.1 Texture synthesis

A number of work has been presented toward synthesizing textures from input examples. Local region-growing techniques generate textures one pixel or one patch at a time.



(a) Set of 16 ω-tiles in [1].



(b) Basic construction process of an ω-tile. The intermediate tile is merged with its matching patch from the input with graph-cut.

Fig. 2 The previous ω-tile set formation process. The four texture patches in the left-most column of (**b**) are *sample patches*, corresponding to the four color squares $\{R, G, B, Y\}$. The sample patches are robustly selected using AIS in this paper

Pixel-based synthesis algorithms [1, 2, 19, 20] grow an output texture pixel by pixel, normally using spatial neighborhood compare to match across different frequency bands. These approaches are fit for stochastic textures, but usually fail on textures with more coherent structures. Patch-based methods [3, 4, 21, 22] copy selected source regions into the output instead of single pixels. They are generally more successful on synthesizing structural textures. Some intermediate techniques [9, 23] between pixel and patch-based methods have also been presented, which somewhat combine the advantages of both. None of the techniques above can avoid laborious neighborhood matching in the input example and this time-consuming process limits their use to only off-line synthesis applications. Recently, efficient GPU-based texture synthesis techniques [24, 25] have also been proposed; however, they always demand a high performance graphics hardware, and their methods suffer from the pixel-based synthesis issue of performing poorly on textures with semantic structures not captured by small neighborhoods. On the other hand, some near real-time texture synthesis methods usually achieve low-quality results for the lack of optimization in the preprocessing [26, 27] or need very complex pre-computation and storing data structures [28]. They are also not available in many real-time environments.

In the work concurrent with ours, Cohen et al. [11] developed a stochastic algorithm to non-periodically tile the plane with a small set of Wang-tiles at run time. Wei [12] extended this work with GPU to improve tile-based texture mapping. Ng et al. [10] presented another approach to generate a set of small texture tiles from an input example. These tiles can also be tiled together to synthesize large textures. Our technique uses their ω-tiles as the tile set pattern. Figure 2 shows a typical ω-tile set in [10]. All these approaches require a set of sample patches extracted from the input example to generate the intermediate tile patterns, so the quality of their texture tiling results is not stable due to the uncertainty of the sample patches. Dong et al. [13, 14] used *genetic algorithm* (GA) to select an optimal set of sample patches from the input and achieved better tiling quality than [11]. Unfortunately, the limitation of GA sometimes plunges the objective into a local optimal solution.

2.2 Artificial immune system and clonal selection

The human immune system (HIS) protects the body against damages from an extremely large number of harmful bacteria and viruses, termed as pathogens. It realizes this largely without any prior knowledge of the structure of these pathogens. An increasing amount of work is being carried out attempting to understand and extract the key mechanisms through which the HIS is able to achieve its detection and protection capabilities. A number of artificial immune systems (AISs) have been built for a wide range of applications including document classification, fraud detection, and network- and host-based intrusion detection [29–31]. These AISs have met with some success and in many cases have rivalled or bettered existing statistical and machine learning techniques. It is also a powerful technique that can be applied to texture synthesis.

The clonal selection algorithm is used by the natural immune system to define the basic features of an immune response to an antigenic stimulus. It establishes the idea that only those cells that recognize the antigens are selected to proliferate. The selected cells are subject to an affinity maturation process, which improves their affinities to the selective antigens. Our approach uses a new *adaptive dynamic clonal selection algorithm* (ADCSA) as the selector (immune algorithm) of the AIS, similar to the structure of the clonal selection algorithms described in [16] and [17]. On the basis of the antibody–antibody affinity, antibody–antigen affinity and their dynamic allotting memory units along with the scale of antibody populations, this algorithm combines the stochastic searching methods with evolutionary searching based on the probability. Its performance is better than the classical genetic algorithm and the traditional clonal selection algorithm like [29].

3 Tile set formation

We choose the size-8 ω-tile set in [10] as the basic tile set, as shown in Fig. 3. We use $\mathbb{B}$ to represent it in this paper. A set of squares $\mathbb{P} = \{R, G, B, Y\}$ are used to compose blocks and then slice the central parts to construct the ω-tiles.

The previous ω-tile construction process is shown in Fig. 2. The approach starts with randomly obtaining a set $\mathbb{T}$ of square sample patches from the input example $\mathcal{S}$, the number is always equal to the members in $\mathbb{P}$ (four in [10]). Each patch is assigned to be related with one color square. Then an intermediate tile $\mathcal{I}$ can be cut from the middle of the texture block, as shown in the left two columns of Fig. 2. Different intermediate tiles are generated according to the different arrangements of the sample patches (corresponding to the arrangements of the color squares). We note that the cross junctions where four quarters meet have to be carefully erased. This process is shown in the middle two columns of Fig. 2. Similar with [10], we also pick a matching patch (patch offset) $\mathcal{C}$ from the input and merge it into the intermediate tile by graph-cut. The matching patch is the same size as the intermediate tile. A circle is employed to constrain the boundary of the cutting curve to maintain the continuity of the patterns between matching sides in the tiling image.

3.1 Tile patterns analysis

The tiling process using ω-tiles is carried out in the scan-line order. Once the tile in the left-top corner is fixed, the rest tiles in the tiling are laid one by one from left to right, and top to bottom, consistent with the square colors of their neighbors. A valid tiling using $\mathbb{B}$ is shown in Fig. 3. The basic tile set composed of only eight tiles will draw undesirable repetitive patterns in a large synthesis image. Two methods are proposed in [10] to overcome this artifact. One way is to pick two patches from the input for each original tile. This method doubles the number of the tiles but assumes at least two choices for each tiling step.

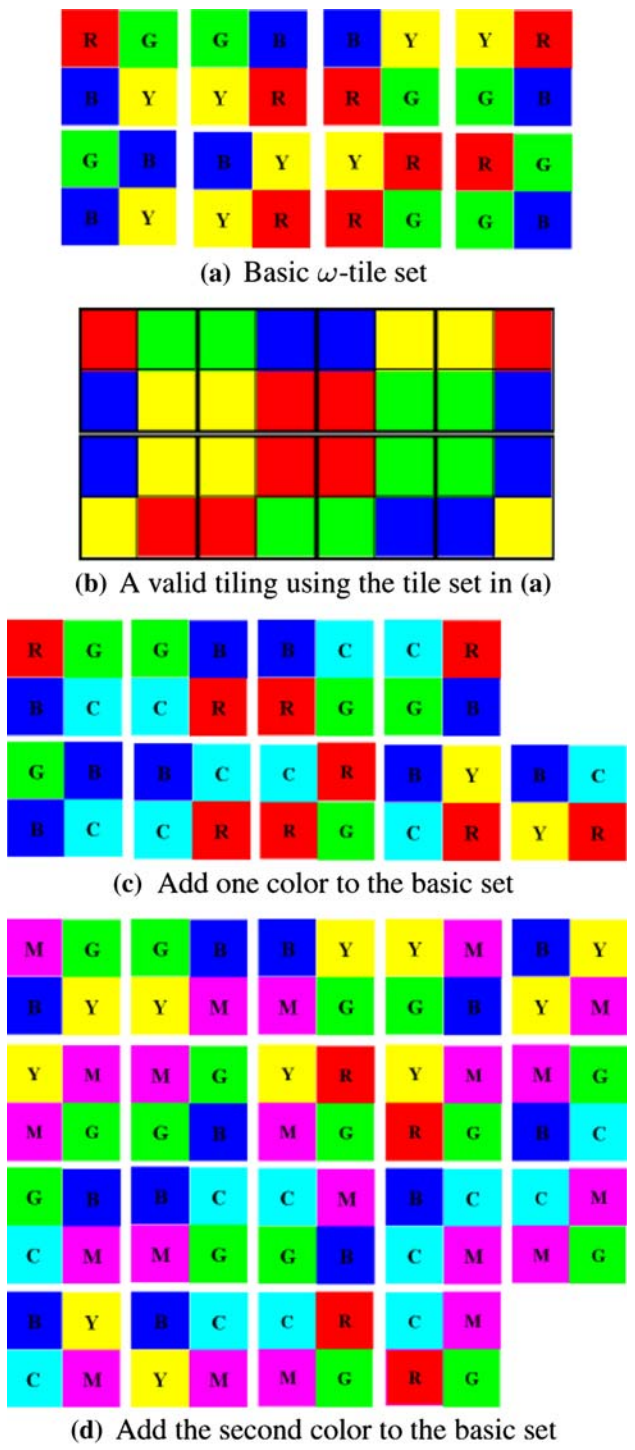

(a) Basic ω-tile set
(b) A valid tiling using the tile set in (a)
(c) Add one color to the basic set
(d) Add the second color to the basic set

Fig. 3 Tile set formation from basic size-8 ω-tile set

However, it can only partly eliminate the repetitive patterns in the tiling. We should not neglect the repetitive patterns caused by the central parts of the sample patches. As shown in the rightmost column of Fig. 2, the central pattern of the sample patch still possesses an important role in the tiling. To solve this problem, we develop an effective method to directly increase the number of the sample patches which are used to form the intermediate ω-tiles, without losing the characteristics of the whole tile set. The other way used in [10] to eliminate repetition is to increase the tiles number by using more arrangements of the sample patches. This method is also suitable for our approach. However, it still cannot avoid the repetitive patterns caused by the sample patches themselves.

3.2 Increase sample patches

We add new patterns into the ω-tile set by enlarging the size of the sample patches set $\mathbb{T}$. This operation is proceeded directly on $\mathbb{B}$. We derive new tiles with the following steps:

1. Randomly pick a square from set $\mathbb{P}$ as the "reference" square (or reference color). Without losing generality, we choose the yellow square as the reference square.
2. Add a new square to the set $\mathbb{P}$. Here, we use C (Cyan) to represent it. Then the new square set is enlarged to be $\mathbb{P}_1 = \{R, G, B, Y, C\}$.
3. Generate new tiles by replacing the yellow squares with cyan squares in $\mathbb{B}$.
4. Add another two tiles by replacing only one yellow square with cyan in the tile $\langle B, Y, Y, R\rangle$.

The new tiles formed by the above method is shown in Fig. 3. The last two tiles are the additional tiles generated by step 4. The new tiles together with $\mathbb{B}$ forms the new ω-tile set $\mathbb{N}_1$. Despite the two "additional" tiles, the other new tiles plus the tile $\langle R, G, G, B\rangle$ can be considered as a copy of the basic set $\mathbb{B}$. It can be used independently to tile arbitrary size area. We use $\mathbb{N}_1'$ to indicate the tile set which possesses all the tiles in $\mathbb{N}_1$ except the two additional tiles. During the tiling process, if only $\mathbb{N}_1'$ is used, there will be no problem if no yellow square and cyan square need to be matched at the same time when placing a tile. This can be treated as a pattern duplication with the basic tile set. The cases shown in Fig. 4a, b appear where yellow square and cyan squares are needed to be matched simultaneously; there will be no matching tile in $\mathbb{N}_1'$. Apparently, here the two additional tiles are necessary to complete the tiling process. In fact, these two patterns are just the extensions of the pattern in Fig. 4, which is one of the tiling patterns

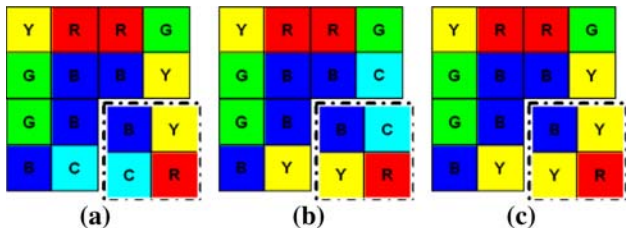


Fig. 4 The cases why the two additional tiles in Fig. 3 are required

using the basic set $\mathbb{B}$. Therefore, the new tile set $\mathbb{N}_1$ can be used to tile any large area and contains all the properties of the ω-tile set which are stated in [10].

Comparing with the size-16 tile set shown in Fig. 2, with the similar number of tiles, our new set $\mathbb{N}_1$ enriches the patterns brought by sample patches in the tiling. And the patterns of the tiles themselves are also more diverse because of the integration of more sample patches than simply using more different arrangements of the squares. We note, additionally, the diagonal repetition tiles (same color squares share a diagonal) are also reduced. This also avoids some repetitive patterns in a tiling.

We can continue to increase the number of sample patches into six by following the same rule. On the basis of the tile set $\mathbb{N}_1$, we pick red as the reference color, and generate new tiles by replacing the red squares with magenta. The tiles with double red squares also need to be derived into two additional tiles by replacing only one square at a time. The new tiles are shown in Fig. 3. Packed with set $\mathbb{N}_1$, we get another new ω-tile set $\mathbb{N}_2$. Here the square set is enlarged to $\mathbb{P}_2 = \{R, G, B, Y, C, M\}$. Obviously, we can keep on increasing the sample patches in $\mathbb{P}_2$ by following this rule if there is enough memory during the rendering process. Then in the tiling process, this method can effectively eliminate the repetitive patterns caused by both the tiles and the sample patches themselves. Usually, the set $\mathbb{N}_2$ which contains 38 tiles is enough to achieve a less-repetitive tiling result.

4 Tile construction

The quality of tiling is decided by the quality of the final tile set. As illustrated in Fig. 2, the graph-cutting result performed between the intermediate tile and the matching patch will directly affect the quality of the tile. Thus, special care should be taken when picking sample patches from the input, so that the intermediate tiles can get feasible matching patches for graph-cut operations. The random picking method used in [11] and [10] is not robust for this situation. The intermediate tile formed by randomly chosen sample patches cannot assure of finding a good matching patch $\mathcal{C}$ from the example. On the other hand, because of the great quantity of pixels in an image, it is almost impossible to use brute-force searching method to find the best sample patches extraction. For example, for an input example of size 128×128, if we set the tile size to be 80×80, there will be $(48 \times 48)^n$ choices for the sample patches extraction (n is the number of sample patches). This is an unacceptable computational requirement for a normal PC in reasonable time. The GA-based sample patches selection technique in [13] achieves better results for Wang-tiles than [11]. However, it sometimes looses the best solution due to the limitation of the evolutionary framework. Our approach can efficiently and accurately solve the sample patches selection problem.

4.1 Algorithm overview

Our robust sample patches selection algorithm is essentially based on AIS. AISs use ideas gleaned from immunology to develop adaptive systems which are capable of performing a wide range of optimization tasks in various research areas. A standard AIS starts with an initial set of random-generated antibodies (immune cells) called a population where each antibody encodes a solution of the optimization problem. All antibodies are evaluated with the antigen by an evaluation function which is some measure of affinity. A selection process based on the affinity values will form a new population. A cycle from one population to the next is called a generation. During each new generation, all antibodies will be updated by the immune operations. Then, the selection process selects antibodies to form a new population. After performing a given number of cycles, or when other termination criteria are satisfied, we denote the best antibody as a solution, which is regarded as the optimal solution of the optimization problem.

From the above, we can see that to design an AIS, it is necessary to choose an appropriate *affinity measure* and an *immune algorithm*. For the problem of sample patches selection, we encode each antibody as a candidate set of sample patches, using the location of each patch at the input as a gene. Clone selection is employed as the immune algorithm in our AIS and we will define the affinity measure in the following section.

The theory known as clone selection is used to explain how the immune system "fights" against an antigen. When a bacterium invades our organism, it starts multiplying and damaging our cells. One form of the immune system found to cope with this replicating antigen was by replicating the immune cells successful in recognizing and fighting against this disease-causing element. Those cells capable of recognizing the antigen reproduce themselves asexually in a way proportional to their degree of recognition; the better the antigenic recognition, the higher the number of offspring (clones) generated. During the process of cell division (reproduction), individual cells suffer a mutation that allows them to become more adapted to (increase affinity with) the antigen recognized: the higher the affinity of the parent cell is, the lower the mutation they suffer. We apply these characteristics to design the immune algorithm in the AIS for sample patches selection.

4.2 Robust sample patches selection

The original problem of sample patches selection is to find the optimal n sample patches from the input example to fill the m tiles with a maximum objective function. For example, for the ω-tile set $\mathbb{N}_2$, $n = 6$, $m = 38$. Then the intermediate tile filled by the optimal sample patches can find a feasible matching patch safely under the given rules for junction flattening. We will make use of clone selection in the AIS to avoid suboptimal solutions. The AIS framework of finding the optimal n sample patches is described as follows.

4.2.1 Initialization

To ensure that an optimal solution can be obtained in a reasonable runtime, an initial population consists of a considerable amount of antibodies is necessary. To start the algorithm, an integer N_p is defined as the number of antibodies. From the input texture, $N_\mathrm{p}/2$ antibodies are randomly chosen, i.e. randomly choose n sample patches from the valid region for each antibody. For the other half, we uniformly divide the valid region into n parts and randomly choose one patch from every part. Then the patches are also randomly arranged to be the genes of each antibody. The antibodies are denoted by $\mathcal{P} = \{A_1, A_2, \ldots, A_{N_\mathrm{p}}\}$, we call it as a population. Every antibody contains n sample patches (n genes) selected from the input example:

$$A_i = (g_i^1, g_i^2, \ldots, g_i^n), \quad i = 1, 2, \ldots, N_\mathrm{p}$$

4.2.2 Evaluation

In our AIS, the clone number of antibodies is determined by a percentage assigned to each antibody. This percentage is proportional to its affinity relative to other antibodies in the population, i.e. antibodies with higher affinities will have more number of clones to produce offsprings in the clone selection process. In the context of sample patches selection, the antigen is defined as the set of valid matching patches in the input example. The *antibody–antigen affinity* of an antibody is evaluated by an evaluation function, which in essence, computes the minimum matching error between the intermediate tile and the candidate matching patch. It involves searching for translations of the input image that match well with the intermediate tile. Let $I(p)$ and $T(p)$ be the pixel colors at the position p in the input example and the intermediate tile, the evaluation function is defined as

$$C_j(t) = \frac{1}{|S_t|} \sum_{p \in S_t} |T(p) - I(p - t)|^2, \quad (j = 1, 2, \ldots, m; t \in P_T) \tag{1}$$

$$\begin{aligned} D_j &= \min\{C_j(t), t \in P_T\} \\ V_i &= \mathrm{variance}(D_1, D_2, \ldots, D_m) \\ E(A_i) &= \lambda \cdot \sum_{j=1}^{m} D_j + (1 - \lambda) \cdot V_i \end{aligned} \tag{2}$$

where $C_j(t)$ is the matching error at translation t of the jth tile, S_t is the portion of the translated input overlapping the tile,

P_T is the set of valid translations (candidate matching patches) in the input, and D_j is the minimum matching error within all the translations. V_i is the variance of all the minimum errors, we add this factor to protect the global quality of the final tile set. It avoids that intermediate tiles with extremely high and extremely low matching errors appear together in the same set. So the evaluation function $E(A_i)$ is the linear combination of the minimum error sum and the variance. We set $\lambda = 0.8$ for all the experiments in this paper. Note that our evaluation function is very similar to the energy function used in [9] for texture optimization, while here we use AIS rather than expectation maximization (EM) to optimize it.

To obtain an optimal tile set, we need to determine the best set of sample patches that has the smallest evaluation value. Hence, to be consistent with the concept of affinity, the antibody–antigen affinity function $f(A_i)$ is defined as the reciprocal of the evaluation function, i.e. $f(A_i) = 1/E(A_i)$. As previously addressed, it is also the objective function to be optimized by our AIS. On the basis of the affinity value of each antibody, the population $\mathcal{P}$ is rearranged from high affinity to low affinity.

4.2.3 *Antibody partition*

According to the affinities, we adaptively allot the antibody population. $\mathcal{P}$ is divided into the memory unit $\mathcal{M}$ and the generic antibody unit $\mathcal{A}_b$:

$$\begin{aligned}
&\mathcal{P} = \{\mathcal{M}, \mathcal{A}_b\}\\
&N_\mathrm{m} = \lfloor N_\mathrm{p} \cdot \min(s^{\mathcal{M}}_{\min} + h, s^{\mathcal{M}}_{\max})\rfloor, (s^{\mathcal{M}}_{\min}, s^{\mathcal{M}}_{\max} \in (0,1))\\
&\mathcal{M} = \{A_1, A_2, \ldots, A_{N_\mathrm{m}}\}, \quad \mathcal{A}_b = \{A_{N_\mathrm{m}+1}, A_{N_\mathrm{m}+2}, \ldots, A_{N_\mathrm{p}}\}
\end{aligned}$$

where $s^{\mathcal{M}}_{\min}$ is a constant set to assure the minimum size of the memory unit, while $s^{\mathcal{M}}_{\max}$ is the upper limit of it. Otherwise

$$h = \frac{\sqrt{\frac{1}{(N_\mathrm{p}-1)\cdot N_\mathrm{p}} \sum_{i=1}^{N_\mathrm{p}} \sum_{j=1}^{N_\mathrm{p}} H_{ij}}}{|u-l|}$$

which is used to measure the diversity of antibody population, $0 < h < 1$, the bigger h is, the better is the diversity. u and l are separatively the upper and lower limit of the antibody genes, here in our algorithm are the left-top and right-bottom coordinate of the valid region for sample patches at the input example. H_{ij} represents the *antibody–antibody affinity*, which is defined as

$$H_{ij} = \|A_i - A_j\| = \sum_{k=1}^{n} \|g_i^k - g_j^k\|, \quad i,j = 1,2,\ldots,N_\mathrm{p}$$

where $\|\bullet\|$ denotes the Euclidean distance of two sample patch locations. Apparently, H is a symmetrical matrix which indicates the diversity of the antibody population.

In this case, the memory unit is corresponding to the memory cells in the human immune system which are able to bind successfully to an antigen. Different mutative principles will be used for memory unit and generic unit in the *clone selection* process.

4.2.4 *Mutation regulation*

We dynamically regulate the mutative probabilities of the antibodies inversely proportional to their antigenic affinities: the higher the affinity, the smaller the mutation rate. Following this rule, the corresponding mutative probability p^i_mu of each antibody is evaluated as

$$p^i_\mathrm{mu} = p^c_\mathrm{mu} + \left[1 + \exp\left(n \cdot \frac{f(A_i)}{\sum_{j=1}^{N_\mathrm{p}} f(A_j)}\right)\right]^{-1}$$

where p^c_mu is a constant to assure the minimum mutative probability of the antibody. Then a further correction is made by

$$p^i_\mathrm{mu} = \begin{cases} p^{\mathcal{M}}_\mathrm{mu}, & p^i_\mathrm{mu} > p^{\mathcal{M}}_\mathrm{mu}, \quad 1 \le i \le N_\mathrm{m} \\ p^{\mathcal{A}_b}_\mathrm{mu}, & p^i_\mathrm{mu} < p^{\mathcal{A}_b}_\mathrm{mu}, \quad N_\mathrm{m}+1 \le i \le N_\mathrm{p} \end{cases}$$

where $p^{\mathcal{M}}_\mathrm{mu}$ and $p^{\mathcal{A}_b}_\mathrm{mu}$ are mutative threshold value of memory unit and generic antibody unit respectively, generally, $p^{\mathcal{M}}_\mathrm{mu} \ll p^{\mathcal{A}_b}_\mathrm{mu} < 1$. In our program, given the minimum memory unit size $s^{\mathcal{M}}_{\min}$, it yields

$$\begin{aligned}
p^c_\mathrm{mu} &= (s^{\mathcal{M}}_{\min} + h)/2.0\\
p^{\mathcal{M}}_\mathrm{mu} &= \min(1.4 \cdot p^c_\mathrm{mu}, 0.3)\\
p^{\mathcal{A}_b}_\mathrm{mu} &= \min(3.0 \cdot p^{\mathcal{M}}_\mathrm{mu}, 0.8)
\end{aligned}$$

These probabilities will be used in the mutation step of clone selection.

4.2.5 *Clone selection*

The process of clone selection in an AIS can be treated as a refinement of the population. It is the most important part of our sample patches selection algorithm. All the antibodies will be applied to this operator independently. We describe our clone selection operator as follows:

1. Clone the ith antibody proportionally to its antibody–antigen affinity, generating a repertoire

$$\mathcal{R}_i = \{A_i^1, A_i^2, \ldots, A_i^{N_c}\}$$

of clones: the higher the affinity, the higher the number of clones. The clone number N_c^i is given by

$$\begin{aligned}
\mu &= \frac{\beta \cdot N_\mathrm{p} \cdot f(A_i)}{\sum_{j=1}^{N_\mathrm{p}} f(A_j)}\\
N_\mathrm{c}^i &= \begin{cases} \max(\alpha \cdot N_\mathrm{p},\ N_\mathrm{p}/3), & 1 \le i \le N_\mathrm{m} \\ \min(\alpha \cdot N_\mathrm{p},\ N_\mathrm{p}/3), & N_\mathrm{m}+1 \le i \le N_\mathrm{p} \end{cases}
\end{aligned}$$

where α is the threshold value for both memory unit and generic unit, β is a multiplying factor, N_p is the total number of antibodies. The antibody with the highest affinity will produce the most clones.

2. The repertoire $\mathcal{R}_i$ is submitted to an affinity maturation process according to its mutative probability and generate a population $\mathcal{R}'_i$ of mutated clones. We use a method similar to BGA mutation [32] in this step. Let $A_i^j = \left\{g_{i,1}^j, g_{i,2}^j, \ldots, g_{i,n}^j\right\}$ be the jth cloned antibody of A_i which is selected to be mutated with probability p_{mu}^i, we randomly pick a gene $g_{i,k}^j$ (k = 1, 2, …, n) for this mutation, then the offspring is denoted as $A_i'^j = \left\{g_{i,1}^j, g_{i,2}^j, \ldots, g_{i,k}'^j, g_{i,k+1}^j, \ldots, g_{i,n}^j\right\}$. In BGA mutation, $g_{i,k}'^j$ is $\left(g_{i,k}^j + 0.1 \cdot \delta \cdot (u - l)\right)$ or $\left(g_{i,k}^j - 0.1 \cdot \delta \cdot (u - l)\right)$ with equal probability. Here $\delta = \sum_{i=0}^{15} \gamma_i 2^{-i}$, $\gamma_i \in \{0, 1\}$ and γ_i takes 1 with probability of 1/16. u and l is the upper limit and lower limit of the gene as addressed previously in the *Antibody Partition* step. If $g_{i,k}'^j$ exceed the interval [l, u], we will change it to $\left(g_{i,k}'^j + 0.2 \cdot (u - l)/1.4\right)$.
This mutation method tests frequently the patches which are close to $g_{i,k}^j$, so it trends toward local search. Furthermore, it is independent of the location in the phenotype space.
3. Determine the affinity $f\left(A_i'^j\right)$ of the mutated clones $\mathcal{R}'_i$.
4. From this set of mutated clones $\mathcal{R}'_i$, reselect the one with highest antigenic affinity $A_i'^j$ to be a candidate to enter the set of new antibody population. If the affinity of this antibody is larger than its respective original antibody A_i, then $A_i'^j$ will replace A_i in the new population.

4.2.6 *Antibody regulation*

Resort the antibodies from high affinity to low affinity, replace (antibody death) the N_{d} lowest affinity antibodies from the population by new randomly chosen individuals.

4.2.7 *Termination*

Two termination criteria are used. Either the process is executed to produce a fixed number N_{g} of generations, or no further improvement for the best solution is observed in N_{o} consecutive generations. If the termination condition is satisfied, the AIS will be terminated and the best solution (the antibody with highest affinity) among all the individuals is chosen. Otherwise, the AIS goes back to the evaluation step and begins the next iteration.

4.3 Working flow

The AIS-enhanced sample patches selection algorithm can effectively reduce the boundary artifacts caused by the mergence of the intermediate tile and its matching patch during the graph-cut process. The whole working flow of our optimized tile-based texture synthesis algorithm is, in pre-computation, first randomly initialize a considerable number of sample patches sets (as the antibodies) from the input example, then use AIS to find the best one. Finally graph-cut is employed for junction elimination. The run-time tiling process is the same as [10], we can synthesize arbitrary size of texture images in real time. Note that we also use Poisson smoothing [33] to remove the prominent seams in ω-tiles after the graph-cut operations.

4.4 Parameters and optimizations

The basic parameters of the AIS are the antibody population size N_{p}, the upper and lower limit of memory unit size $s_{\min}^{\mathcal{M}}$ and $s_{\max}^{\mathcal{M}}$, the threshold α and multiplying factor β for mutative probabilities, the antibody death number N_{d}, the maximum iteration times N_{g}, and the maximum no-improvement times N_{o}. In our approach, we use the same settings for all the experiments with $\{N_{\mathrm{p}} = 40, s_{\min}^{\mathcal{M}} = 0.2, s_{\max}^{\mathcal{M}} = 0.4, \alpha = 0.4, \beta = 4, N_{\mathrm{d}} = 10, N_{\mathrm{g}} = 50, N_{\mathrm{o}} = 6.\}$.

The most time-consuming procedure in our AIS is the fitness evaluation of the chromosomes. In this procedure, we use approximate-nearest-neighbor search (ANN) [34] to accelerate the neighborhood matching operation between each intermediate tile and the input example. Note that the other techniques such as TSVQ [2, 35], FFT [36, 37] and mixture trees [38] can also be employed here.

5 Results and discussions

Figure 5 shows some texture synthesis results using our algorithm. All the results are tiled in real-time with the ω-tile set $\mathbb{N}_2$ which contains 38 tiles (illustrated in Fig. 3). Execution times of the AIS-based sample patches selection process are listed in Table 1. All timing results are reported for our unoptimized C++ code on a Pentium 4 3.2 GHz PC with 1 GB RAM. The sequence of the example names is consistent with the image positions in Fig. 5, from left to right and from top to bottom. The timings indicate that using AIS is an efficient way to select feasible sample patches. We could see that our algorithms could generate high-quality results for random and semi-structural texture examples, while still causes apparent artifacts for high-structural textures, like the eggs example.

We can generate the distance function defined in Eq. 1 to incorporate other characteristics of the texture besides color. For example, to use image gradients as an additional similarity metric, we could define the distance as

$$C(t) = \sum_{p \in S_t} \frac{|T(p) - I(p-t)|^2}{\mu(\|\nabla T\| + \|\nabla I\|)} \tag{3}$$

or

$$C(t) = \sum_{p \in S_t} |T(p) - I(p-t)|^2 + \mu \sum_{p \in S_t} |\nabla T - \nabla I|^2 \tag{4}$$

where $\nabla = \left[\frac{\partial .}{\partial x}, \frac{\partial .}{\partial y}\right]$ is the gradient operator and μ is a relative weighting coefficient ($\mu = 10$ in our experiments). Figure 6 shows the synthesis results using different distance metrics. Even though we have experimented with color and gradient, one could use other distance metric which measures some property of the texture patch. For most textures, we can

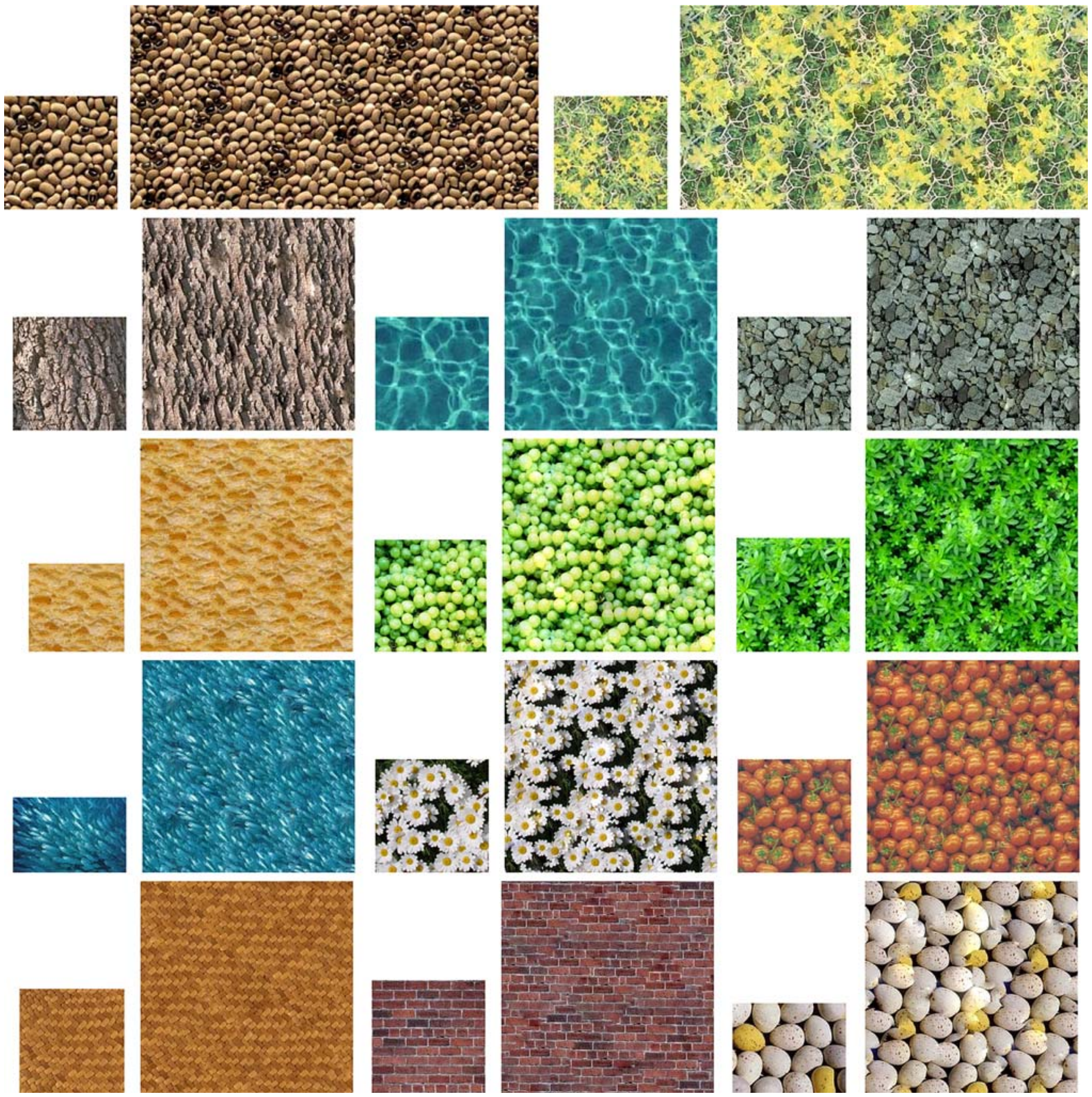


Fig. 5 Results for image texture synthesis. For each texture, the input is on the *left* and the output on the *right*. All results are generated in real-time with the corresponding ω-tiles

Table 1 Sample patches selection timings for the examples in Fig. 5

Example name	Example size	Tile size	AIS timing
Beans	128 × 128	80 × 80	1 min 8 s
Yellow leaves	160 × 160	140 × 140	2 min 38 s
Tree barks	128 × 128	80 × 80	1 min 12 s
Caustics	128 × 128	80 × 80	1 min 10 s
Stones	128 × 128	80 × 80	0 min 58 s
Bread	108 × 99	70 × 70	1 min 5 s
Grape	144 × 144	100 × 100	1 min 56 s
Grass	128 × 128	80 × 80	1 min 18 s
School	128 × 85	70 × 70	1 min 5 s
Flowers	128 × 128	80 × 80	1 min 9 s
Tomatoes	128 × 128	80 × 80	1 min 14 s
Fabric	128 × 128	80 × 80	1 min 17 s
Bricks	128 × 128	80 × 80	1 min 1 s
Eggs	128 × 102	80 × 80	1 min 54 s

simply use the color as the distance metric, as all the experiments in Fig. 5 do.

The most important parameters in AIS are the population size *pop_size* and the generation number N_g. In Figs. 7 and 8 we show comparisons of using different population sizes and different generation numbers in AIS. The other parameters are set to be the same as in Fig. 5. The input examples are the same as the input texture in the first row in Fig. 9. We can see the quality increase of the output textures when the parameters change. Table 2 shows the comparisons of the AIS training results using different population sizes and generation numbers. For each result, the left value in the brackets is the antibody–antigen affinity value of the best antibodies when AIS terminates, and the right value is the AIS training time. The random operation means that we calculate the average evaluation value of 100 randomly obtained sample patches sets. And the semi-random operation means that we compute the minimum evaluation value

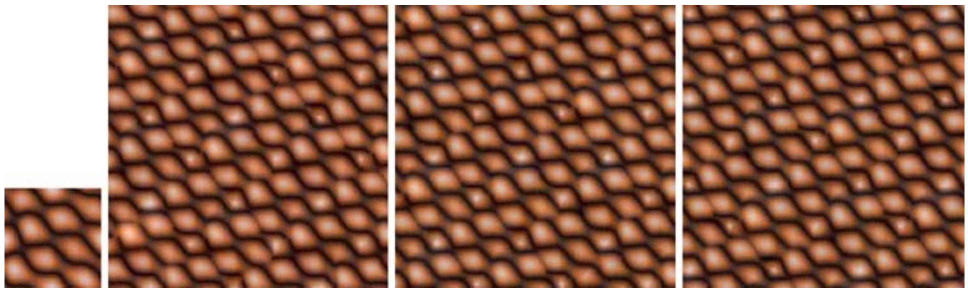


Fig. 6 Results comparison when using different distance metrics. From *left* to *right*: the input example, result using Eqs. 1, 3 and 4

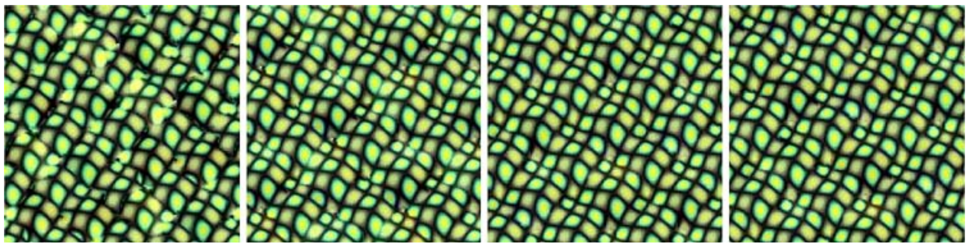


Fig. 7 Results comparison of using different number of antibodies in AIS. The size of input example is 64 × 64 and the tile size is 48 × 48. N_g = 50. From *left* to *right*: N_p = 10, 20, 40, 80, the antibody–antigen affinity f = 0.988, 1.211, 1.396, 1.412, training time: 8″, 13″, 26″, 48″

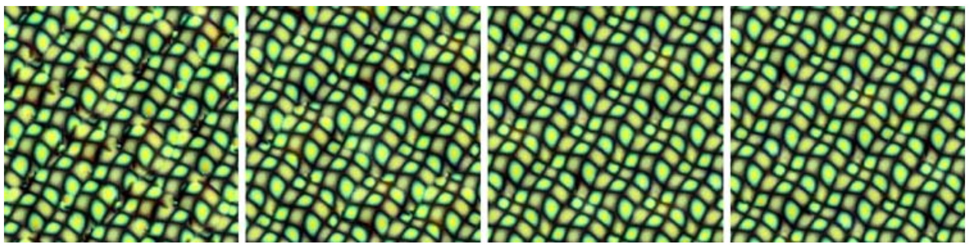


Fig. 8 Results comparison of using different generation numbers in AIS. The size of input example is 64 × 64 and the tile size is 48 × 48. N_p = 40. From *left* to *right*: N_g = 10, 30, 50, 80, the antibody–antigen affinity f = 0.972, 1.269, 1.385, 1.403, training time: 6″, 15″, 33″, 51″

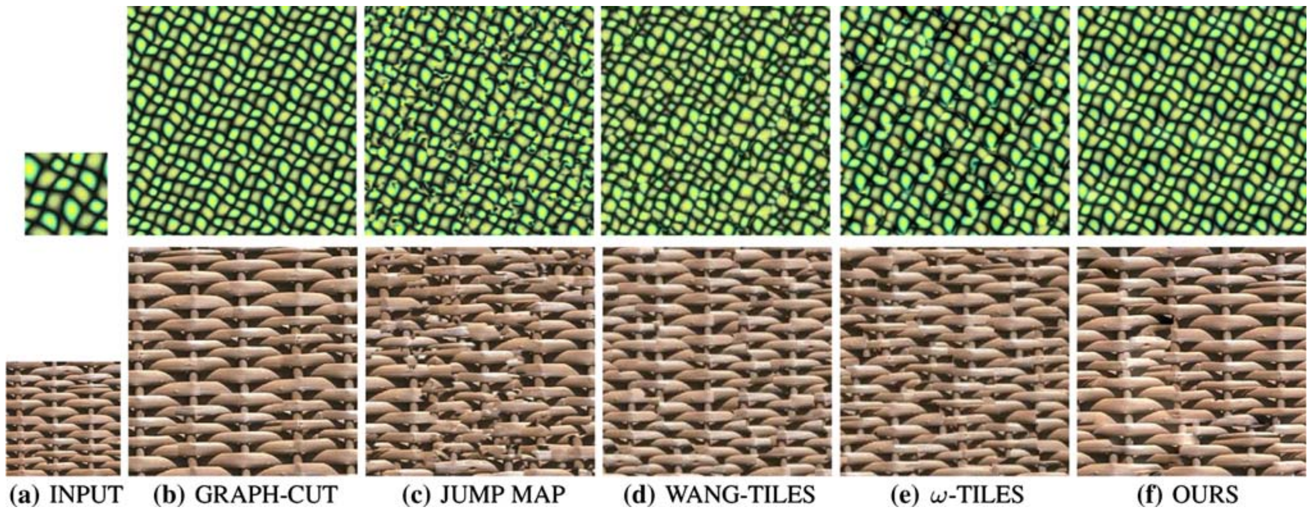


Fig. 9 Comparison of texture synthesis results with various other techniques. Results for other techniques are obtained from their web pages

Table 2 The antibody–antigen affinity value of the best antibodies when AIS terminates and AIS training times using different parameters

Example name	(N_p, N_g)							
	Random	Semi-random	(10, 10)	(20, 30)	(20, 50)	(40, 30)	(40, 50)	(40, 80)
Beans	(0.179, –)	(0.244, –)	(0.297, 18″)	(0.316, 34″)	(0.339, 56″)	(0.347, 58″)	(0.353, 1′8″)	(0.385, 2′24″)
Yellow leaves	(0.069, –)	(0.083, –)	(0.093, 39″)	(0.122, 1′13″)	(0.129, 1′17″)	(0.134, 1′34″)	(0.143, 2′38″)	(0.155, 6′7″)
Tree barks	(0.373, –)	(0.441, –)	(0.468, 16″)	(0.475, 35″)	(0.497, 49″)	(0.511, 56″)	(0.532, 1′12″)	(0.541, 2′7″)
Caustics	(0.137, –)	(0.219, –)	(0.266, 15″)	(0.271, 41″)	(0.283, 47″)	(0.296, 1′11″)	(0.315, 1′10″)	(0.322, 2′17″)
Stones	(0.247, –)	(0.338, –)	(0.358, 11″)	(0.361, 32″)	(0.387, 44″)	(0.392, 46″)	(0.418, 58″)	(0.429, 2′7″)
Bread	(0.164, –)	(0.229, –)	(0.268, 8″)	(0.276, 29″)	(0.271, 46″)	(0.273, 44″)	(0.285, 1′15″)	(0.291, 2′11″)
Grape	(0.237, –)	(0.319, –)	(0.362, 17″)	(0.371, 42″)	(0.397, 1′4″)	(0.411, 1′26″)	(0.435, 1′56″)	(0.471, 4′49″)
Grass	(0.230, –)	(0.275, –)	(0.351, 14″)	(0.384, 38″)	(0.411, 50″)	(0.417, 57″)	(0.423, 1′18″)	(0.428, 2′28″)
School	(0.271, –)	(0.352, –)	(0.387, 11″)	(0.399, 33″)	(0.416, 42″)	(0.428, 51″)	(0.451, 1′5″)	(0.465, 1′59″)
Flowers	(1.092, –)	(1.171, –)	(1.181, 18″)	(1.202, 36″)	(1.297, 47″)	(1.307, 50″)	(1.336, 1′19″)	(1.368, 2′40″)
Tomato	(0.165, –)	(0.204, –)	(0.257, 19″)	(0.265, 32″)	(0.271, 46″)	(0.273, 51″)	(0.296, 1′14″)	(0.311, 2′10″)
Fabric	(0.081, –)	(0.121, –)	(0.144, 21″)	(0.148, 37″)	(0.151, 51″)	(0.153, 1′11″)	(0.167, 1′17″)	(0.175, 2′29″)
Brick	(0.152, –)	(0.206, –)	(0.221, 14″)	(0.227, 36″)	(0.239, 44″)	(0.242, 51″)	(0.256, 1′11″)	(0.266, 2′24″)
Eggs	(0.558, –)	(0.659, –)	(0.772, 27″)	(0.793, 38″)	(0.831, 59″)	(0.834, 1′17″)	(0.998, 1′54″)	(1.055, 2′29″)

from 100 randomly obtained sample patches sets. Results show that the increase of population size and generation number both increase the affinity value of the antibodies, while simultaneously cost more training time. We can see that normally the setting of $\{N_p = 40, N_g = 50\}$ is enough for most synthesis. The setting of $\{N_p = 40, N_g = 80\}$ could achieve bigger affinity value, but the training time is nearly the double of $\{N_p = 40, N_g = 50\}$.

We compare our results with other techniques in Fig. 9. We can see that the qualities of our results are comparable with the off-line graph-cut method [4] (even though, their results outperform ours when synthesizing some structural textures) while better than the other CPU-based real-time techniques. The results in Figs. 5 and 9 show that our method is a very good choice for textures without very clear structures, especially for natural textures [20].

6 Conclusion and future work

We have presented a novel optimization-based technique for tile-based texture synthesis. Our results for both texture synthesis and image tiling are comparable to state of the art. We define a pattern repetitive principle that allows us to derive new ω-tile sets from the existing one. An optimized sample patches selection algorithm based on AIS is used to improve the quality of the whole tile set. The experimental results demonstrate that the quality of tile-based texture synthesis is markedly improved after using the proposed robust sample patches selection. This framework is also fit for quality improvement of Wang-tile-based texture synthesis [11]. In the real applications, the sample patches selection and ω-tile set construct are preceded as pre-processing. The tile sets of different textures are saved into a database. Then at run-time we only need to pick the pre-computed tile set and perform the texture tiling process in real-time. Our technique can be nicely applied in the environment where real-time texture synthesis is needed, such as 3D games and real-time virtual reality systems, while the local-region-growing methods such as image quilting, graph-cut and texture optimization are not applicable (need seconds or minutes to generate an image).

A limitation of our technique is that because it tries to erase the junctions in the intermediate tiles by a single patch from the input example, it is always constrained by the patterns of the intermediate tiles. It is manifested as relatively low qualities when synthesizing some structural textures, for example, the eggs texture in Fig. 5.

For future work, we wish to extend our tile-based synthesis technique to handle image or geometric textures on 3D models. Another potential direction is to experiment with other local-region-growing texture synthesis methods, such as texture optimization [9], fractional Fourier texture masks [39] and appearance-space texture synthesis [25], in the tile construction step to improve the synthesizing quality of structural textures.

Acknowledgments We thank Steve Zelinka, Vivek Kwatra and Tuen-Young Ng for sharing their results and texture samples on the web sites. Thank Professor Ramin Zabih and his students at Cornell University for sharing their code for computing graph min-cut [40]. This work is supported by National Natural Science Foundation of China projects no. 60073007, 60473110; by National High-Tech Research and Development Plan 863 of China under Grant No. 2006AA01Z301, and by the MOST International collaboration project no. 2007DFC10740.

References

1. Efros AA, Leung TK (1999) Texture synthesis by non-parametric sampling. In: ICCV '99: Proceedings of the international conference on computer vision, vol 2. IEEE Computer Society, Washington, p 1033
2. Wei LY, Levoy M (2000) Fast texture synthesis using tree-structured vector quantization. In: SIGGRAPH '00: Proceedings of the 27th annual conference on computer graphics and interactive techniques. ACM Press, New York, pp 479–488
3. Efros AA, Freeman WT (2001) Image quilting for texture synthesis and transfer. In: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM Press, New York, pp 341–346
4. Kwatra V, Schödl A, Essa I, Turk G, Bobick A (2003) Graphcut textures: image and video synthesis using graph cuts. ACM Trans Graph 22:277–286
5. Heeger DJ, Bergen JR (1995) Pyramid-based texture analysis/synthesis. In: SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. ACM Press, New York, pp 229–238
6. Paget R, Longstaff ID (1998) Texture synthesis and unsupervised recognition with a nonparametric multiscale markov random field model. IEEE Trans Image Process 7:925–931
7. Portilla J, Simoncelli EP (2000) A parametric texture model based on joint statistics of complex wavelet coefficients. Int J Comput Vis 40:49–70
8. Freeman WT, Jones TR, Pasztor EC (2002) Example-based super-resolution. IEEE Comput Graph Appl 22:56–65
9. Kwatra V, Essa I, Bobick A, Kwatra N (2005) Texture optimization for example-based synthesis. ACM Trans Graph 24:795–802
10. Ng TY, Wen C, Tan TS, Zhang X, Kim YJ (2005) Generating an ω-tile set for texture synthesis. In: Proceedings of computer graphics international 2005 (CGI'05), Stone Brook, NY, USA, pp 177–184
11. Cohen MF, Shade J, Hiller S, Deussen O (2003) Wang tiles for image and texture generation. ACM Trans Graph 22:287–294
12. Wei LY (2004) Tile-based texture mapping on graphics hardware. In: HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on graphics hardware. ACM Press, New York, pp 55–63
13. Dong W, Sun S, Paul JC (2005) Optimal sample patches selection for tile-based texture synthesis. In: CAD-CG '05: Proceedings of the 9th international conference on computer aided design and computer graphics (CAD-CG'05). IEEE Computer Society, Washington, pp 503–508
14. Dong W, Zhou N, Paul JC (2007) Optimized tile-based texture synthesis. In: GI '07: Proceedings of graphics interface 2007. ACM, New York, pp 249–256
15. Kim J, Bentley PJ (2001) Towards an artificial immune system for network intrusion detection: an investigation of clonal selection with a negative selection operator. In: Proceedings of the 2001 Congress on evolutionary computation CEC2001. IEEE Press, Seoul, pp 1244–1252
16. de Castro LN, Zuben FJV (2002) Learning and optimization using the clonal selection principle. IEEE Trans Evol Comput 6:239–251
17. Du H, Jiao L, Gong M, Liu R (2004) Adaptive dynamic clone selection algorithms. In: Lecture notes in computer science (RSCTC'2004 proceedings) , vol 3066, pp 768–773
18. Ishida Y (2004) Immunity-based systems: a design perspective. Springer, New York
19. Bonet JSD (1997) Multiresolution sampling procedure for analysis and synthesis of texture images. In: SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques. ACM Press, New York, pp 361–368
20. Ashikhmin M (2001) Synthesizing natural textures. In: SI3D '01: Proceedings of the 2001 symposium on interactive 3D graphics. ACM Press, New York, pp 217–226
21. Wu Q, Yu Y (2004) Feature matching and deformation for texture synthesis. ACM Trans Graph 23:364–367
22. Liu Y, Lin WC, Hays J (2004) Near-regular texture analysis and manipulation. ACM Trans Graph 23:368–376
23. Nealen A, Alexa M (2003) Hybrid texture synthesis. In: EGRW '03: Proceedings of the 14th Eurographics workshop on rendering, Aire-la-Ville, Eurographics Association, Switzerland, pp 97–105
24. Lefebvre S, Hoppe H (2005) Parallel controllable texture synthesis. ACM Trans Graph 24:777–786
25. Lefebvre S, Hoppe H (2006) Appearance-space texture synthesis. ACM Trans Graph 25:541–548
26. Zelinka S, Garland M (2002) Towards real-time texture synthesis with the jump map. In: EGRW '02: Proceedings of the 13th

Eurographics workshop on Rendering, Aire-la-Ville, Eurographics Association, Switzerland, pp 99–104
27. Zelinka S, Garland M (2004) Jump map-based interactive texture synthesis. ACM Trans Graph 23:930–962
28. Liang L, Liu C, Xu YQ, Guo B, Shum HY (2001) Real-time texture synthesis by patch-based sampling. ACM Trans Graph 20:127–150
29. de Castro LN, Zuben FJV (2000) The clonal selection algorithm with engineering applications. In: Proceedings of GECCO'00: Workshop on Artificial Immune Systems and Their Applications, Las Vegas, Nevada, USA, pp 36–39
30. de Castro LN, Timmis J (2002) Immune systems: a new computational intelligence approach. Springer, Berlin
31. de França FO, Zuben FJV, de Castro LN (2005) An artificial immune network for multimodal function optimization on dynamic environments. In: GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, ACM Press, New York, pp 289–296
32. Pan Z, Kang L (1997) An adaptive evolutionary algorithm for numerical optimization. In: Simulated evolution and learning: First Asia-Pacific Conf. (SEAL'96), Selected papers, Springer, Berlin, pp 27–34
33. Pérez P, Gangnet M, Blake A (2003) Poisson image editing. ACM Trans Graph 22:313–318
34. Arya S, Mount DM, Netanyahu NS, Silverman R, Wu AY (1998) An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J ACM 45:891–923
35. Gersho A, Gray RM (1991) Vector quantization and signal compression. Kluwer, Norwell
36. Kilthau SL, Drew MS, Möller T (2002) Full search content independent block matching based on the fast fourier transform. In: Proceedings of international conference on image processing 2002, vol 1. Vancouver, BC, Canada, pp 669–672
37. Soler C, Cani MP, Angelidis A (2002) Hierarchical pattern mapping. ACM Trans Graph 21:673–680
38. Dellaert F, Kwatra V, Oh SM (2005) Mixture trees for modeling and fast conditional sampling with applications in vision and graphics. In: CVPR '05: Proceedings of the 2005 IEEE Computer Society conference on computer vision and pattern recognition (CVPR'05) , vol 1. IEEE Computer Society, Washington, pp 619–624
39. Nicoll A, Meseth J, Müller G, Klein R (2005) Fractional fourier texture masks: guiding near-regular texture synthesis. Comput Graph Forum 24:569–579
40. Boykov Y, Veksler O, Zabih R (2001) Fast approximate energy minimization via graph cuts. IEEE Trans Pattern Anal Mach Intell 23:1222–1239