Published in Image Processing On Line on 2017–01–11.
Submitted on 2016–02–15, accepted on 2016–09–16.
ISSN 2105–1232 © 2017 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2017.171

Efros and Freeman Image Quilting Algorithm for Texture Synthesis

Lara Raad[1], Bruno Galerne[2]

[1] CMLA, ENS Cachan, France (raad@cmla.ens-cachan.fr)
[2] Laboratoire MAP5 (UMR CNRS 8145), Université Paris Descartes, Sorbonne Paris Cité
(bruno.galerne@parisdescartes.fr)

Abstract

Exemplar-based texture synthesis is defined as the process of generating, from an input texture sample, new texture images that are perceptually equivalent to the input. Efros and Freeman's method is a non-parametric patch-based method which computes an output texture image by quilting together patches taken from the input sample. The main innovation of their work relies in the stitching technique which significantly reduces the transition effect between patches. In this paper, we propose a detailed analysis and implementation of their work. We provide a complete mathematical description of the linear programming problem used for the quilting step as well as implementation details. Additionally we propose a partially parallel version of the quilting technique.

Source Code

The source code and an online demonstration of the algorithm described in this article are accessible at the IPOL web page of this article[1].

Keywords: texture synthesis; quilting; patch; linear programming

1 Introduction

Texture synthesis is a classical image processing problem that finds its applications in virtual reality rendering (video games, animation movies, ...). Given an input texture image, it consists in producing output texture images that are both visually similar to and pixel-wise different from the input, and having possibly a larger size. One can separate texture synthesis algorithms into two categories, namely statistical constraint approaches and non-parametric patch-based methods, although "hybrid" algorithms have been proposed recently [15, 13].

Statistical constraint texture synthesis algorithms model the texture based on statistical and/or perceptual considerations. They generally involve two different steps, one for analysis and one for synthesis. The analysis step consists in estimating a set of relevant statistics from the input texture

[1] https://doi.org/10.5201/ipol.2017.171

image. The synthesis step computes an image that satisfies the statistical constraints estimated during the analysis step. Following the seminal paper of Heeger and Bergen [7, 2], several methods are based on statistics of wavelet coefficients or more involved multiscale image representations [12, 11, 14]. Another approach initially proposed by van Wijk [16], which consists in randomizing the Fourier phase of an image, has been extended to an exemplar-based synthesis method in [6, 5]. This method yields excellent results for micro-textures, which are defined as textures that do not present salient geometric patterns and that are not constituted of individual discernible objects.

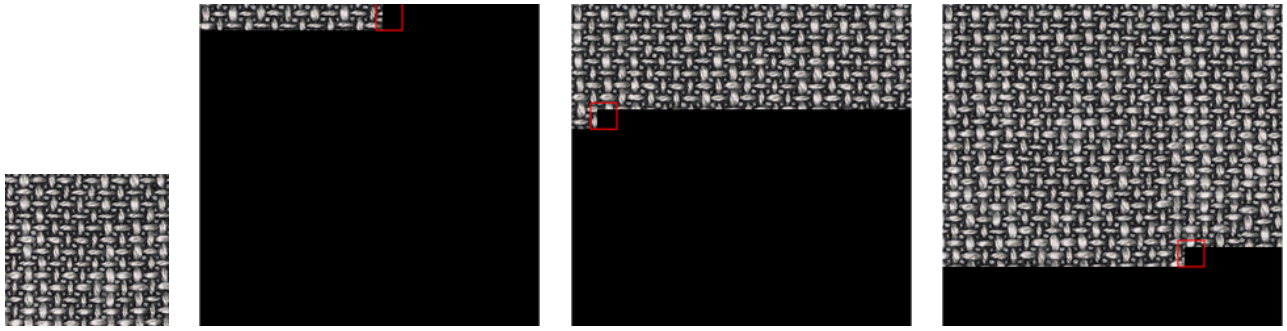
Non-parametric patch-based algorithms attempt at producing a new texture by arranging local neighborhoods of the input texture in a consistent way. The first methods of this kind are sequential algorithms that create a new texture one pixel at a time [4, 18]. These algorithms represented a breakthrough in the field since they are able to reproduce macro-textures with specific geometric structures, see e.g. the IPOL paper [1]. Along with the method of Liang et al.[10] developed at the same period, image quilting [3], the algorithm investigated in this paper, computes the new texture by arranging seamlessly full patches of the input texture. The main innovation of this method is the procedure to stitch a new patch in the sequentially built output texture to avoid discontinuities as much as possible. This is achieved by computing an optimal boundary cut between the patch and the synthesis area thanks to a linear programming optimization (see Section 2 for details). Let us note that another solution of this "stitching step" is proposed in [9] using graphcuts. Many contributions have since improved the results of image quilting, at least regarding the computational cost, and we refer to the state of the art [17] for a more complete survey of this category of texture synthesis algorithms. Let us also mention the recent paper [8] that discusses and attempts to solve some limitations of these approaches.

The plan of the paper is as follows. Section 2 describes in detail all the steps of the algorithm and in particular gives a detailed mathematical description of the linear programming problem for the computation of the minimum error boundary cut. Section 3 gives implementation details and discusses a parallelization of the Efros-Freeman algorithm which, to the best of our knowledge, is a contribution of this paper. Section 4 presents numerous experiments of our implementation of the Efros-Freeman algorithm. This experimental section shows that this algorithm usually produces visually good results. However, a set of failure cases shows that the shortcomings of the Efros-Leung algorithm, that is, garbage growing and verbatim copy [4, 1], are also present in the Efros-Freeman results, but only at a larger scale.

2 Algorithm Description

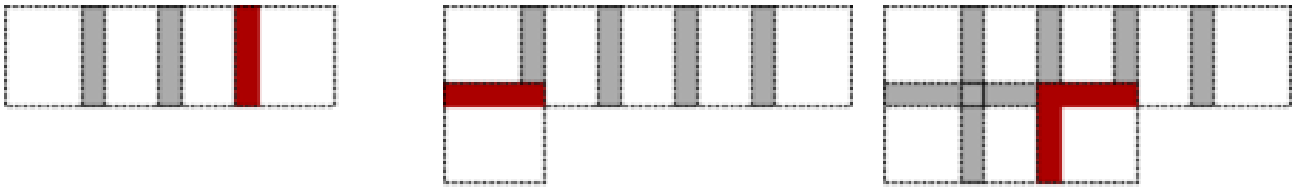
In [3] the authors propose a sequential patch-based algorithm to synthesize textures. I_0 and I_s denote the input sample and the output texture respectively. The output image I_s is constructed patch by patch in a raster scan order. The goal of each iteration is to fill a patch P_{old} of I_s that is only partially defined on a region called overlap region (see Figure 1 and Figure 2). To do so a patch P_{in} of I_0 that matches P_{old} on the overlap region is randomly selected. An optimal boundary cut between P_{old} and P_{in} is then computed within the overlap region. This optimal boundary cut is used to construct the new patch P_{new} by blending P_{old} and P_{in} along the cut (see Figure 3).

The whole image quilting algorithm is described in Algorithm 1 and the remaining of this section will detail each step of the algorithm, namely the initialization, the patch search procedure to select P_{in}, the computation of the minimum error boundary and the blending procedure along the boundary.



input iteration 10 iteration 87 iteration 217

Figure 1: Three different iterations of the synthesis process are shown. At each iteration a patch is being synthesized. This patch is represented by the red square for the three cases.



(a) Vertical overlap (b) Horizontal overlap (c) L-shaped overlap

Figure 2: Three overlap cases arises in the raster scan order: vertical overlap for the first row, horizontal overlap for the first column, and L-shaped overlap everywhere else.

(a) (b) (c) (d)

Figure 3: Quilting a square patch from the input texture into the synthesized texture. (a) sub-part of the synthesized texture. The red dotted zone shows where the new patch will be quilted. (b) patch to quilt in the red dotted zone shown in (a). (c) error surface between the patch in (b) and the red dotted zone in (a). The red path shows the minimum error boundary. (d) the patch in (b) is quilted along the minimum error boundary in the corresponding zone showed in (a).

2.1 Initialization

The first step of the algorithm is to initialize I_s. For that a random patch P_{in} of size $w_p \times w_p$ is taken from I_0 and placed at the top-left corner of I_s.

Algorithm 1 Image Quilting

Input: Sample texture I_0, patch size w_p, overlap size w_o, tolerance parameter ε, output/input size ratio r

Output: Synthesized texture I_s

1: Initialize I_s
2: **for each** patch P_{old} in I_s **do**
3: Select a compatible patch $P_{\text{in}} \in I_0$ using the Patch Selection algorithm (see Algorithm 2)
4: Compute minimum error boundary cut between P_{old} and P_{in} (see Algorithm 3)
5: Construct the patch P_{new} by blending P_{old} and P_{in} along the boundary cut (see Equation (4))
6: Replace P_{old} with P_{new} within I_s
7: **end for**

2.2 Patch Selection

Once the image has been initialized the algorithm synthesizes the remaining patches of I_s sequentially in raster scan order. At each iteration one has to fill a patch P_{old} of I_s that is only defined on an overlap region of width w_o. Note that there are three possible overlap regions: vertical overlap for the first row, horizontal overlap for the first column, and L-shaped overlap everywhere else (see Figure 2).

To select a patch P_{in} of the input image I_0 one computes the square distance between the overlap region of the patch P_{old} of I_s and the corresponding regions of all the patches of I_0. The minimal distance $d_{\min}$ is determined and P_{in} is randomly picked among all patches whose distance to P_{old} is lower than $(1+\varepsilon)d_{\min}$ where ε is the tolerance parameter.

To conclude this section let us give a detailed expression of the distance used to compare patches. A patch of I_0 is represented by the position of its top-left corner $(m,n) \in \{0,\dots,M_0-w_p\} \times \{0,\dots,N_0-w_p\}$. The squared distance image D contains at each position (m,n) the distance between P_{old} and the patch from I_0 whose top-left corner is (m,n) according to some binary weight Q that equals one in the overlap region and zero otherwise. More precisely for all $(m,n) \in \{0,\dots,M_0-w_p\} \times \{0,\dots,N_0-w_P\}$, one has

$$D(m,n) = \sum_{i,j=0}^{w_p-1} Q(i,j)(P_{\text{old}}(i,j) - I_0(m+i,n+j))^2. \tag{1}$$

The computation of the squared distance image D is discussed in Section 3.1 and the whole patch selection procedure is summarized in Algorithm 2.

Algorithm 2 Patch Selection

Input: Sample texture I_0, patch under construction P_{old}, patch size w_p, tolerance $\varepsilon > 0$, binary weight Q defining the overlap region

Output: Patch position (m,n)

1: Compute the squared distance image D (see Equation (1)) containing the distances between the patch P_{old} and all patches of I_0.
2: $d_{\min} \leftarrow \min_{(k,l)} D(k,l)$.
3: Uniformly draw a patch position (m,n) among the set $\{(k,l),\ D(k,l) < (1+\varepsilon)d_{\min}\}$.

2.3 Minimum Error Boundary Cut

This step of the algorithm is the main contribution of the Efros-Freeman algorithm [3].

The Patch Search step (see Section 2.2) gives a patch P_{in} of I_0 having coordinates (m, n) that is similar to the partially defined current patch P_{old} in their overlap region. We recall that the overlap regions that arise in the raster scan order are of three types: vertical, horizontal, and L-shaped overlap. These three cases are illustrated in Figure 2. To get the final patch P one must combine the patches P_{old} and P_{in}. Denoting Q the binary weight for the overlap regions as in the previous section, then, for any binary image M such that $0 \leq M(i,j) \leq Q(i,j),\ (i,j) \in \{1,\dots,w_p\}^2$, P can be defined as the combination

$$P_{\text{new}} = MP_{\text{old}} + (1 - M)\, P_{\text{in}}.$$

The main idea of Efros and Freeman [3] is looking for a binary shape M where the transition between P_{old} and P_{new} along the boundary of the shape is minimal. For simplicity, and to be able to use linear programming, the authors do not allow for any shape, but only for the ones whose boundaries are simple forward paths from one end to the other of the overlap region. This results in two pieces of image that are sewn together along some general boundary path, hence the algorithm name "quilting". In the remaining of this section, we describe in detail the computation of the minimum error boundary cut for the three overlap cases, starting with the vertical and horizontal cases.

2.3.1 Connected Paths and Boundary Error

In the following, we call a *path* of length $\ell \geq 1$ in $\{0,\dots,w_p-1\}^2$ any ordered sequence $\gamma = (\gamma_0, \gamma_1, \dots, \gamma_{\ell-1})$ of 8-connected pixels $\gamma_k \in \{0,\dots,w_p-1\}^2$, that is, for all $k \in \{0,\dots,\ell-2\}$, $\max(|\gamma^1_{k+1} - \gamma^1_k|, |\gamma^2_{k+1} - \gamma^2_k|) = 1$ (for convenience, within this section the first and second coordinates of a pixel γ_k are denoted by γ^1_k and γ^2_k).

Let us recall that we use matrix coordinates for the pixels. Hence a path γ of length ℓ is said to be *vertical* if for all $k \in \{0,\dots,\ell-2\}$, $\gamma^1_{k+1} - \gamma^1_k = -1$ (vertical paths are oriented from bottom to top), and *horizontal* if for all $k \in \{0,\dots,\ell-2\}$, $\gamma^2_{k+1} - \gamma^2_k = -1$ (horizontal paths are oriented from right to left).

Denoting by $e(i,j) = (P_{\text{new}}(i,j) - P_{\text{old}}(i,j))^2$ the squared difference between the two patches P_{old} and P_{new}, the boundary error of a path γ of length ℓ is defined by

$$\mathcal{E}(\gamma) = \sum_{k=0}^{\ell-1} e(\gamma_k).$$

2.3.2 Vertical Boundary Cuts

In this section we explain how the optimal boundary between P_{old} and P_{new} is defined and computed in the case of vertical overlap, that is when the overlap region is the rectangle $\{0,\dots,w_p-1\} \times \{0,\dots,w_o-1\}$.

The optimal boundary is defined as the vertical path γ that minimizes the boundary error $\mathcal{E}(\gamma)$ while joining both ends of the overlap regions. More precisely, define the admissible vertical paths as

$$\Gamma_v = \{\gamma = (\gamma_0,\dots,\gamma_{w_p-1}),\ \forall k,\ \gamma^1_k = w_p - 1 - k \text{ and } \gamma^2_k \in \{0,\dots,w_o-1\}\},$$

(one notices that paths of $\gamma \in \Gamma_v$ are indeed vertical since $\gamma^1_{k+1} - \gamma^1_k = -1$). Then an optimal boundary is defined as any solution of the optimization problem

$$\min_{\gamma \in \Gamma_v} \mathcal{E}(\gamma). \tag{2}$$

Problem (2) is solved using dynamic programming. This is possible because Problem (2) verifies the principle of optimality: if $(\gamma_0,\dots,\gamma_{w_p-1})$ is an optimal solution of Problem (2), then, for all

$0 \leq r \leq w_p - 1$, the subpath $(\gamma_0, \dots, \gamma_{r-1})$ is an optimal vertical path to reach the r-th point γ_{r-1} starting from the bottom of the overlap region.

Let us now discuss in detail the dynamic programming method to solve Problem (2). For all points $(i,j) \in \{0,\dots,w_p-1\} \times \{0,\dots,w_o-1\}$, denote by $\Gamma_v^{(i,j)}$ the set of vertical paths γ that start at the bottom side of the overlap region and end at the point (i,j), that is

$$\Gamma_v^{(i,j)} = \{\gamma = (\gamma_0, \dots, \gamma_{w_p-1-i}),\ \forall k,\ \gamma_k^1 = w_p - 1 - k,\ \gamma_k^2 \in \{0,\dots,w_o-1\},\ \text{and } \gamma_{w_p-1-i} = (i,j)\}.$$

Now, for all points $(i,j) \in \{0,\dots,w_p-1\} \times \{0,\dots,w_o-1\}$, define $E_v(i,j)$ as the minimal cumulative vertical error to reach (i,j) starting from the bottom side, that is,

$$E_v(i,j) = \min_{\gamma \in \Gamma_v^{(i,j)}} \mathcal{E}(\gamma).$$

Then, since

$$\Gamma_v = \bigcup_{j \in \{0,\dots,w_o-1\}} \Gamma_v^{(0,j)},$$

one has

$$\min_{\gamma \in \Gamma_v} \mathcal{E}(\gamma) = \min_{j \in \{0,\dots,w_o-1\}} E_v(0,j).$$

Now, remark that the last-but-one point of an optimal vertical path that ends at (i,j) is one of the (at most) three points below (i,j), namely the points $(i+1,j-1)$, $(i+1,j)$, $(i+1,j+1)$ (there can be only two neighboring points if (i,j) is at the border, that is, if $j=0$ or $j=w_o-1$). Hence, for all $i \in \{0,\dots,w_p-2\}$, one has

$$E_v(i,j) = e(i,j) + \min(E_v(i+1,j-1), E_v(i+1,j), E_v(i+1,j+1)), \tag{3}$$

where by convention, if one of the index $j-1$ or $j+1$ is not a valid index, the corresponding term $E_v(i+1,j-1)$ or $E_v(i+1,j+1)$ is discarded from the minimum. Besides, for the last line $i = w_p - 1$, one simply has $E_v(w_p-1,j) = e(w_p-1,j)$ since the paths are only made of one pixel. Hence the dynamic programming procedure for solving Problem (2) is to compute the costs $E_v(i,j)$ line by line from bottom to top using Equation (3) recursively, and then search for the minimal value of the first line $j^* = \operatorname{argmin}_j E_v(0,j)$. The full optimal path γ can then be traced back starting from this coordinate $(0,j^*) = (\gamma_{w_p-1}^1, \gamma_{w_p-1}^2)$. More precisely, if (i,j) are the coordinates of the point γ_{w_p-1-i} of the optimal path γ, then its preceding point $\gamma_{w_p-1-(i+1)}$ is given by

$$\gamma_{w_p-1-(i+1)} = \operatorname{argmin}(E_v(i+1,j-1), E_v(i+1,j), E_v(i+1,j+1)).$$

In practice one stores the matrix $T_v(i,j) = \operatorname{argmin}(E_v(i+1,j-1), E_v(i+1,j), E_v(i+1,j+1))$ while computing the vertical cumulative error E_v given by Equation (3) in order to trace back the path without additional computation since for $i = w_p - 2$ to 0, $\gamma_i = T_v(\gamma_{i+1})$.

2.3.3 Horizontal Boundary Cuts

The case of horizontal overlap is just the symmetric case of vertical overlap one. Still let us introduce the notation that will be necessary for dealing with the L-shaped overlap case. One defines the set of horizontal paths as

$$\Gamma_h = \{\gamma = (\gamma_0, \dots, \gamma_{w_p-1}),\ \forall k,\ \gamma_k^2 = w_p - 1 - k \text{ and } \gamma_k^1 \in \{0,\dots,w_o-1\}\},$$

and for all points $(i,j) \in \{0,\dots,w_o-1\} \times \{0,\dots,w_p-1\}$, one defines

$$\Gamma_h^{(i,j)} = \{\gamma = (\gamma_0, \dots, \gamma_{w_p-1-j}),\ \forall k,\ \gamma_k^2 = w_p - 1 - k,\ \gamma_k^1 \in \{0,\dots,w_o-1\},\ \text{and } \gamma_{w_p-1-j} = (i,j)\},$$

the set of paths that start at the right side of the overlap region and end at (i, j), as well as

$$E_h(i,j) = \min_{\gamma \in \Gamma_h^{(i,j)}} \mathcal{E}(\gamma),$$

the minimal cumulative horizontal error to reach (i, j) starting from the right side of the overlap region. Then $E_h(i, w_p - 1) = e(i, w_p - 1)$, and for all $j \in \{0, \dots, w_p - 2\}$, one has the recursive relation

$$E_h(i,j) = \min(E_h(i-1, j+1), E_h(i, j+1), E_h(i+1, j+1)),$$

which enables to compute $E_h(i, j)$ for all $(i, j) \in \{0, \dots, w_o - 1\} \times \{0, \dots, w_p - 1\}$. Since

$$\min_{\gamma \in \Gamma_h} \mathcal{E}(\gamma) = \min_{i \in \{0,\dots,w_o-1\}} E_h(i, 0)$$

the end point of the optimal horizontal path is $(i^*, 0)$ where $i^* = \operatorname{argmin}_i E_h(i, 0)$ and the path can be traced back thanks to the matrix $T_h(i, j) = \operatorname{argmin}(E_h(i-1, j+1), E_h(i, j+1), E_h(i+1, j+1))$.

2.3.4 L-shaped Boundary Cuts

The case of L-shaped overlap regions is slightly more complex than the vertical and horizontal cases since the geometry of the L-shape does not allow for a clear ordering of the pixels of the path. The original paper [3] only mentions that, in the L-shaped overlap case, "the minimal paths meet in the middle and the overall minimum is chosen for the cut". We propose below a rigorous interpretation of this sentence.

A separating path will start at the bottom side of the L-shape (that is, the points (i, j) with $i = w_p - 1$ and $j \in \{0, \dots, w_o - 1\}$) and end at the right side of the L-shaped (that is, the points (i, j) with $i \in \{0, \dots, w_o - 1\}$ and $j = w_p - 1$). To restrict the geometry of admissible paths (and allowing for dynamic programming), it is further required that an L-shaped path has to meet at some diagonal point (i, i), $i \in \{0, \dots, w_o - 1\}$, that the first part from the bottom side to this diagonal point (i, i) is a vertical path, and that the remaining part from the diagonal point (i, i) to the right side is a reversed horizontal path. More formally, for all indexes $i \in \{0, \dots, w_o - 1\}$, one defines

$$\Gamma_L^i = \{\gamma = (\gamma_0, \dots, \gamma_{2(w_p-i)-1}),\ (\gamma_0, \dots, \gamma_{w_p-i-1}) \in \Gamma_v^{(i,i)} \text{ and } (\gamma_{2(w_p-i)-1}, \dots, \gamma_{w_p-i-1}) \in \Gamma_h^{(i,i)}\},$$

and for all $\gamma \in \Gamma_L^i$ we denote by γ^v and γ^h its associated vertical and horizontal paths of $\Gamma_v^{(i,i)}$ and $\Gamma_h^{(i,i)}$ respectively. The set of the admissible L-shaped boundaries Γ_L is then defined as the disjoint union of the sets Γ_L^i, that is,

$$\Gamma_L = \bigcup_{i=0}^{w_0-1} \Gamma_L^i.$$

As before, we search for an optimal L-shaped boundary cut $\gamma \in \Gamma_L$ having minimal boundary error

$$\mathcal{E}(\gamma) = \sum_{k=0}^{\ell(\gamma)-1} e(\gamma_k),$$

where $\ell(\gamma)$ is the length of the path γ (which is equal to $2(w_p - i) - 1$ if $\gamma \in \Gamma_L^i$). The optimal L-shaped boundary can be found in splitting the above sum into a vertical part and an horizontal part, since for all $\gamma \in \Gamma_L^i$ one has

$$\mathcal{E}(\gamma) = \mathcal{E}(\gamma^v) + \mathcal{E}(\gamma^h) - e(i, i).$$

Hence one has

$$\begin{aligned}\min_{\gamma\in\Gamma_L}\mathcal{E}(\gamma) &= \min_{i\in\{0,\dots,w_o-1\}}\min_{\gamma\in\Gamma_L^i}\mathcal{E}(\gamma)\\ &= \min_{i\in\{0,\dots,w_o-1\}}\min_{\gamma\in\Gamma_L^i}\mathcal{E}(\gamma^h)+\mathcal{E}(\gamma^v)-e(i,i)\\ &= \min_{i\in\{0,\dots,w_o-1\}}\left[\left(\min_{\gamma^v\in\Gamma_v^{(i,i)}}\mathcal{E}(\gamma^v)\right)+\left(\min_{\gamma^h\in\Gamma_h^{(i,i)}}\mathcal{E}(\gamma^h)\right)-e(i,i)\right]\\ &= \min_{i\in\{0,\dots,w_o-1\}}E_v(i,i)+E_h(i,i)-e(i,i),\end{aligned}$$

where E_v and E_h are the vertical and horizontal cumulative errors defined in the previous sections. Hence,

$$\min_{\gamma\in\Gamma_L}\mathcal{E}(\gamma)=\min_{i\in\{0,\dots,w_o-1\}}E_v(i,i)+E_h(i,i)-e(i,i).$$

The above equation enables us to determine the optimal position (i^*,i^*) on the diagonal of the optimal path γ once the matrices E_v and E_h have been computed recursively. We can then trace back the vertical part γ^v and the horizontal part γ^h of the optimal path γ using the matrices T_v and T_h. Let us remark that the "overall minimum" evoked in [3] must not be interpreted as the minimum of $E_v(i,i)+E_h(i,i)$ but the minimum of $E_v(i,i)+E_h(i,i)-e(i,i)$.

2.4 Blending along the Cut

This is the last step of an iteration. Its goal is to construct the new patch P_{new} by blending P_{old} and P_{in} using the previously computed boundary cut. The boundary cut defines a binary mask M that equals one on the left and/or top of the cut and zero otherwise. The patch P_{new} can be defined as

$$P_{\text{new}}=MP_{\text{old}}+(1-M)\,P_{\text{in}}. \tag{4}$$

We noticed that the Matlab implementation of the authors proposes an optional smoothing of the binary mask presumably to avoid noticeable transitions along the cut. We implemented this option but we did not observe noticeable improvements. Since it is not discussed in the original paper we do not use it for the experimental results of this paper.

3 Implementation

3.1 Computing Patch Distances with FFT

In this section we explicit an algorithm to compute the squared distance between a patch P_{old} and all the patches of the input texture I_0 using the Fast Fourier Transform (FFT), that is, to compute the squared distance image D of Equation (1) involved in the patch search algorithm (see Algorithm 2). To the best of our knowledge, regarding texture synthesis this acceleration was first discussed by Kwatra et al. [9]. Recall that we consider a patch P_{old} of size $w_p\times w_p$ and that I_0 is of size $M_0\times N_0$. Hence the naive summation to compute

$$D(m,n)=\sum_{i,j=0}^{w_p-1}Q(i,j)(P_{\text{old}}(i,j)-I_0(m+i,n+j))^2,$$

for all $(m,n)\in\{0,\dots,M_0-w_p\}\times\{0,\dots,N_0-w_p\}$ requires around $w_p^2M_0N_0$ operations with $w_p=40$ pixels as a typical value. An important asset of the proposed FFT-based implementation is that the

Algorithm 3 Minimum Error Boundary Cut

Input: Output texture I_s, patch P_{old} (from I_s), patch P_{in} (from I_0), patch size w_p, overlap size w_o
Output: Boundary cut γ

Compute the squared difference $e(i,j) = (P_{\text{in}}(i,j) - P_{\text{old}}(i,j))^2$
switch (overlap type)
case vertical**:**
  Compute the minimal cumulative vertical error E_v:
  $E_v(w_p - 1, j) = e(w_p - 1, j), \quad j \in \{0, \dots, w_o - 1\}$
  for $i = w_p - 2$ to 0 **do**
    $E_v(i,j) = e(i,j) + \min(E_v(i+1, j-1), E_v(i+1, j), E_v(i+1, j+1)), \quad j \in \{0, \dots, w_o - 1\}$
    $T_v(i,j) = \text{argmin}(E_v(i+1, j-1), E_v(i+1, j), E_v(i+1, j+1)), \quad j \in \{0, \dots, w_o - 1\}$
  end for
  Determine $j^* = \text{argmin}_j E_v(0, j)$
  Trace back the path γ starting at $\gamma_{w_p - 1} = (0, j^*)$ using T_v: for $i = w_p - 2$ to 0, $\gamma_i = T_v(\gamma_{i+1})$
case horizontal**:**
  Compute the minimal cumulative horizontal error E_h:
  $E_h(i, w_p - 1) = e(i, w_p - 1), \quad i \in \{0, \dots, w_o - 1\}$
  for $j = w_p - 2$ to 0 **do**
    $E_h(i,j) = e(i,j) + \min(E_h(i-1, j+1), E_h(i, j+1), E_h(i+1, j+1)), \quad i \in \{0, \dots, w_o - 1\}$
    $T_h(i,j) = \text{argmin}(E_h(i-1, j+1), E_h(i, j+1), E_h(i+1, j+1)), \quad i \in \{0, \dots, w_o - 1\}$
  end for
  Determine $i^* = \text{argmin}_i E_h(i, 0)$
  Trace back the path γ starting at $\gamma_{w_p - 1} = (i^*, 0)$ using T_h: for $j = w_p - 2$ to 0, $\gamma_j = T_h(\gamma_{j+1})$
case L-shaped**:**
  Compute the minimal cumulative vertical error E_v as in the vertical case
  Compute the minimal cumulative horizontal error E_h as in the horizontal case
  Determine $i^* = \text{argmin}_i(E_v(i,i) + E_h(i,i) - e(i,i))$ (with $i \in \{0, \dots, w_o - 1\}$)
  Trace the vertical part of γ starting at (i^*, i^*) using T_v: for $i = w_p - i^* - 2$ to 0, $\gamma_i = T_v(\gamma_{i+1})$
  Trace the horizontal part of γ starting at (i^*, i^*) using T_h: for $j = w_p - i^*$ to $2(w_p - i^*) - 1$, $\gamma_j = T_h(\gamma_{j-1})$
end switch

computational cost is limited to two FFT calls for images of size $M_0 \times N_0$ and is thus independent of the patch width w_p.

Let us first recall some notation. A patch of size $w_p \times w_p$ within an image is represented by its top-left corner, hence all admissible patches of I_0 have (m,n)-coordinates in the set $\{0, \dots, M_0 - w_p\} \times \{0, \dots, N_0 - w_p\}$.

Discrete Fourier Transform. Given any image $V \in \mathbb{R}^{M_0 \times N_0}$ we denote by $\mathcal{F}(V) = \hat{V}$ the discrete Fourier transform of V and $\mathcal{F}^{-1}(V) = \check{V}$ the inverse discrete Fourier transform of V, defined by

$$\hat{V}(k,l) = \frac{1}{M_0 N_0} \sum_{m=0}^{M_0-1} \sum_{n=0}^{N_0-1} V(m,n) e^{-2i\pi\left(\frac{km}{M_0} + \frac{nl}{N_0}\right)} \quad \text{and} \quad \check{V}(k,l) = \sum_{m=0}^{M_0-1} \sum_{n=0}^{N_0-1} V(m,n) e^{2i\pi\left(\frac{km}{M_0} + \frac{nl}{N_0}\right)}.$$

Convolution Product. Denote by $V * W$ the convolution product between V and W, that is,

$$V * W(m,n) = \sum_{k=0}^{M_0-1} \sum_{l=0}^{N_0-1} V(k,l) W(m-k, n-l),$$

where the indexes $(m-k, n-l)$ are understood modulo (M_0, N_0). With these conventions for the DFT, one has $\mathcal{F}(V * W) = M_0 N_0 \hat{V}\hat{W}$, where the multiplication between images is the component-wise product. However, recall that the FFTW library computes the Fourier transform without normalization. Hence the operators that are computed are respectively $M_0 N_0 \mathcal{F}$ for the forward transform and $\mathcal{F}^{-1}$ for the backward transform. Hence we have

$$V * W = \mathcal{F}^{-1}(\mathcal{F}(V * W)) = \frac{1}{M_0 N_0} \mathcal{F}^{-1}(M_0 N_0 \mathcal{F}(V) M_0 N_0 \mathcal{F}(W)).$$

This means that after multiplying the two FFTW forward transforms and performing the backward inverse transform, one has to normalize dividing by the size of the images.

Cross-correlation. Let us denote by $\Gamma(V, W)$ the cross-correlation between two images,

$$\Gamma(V,W)(m,n) = \sum_{k=0}^{M_0-1} \sum_{l=0}^{N_0-1} V(k,l) W(m+k, n+l).$$

Note also $\tilde{V}$ the symmetric of V with respect to the origin, that is, $\tilde{V}(m,n) = V(-m,-n)$. Note that one has $\Gamma(W,V)(m,n) = \Gamma(V,W)(-m,-n)$, that is, $\Gamma(W,V) = \widetilde{\Gamma(V,W)}$ and that the cross-correlation is simply a convolution between the symmetric version of the first image and the second image, that is,

$$\Gamma(V,W)(m,n) = \tilde{V} * W(m,n) = V * \tilde{W}(-m,-n).$$

Hence the computational cost for a cross-correlation image is the same that for a convolution product, that is three FFT calls.

We need to compute the squared distance between a patch P_{old} and all the patches of I_0 according to some binary weight Q that represents the overlap region (that is a horizontal, vertical, or L-shaped mask). More precisely for all $(m,n) \in \{0, \dots, M_0 - w_p\} \times \{0, \dots, N_0 - w_p\}$, we want to compute

$$D(m,n) = \sum_{i,j=0}^{w_p-1} Q(i,j)(P_{\text{old}}(i,j) - I_0(m+i, n+j))^2.$$

Let us denote by P_{ext} and Q_{ext} the extensions of P_{old} and Q into images of size $M_0 \times N_0$ by filling the domain with 0-valued pixels, that is

$$P_{\text{ext}}(m,n) = \begin{cases} P_{\text{old}}(m,n) & \text{if } (m,n) \in \{0, \dots, w_p - 1\} \times \{0, \dots, w_p - 1\}, \\ 0 & \text{otherwise,} \end{cases}$$

and similarly for Q_{ext}. Then one has

$$\begin{aligned} D(m,n) &= \sum_{i,j=0}^{w_p-1} Q(i,j)(P_{\text{old}}(i,j) - I_0(m+i,n+j))^2 \\ &= \sum_{k=0}^{M_0-1}\sum_{l=0}^{N_0-1} Q_{\text{ext}}(k,l)(P_{\text{ext}}(k,l) - I_0(m+k,n+l))^2 \\ &= \sum_{k=0}^{M_0-1}\sum_{l=0}^{N_0-1} \big(Q_{\text{ext}}(k,l)P_{\text{ext}}(k,l)^2 - 2Q_{\text{ext}}(k,l)P_{\text{ext}}(k,l)I_0(m+k,n+l) + Q_{\text{ext}}(k,l)I_0(m+k,n+l)^2\big) \\ &= \sum_{i,j=0}^{w_p-1} Q(i,j)P_{\text{old}}(i,j)^2 - 2\Gamma(Q_{\text{ext}}P_{\text{ext}}, I_0)(m,n) + \Gamma(Q_{\text{ext}}, I_0^2)(m,n), \end{aligned}$$

where the multiplications $Q_{\text{ext}}P_{\text{ext}}$ and I_0^2 are component-wise.

Within our procedure, we will compare a patch P_{old} that is taken from the output image that is only partially defined, and the binary weight Q representing the overlap region. Hence, if all the undefined pixels of the output patch P_{old} are set to 0 (by initializing I_s to 0), one has

$$\forall (i,j) \in \{0, w_p - 1\}^2, \quad Q(i,j)P_{\text{old}}(i,j) = P_{\text{old}}(i,j).$$

Consequently,

$$\sum_{i,j=0}^{w_p-1} Q(i,j)P_{\text{old}}(i,j)^2 = \sum_{i,j=0}^{w_p-1} P_{\text{old}}(i,j)^2 = \|P_{\text{old}}\|_2^2,$$

and $Q_{\text{ext}}P_{\text{ext}} = P_{\text{ext}}$, and thus $\Gamma(Q_{\text{ext}}P_{\text{ext}}, I_0) = \Gamma(P_{\text{ext}}, I_0)$. Hence, the binary mask Q is only influential when computing $\Gamma(Q_{\text{ext}}, I_0^2)$, a computation that is done just once at the beginning of the procedure since it does not depend on P_{old}.

In the end, the image D is simply given by

$$D = \|P_{\text{old}}\|_2^2 - 2\Gamma(P_{\text{ext}}, I_0) + \Gamma(Q_{\text{ext}}, I_0^2), \tag{5}$$

where the last image $\Gamma(Q_{\text{ext}}, I_0^2)$ is computed once before running the algorithm and stored in memory. Hence the cost for computing D is only three FFT calls for the computation of the cross-correlation $\Gamma(P_{\text{ext}}, I_0)$, and is even reduced to two FFT calls by storing the DFT of I_0.

Note that the image D is defined for all $(m,n) \in \{0, \dots, M_0 - 1\} \times \{0, \dots, N_0 - 1\}$, but the squared distances for (m,n)-coordinates outside $\{0, \dots, M_0 - w_p\} \times \{0, \dots, N_0 - w_p\}$ correspond to patches of size $w_p \times w_p$ that are defined by periodic boundary conditions and thus those patches must be discarded when searching for the minimal distance.

Minimum distance. Theoretically the results of the distance computation using the sum square differences or the FFT are equal. But in practice, the computation using the FFT is subject to rounding errors. Especially if there is a patch such that $D(m,n) = 0$, when adding positive and negative terms in Equation (5) the result might be negative due to numerical errors. This rounding error problem leads to a negative minimal squared distance $d_{\min}$, which leads to errors in the patch search algorithm (Algorithm 2) since then none of patches satisfy the condition $D(m,n) < (1+\varepsilon)d_{\min}$. To avoid this, all squared distances $D(m,n)$ smaller than 1 are set to $D(m,n) = 1$ (the value 1 is smallest distance-value greater than 0).

3.2 Parallelization

Although the Efros-Freeman algorithm was presented as a purely sequential algorithm, we found out that the procedure can be partially parallelized, and thus the algorithm significantly accelerated when running on multi-core platforms.

In general, non-parametric patch based methods can not be completely parallelized because the sequential assignment of patches is strongly dependent of the previous synthesis steps. Nevertheless, in the case of the quilting method, a new patch assignment is only correlated to some previous synthesis steps and not to all of them. To illustrate this let us consider the first few steps of the algorithm. The first step of the algorithm adds the seed patch and the second step adds a second patch on the right. When adding the third patch, the first patch of the second row can be created at the same time since it only depends on the one that is on top of it and on its top-right. More generally the synthesis of a new row of patches can be started as soon as two patches of the previous row have been synthesized. This procedure is illustrated in Figure 4.

Let us discuss the acceleration induced by the parallelization. Suppose we synthesize an image I_s made of $N_r \times N_c$ patches where N_r is the number patch rows and N_c is the number of patch columns. Without the parallelization the number of iterations N_{it} is $N_{\text{it}} = N_r N_c$. When parallelizing the number of iterations N_{it} is reduced to $2(N_r - 1) + N_c$. This parallelization is especially effective if one has as many processors available as patches to synthesize simultaneously in one iteration. The maximal number of patches to synthesize simultaneously is equal to $\min\left(N_r, \left\lfloor \frac{N_c+1}{2} \right\rfloor\right)$, where $\lfloor x \rfloor = \max(n \in \mathbb{N}, n \leq x)$, since within an iteration there is at most one new patch per row and one new patch per pair of columns.

The parallel version of the quilting algorithm is summarized in Algorithm 4. Let us notice that for this parallelization to be valid the overlap size must satisfy $w_o \leq w_p/2$ which is always the case in practice.

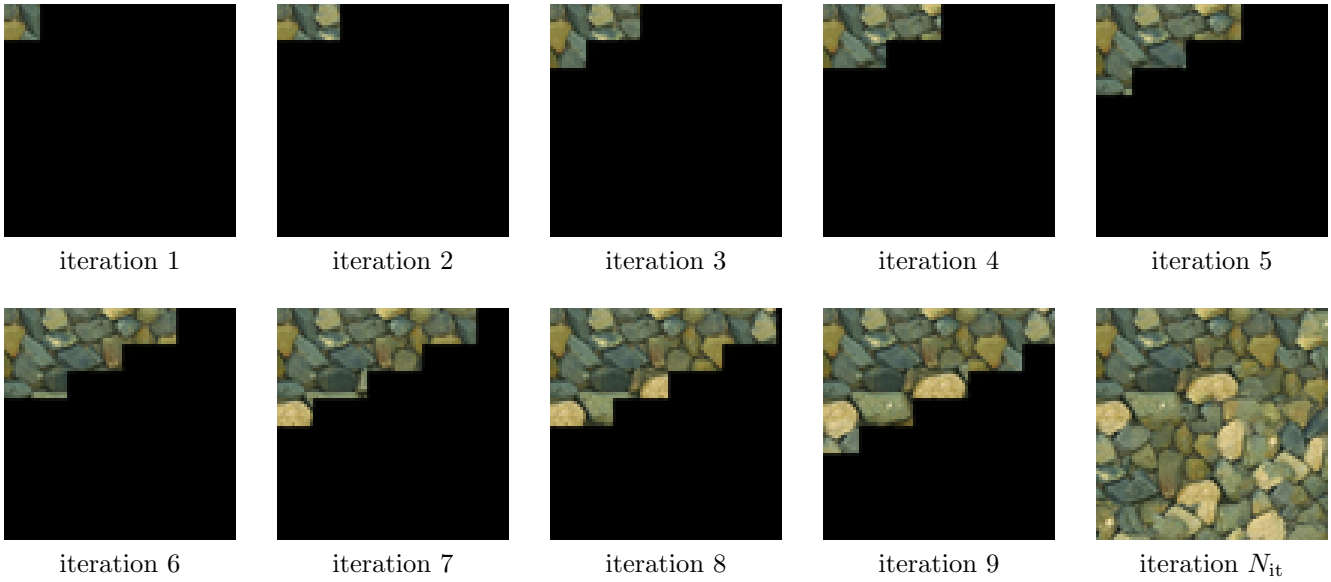


Figure 4: Evolution of the synthesized image with the parallelization. New patches from two subsequent iterations are added simultaneously.

Algorithm 4 Parallelization

Input: Sample texture I_0, N_r is the number patch rows, N_c is the number of patch columns
Output: Synthesized texture I_s

1: $N_{\text{it}} \leftarrow 2(N_r - 1) + N_c$
2: Initialize I_s
3: **for** $k = 1$ **to** $N_{\text{it}} - 1$ **do**
4: **for** $i = \max(0, \lfloor \frac{k-N_c+1}{2} \rfloor)$ **to** $\min(N_r - 1, \lfloor \frac{k}{2} \rfloor)$ **do**
5: $I_s \leftarrow$ synthesize a new patch at position $\big(i(w_p - w_o), (k - 2i)(w_p - w_o)\big)$
6: **end for**
7: **end for**

This loop is run in parallel

4 Experiments

In this section texture synthesis results are shown using the algorithm described previously [3]. Two aspects of this method are emphasized. On the one hand the visual quality of the result and on the other hand the tendency of the method to verbatim copy parts of the input sample. For this we represent each result as shown in Figure 5: the input sample I_0, the color map, the synthesized texture I_s and the synthesis map. The color map is an image of size equal to the size of I_0 and each of its pixels is assigned a different color. This allows to visualize each position of I_0 with the corresponding color of the color map. The synthesis map shows for each synthesized patch its initial position in the texture sample. It allows to identify exactly the verbatim copy regions which correspond to the continuous color areas of the synthesis map. Each pixel $p_i^{'}$ in I_s is assigned a pixel p_j from I_0. The synthesis map at position $p_i^{'}$ is mapped to the value of the color map at position p_j.

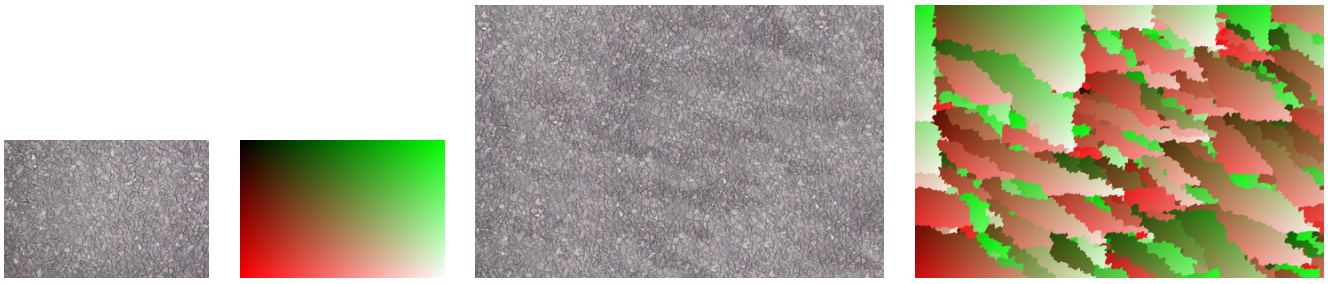


Figure 5: Results representation. From left to right: texture sample, position map, synthesized image and synthesis map. The synthesis map shows for each synthesized patch its position in the texture sample. It allows then to identify exactly the verbatim copy regions (continuous color areas of the map).

Let us discuss the influence of the parameters. There are three of them: the patch size w_p, the overlap size w_o and the tolerance parameter ε. The overlap size is expressed as the proportion with respect to the patch size. That is $w_o = 0.25$ implicitly means $w_o = 0.25 w_p$.

In Figures 6 and 7 successful results are shown for different types of textures. For each example the patch size w_p is adapted taking one of the following values $\{10, 20, 40, 80\}$. The remaining parameters are fixed to $w_o = 0.25$ and $\varepsilon = 0.1$. One can notice that for some texture samples, for example the brick wall in Figure 6 and the texture samples of the last row in Figure 7, even though the results are visually satisfying they are made of large parts of the input sample.

In Figure 8 three failure cases are shown. The first one is due to the patch size. When this is too small, in particular for macro textures, the algorithm fails to recover the details of the different scales as can be seen in the first row of Figure 8. The second failure is the verbatim copy effect. For some texture samples the verbatim copy zones are visually noticeable and unnatural. This is

illustrated in the second row of Figure 8. The last failure case is the "growing garbage" drawback, which occurs when the method is stuck in a region of the input sample repeating it on a large part of the output. This is due to the local aspect of the algorithm and it is more noticeable when the texture examples are not stationary. This is also shown in Figure 8 in the last row. It is important to notice that the two last failure cases are more noticeable when the output size - input size ratio is higher than two.

Influence of the patch size

First of all the patch size influence is analyzed and these results are shown in Figure 9. For this, $w_o = 0.25$ and $\varepsilon = 0.1$ are fixed and the synthesis results for $w_p \in \{10, 20, 40, 80\}$ are compared. This parameter clearly depends on the input sample. In general, macro textures have details at different scales. If the patch size chosen is not able to capture them all then the synthesis fails. A second observation is that the larger the patch size is the larger the verbatim copy zones are. For smaller patches this effect is reduced, since more "similar" patches are available in the patch search step and then the set of candidate patches is larger, allowing more variation.

Influence of the overlap size

The overlap size is an important parameter in this method. To analyze its influence $w_p = 40$ and $\varepsilon = 0.1$ are fixed and the results for $w_o \in \{0.10, 0.25, 0.5\}$ are compared. The general conclusion is that larger overlap sizes often give a seamless transition between patches but are more prone to growing garbage for some texture samples as can be seen in the two first examples in Figure 10 for $w_o = 0.50$. On the other hand if w_o is low the set of candidate patches is bigger and thus reduces the verbatim copy effect at the cost of decreasing the visual quality of the results as can be seen in the third example in Figure 10.

Influence of the tolerance parameter

This last parameter is directly related to the verbatim copy effect. For this analysis $w_p = 40$ and $w_o = 0.25$ are fixed and the results for $\varepsilon \in \{0.05, 0.1, 0.3, 0.5, 0.7\}$ are compared. Increasing ε implies having more candidate patches for the synthesis. This allows more variation when choosing a patch in the image and this is directly seen in the synthesis maps of the examples in Figure 11 where for $\varepsilon \in \{0.5, 0.7\}$ the patches are taken more "randomly" from the input sample thus reducing the size of the verbatim copy regions. For some texture examples the visual quality of the result can decrease. On the other hand, as expected, low values of ε lead to very large verbatim copy areas.

5 Conclusion

In this paper we analyzed in detail Efros and Freeman's texture synthesis algorithm [3]. Extensive numerical experiments have been proposed to illustrate the performance of the method as well as the influence of its parameters. We conclude from these experiments that, in general, for the correct set of parameters, the visual results are satisfying at the cost of verbatim copying large parts of the input textures that might be visually disturbing. This is a common drawback of Efros and Leung's method [4]. Another issue that arises in [3] is the garbage growing effect. This is especially apparent when the input texture is not stationary. We also noticed that it is related to the raster scan order used to synthesize the image, which propagates the errors. All these drawbacks are in general more apparent when synthesizing an image significantly larger than the input. Hence, due to the raster

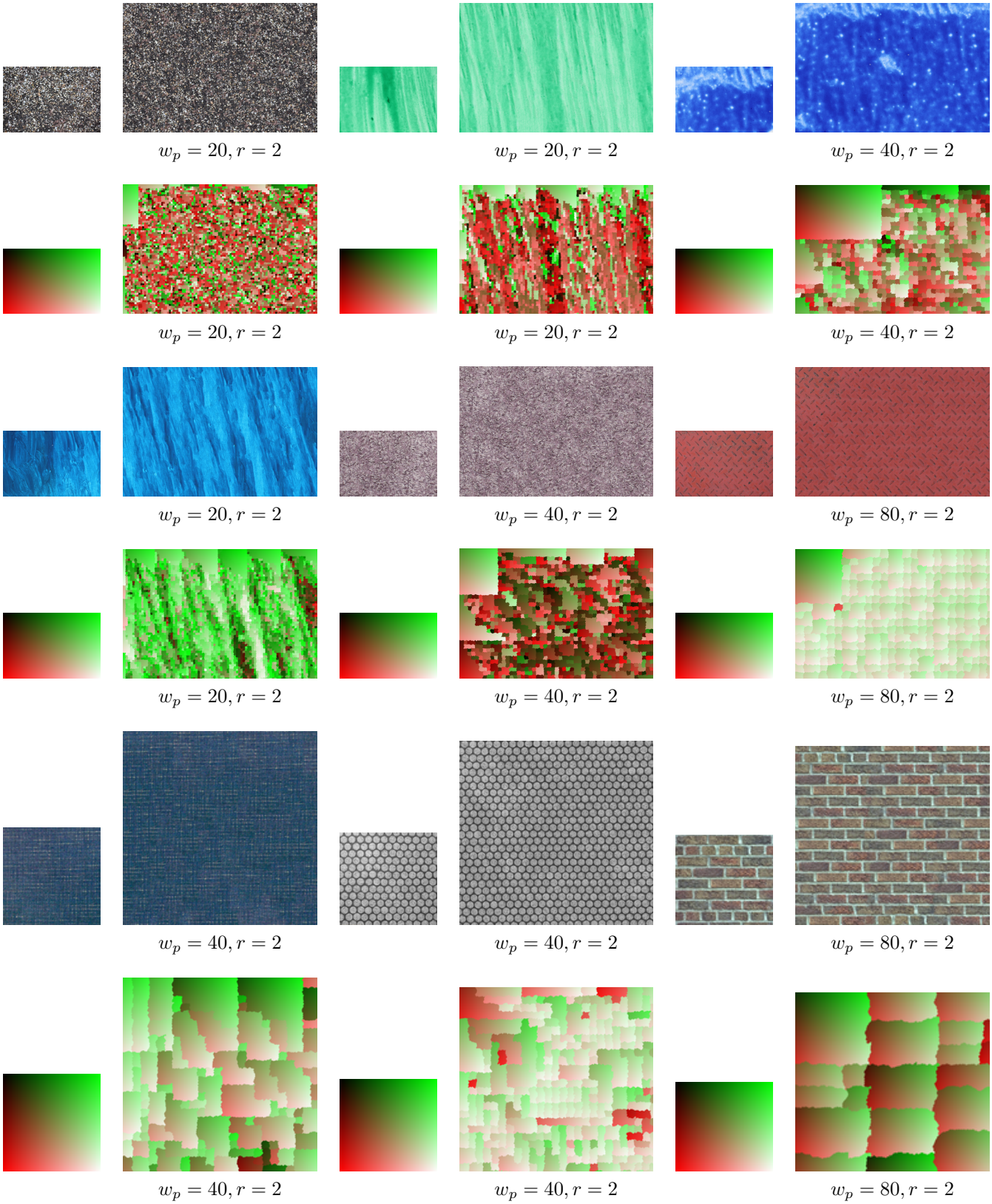


Figure 6: Successful results of Efros and Freeman image quilting algorithms. The small images represent the example texture and the big ones the corresponding synthesis result. Each row of examples is followed by a row containing the corresponding color and synthesis maps. For all the examples the patch size w_p and the ratio r used is indicated. The overlap size is fixed to $w_o = 0.25$ and the tolerance error to $\varepsilon = 0.1$.

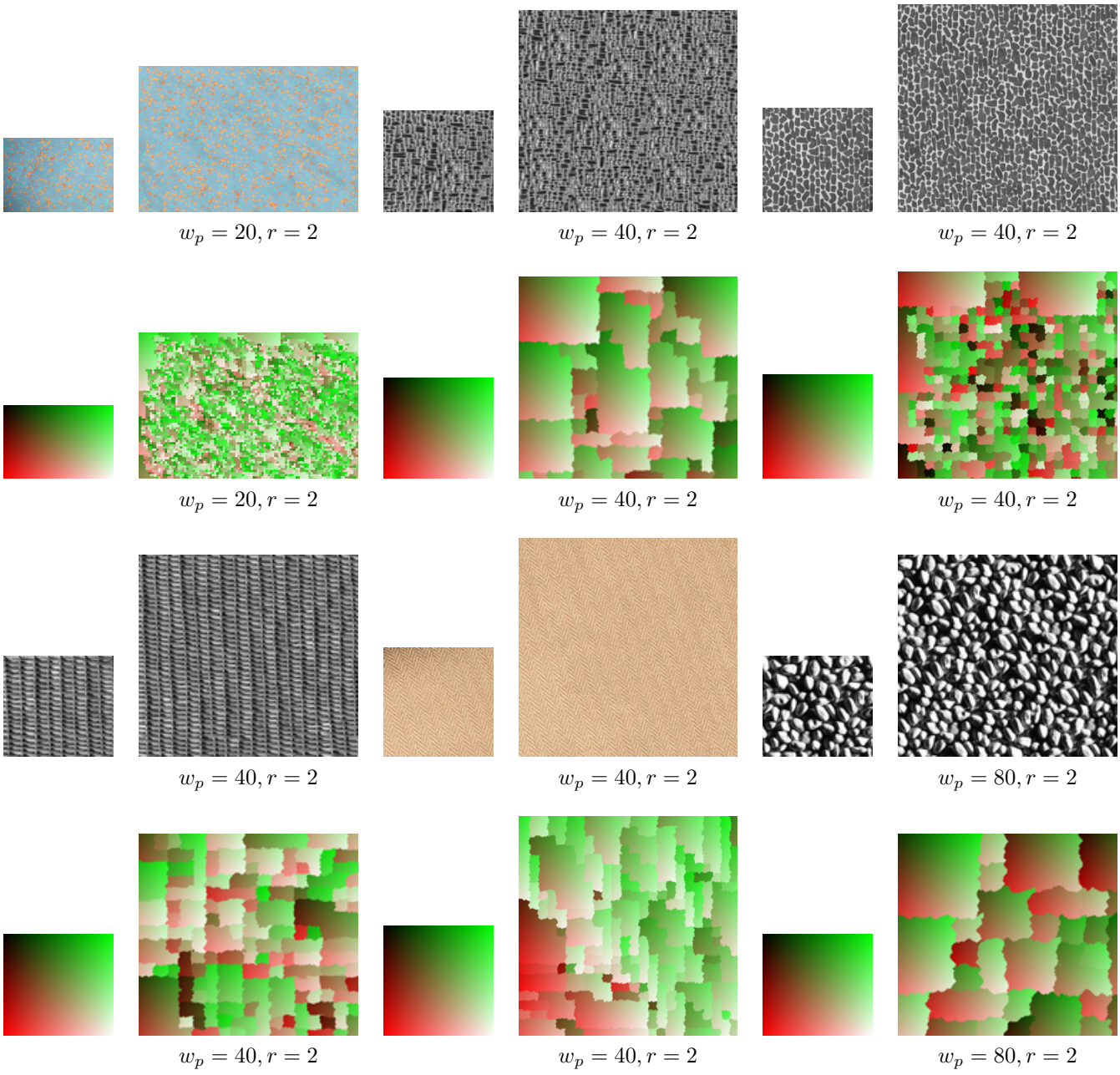


Figure 7: Successful results of Efros and Freeman image quilting algorithms. The small images represent the example texture and the big ones the corresponding synthesis result. Each row of examples is followed by a row containing the corresponding color and synthesis maps. For all the examples the patch size w_p and the ratio r used is indicated. The overlap size is fixed to $w_o = 0.25$ and the tolerance error to $\varepsilon = 0.1$.

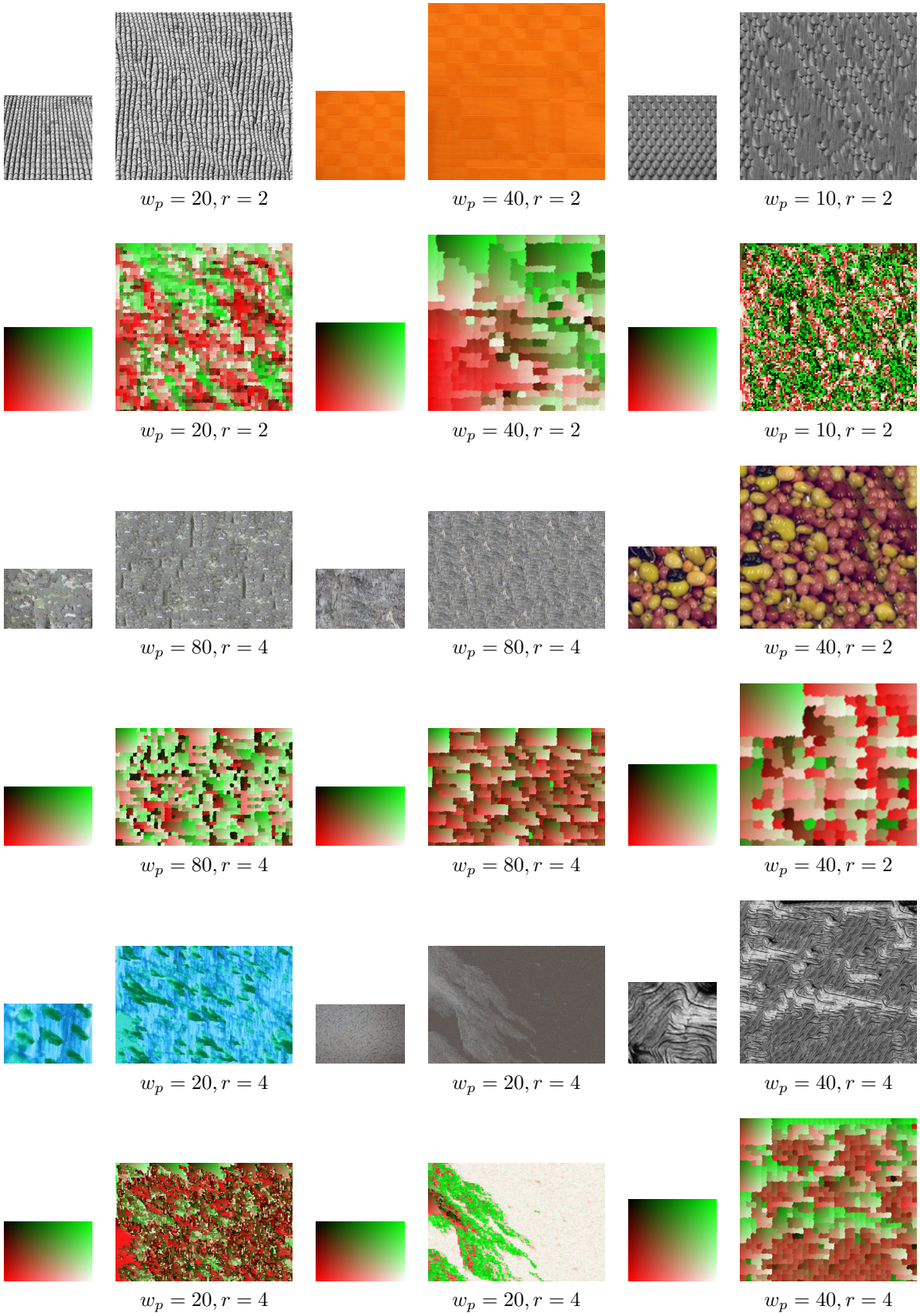


Figure 8: Failures of Efros and Freeman image quilting algorithms. The small images represent the example texture and the big ones the corresponding synthesis result. The examples in the first row show failures related to an incorrect patch size. The examples in the second row show the case of verbatim copy failures. The examples in the the third row show the case of growing garbage. For all the examples the patch size w_p and the ratio r used is indicated.

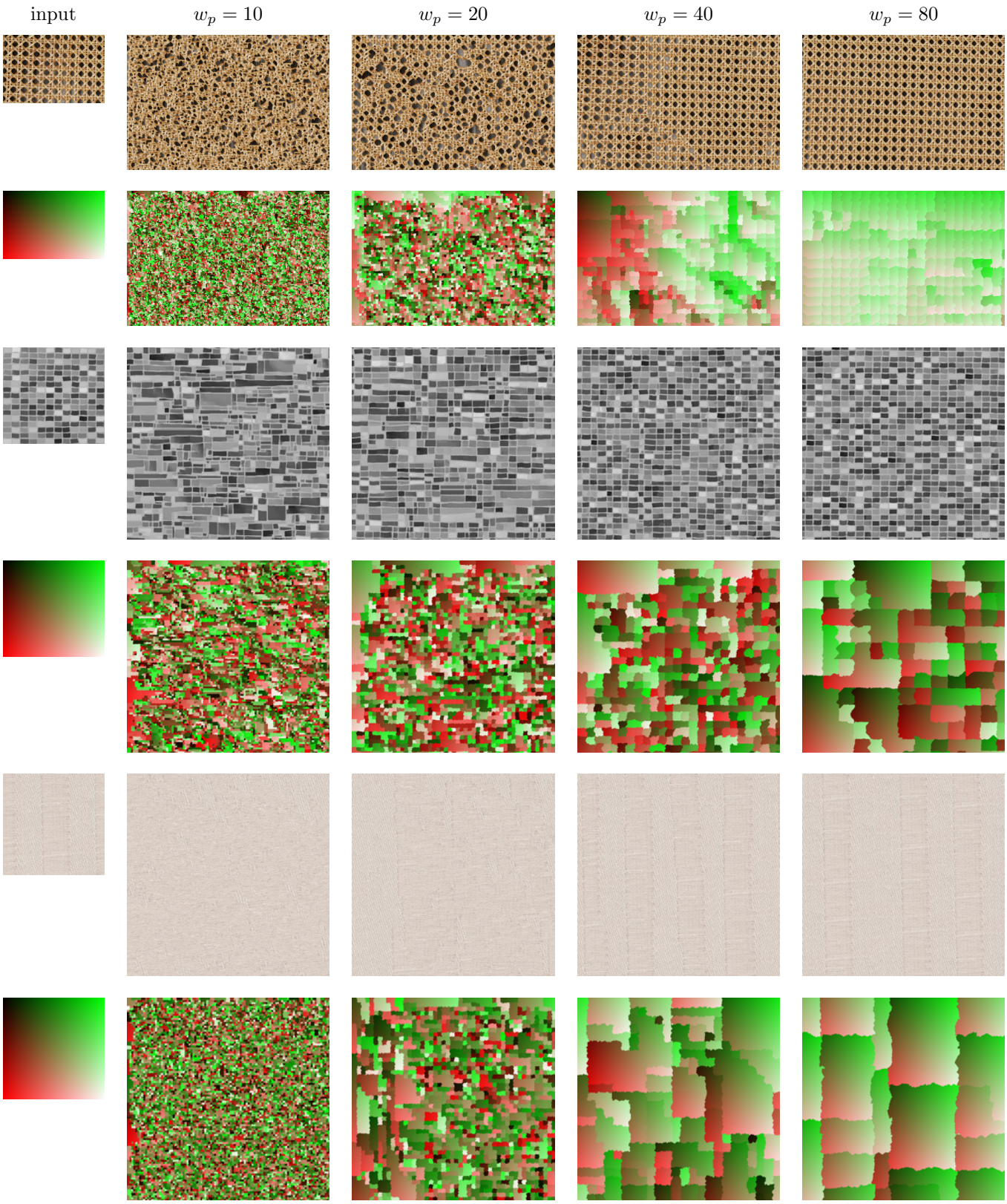


Figure 9: Patch size influence. Each pair of rows shows from left to right the input, the synthesis results for $w_p = 10, 20, 40, 80$ and the corresponding color and synthesis maps. For all the examples $w_o = 0.25$ and $\varepsilon = 0.1$.

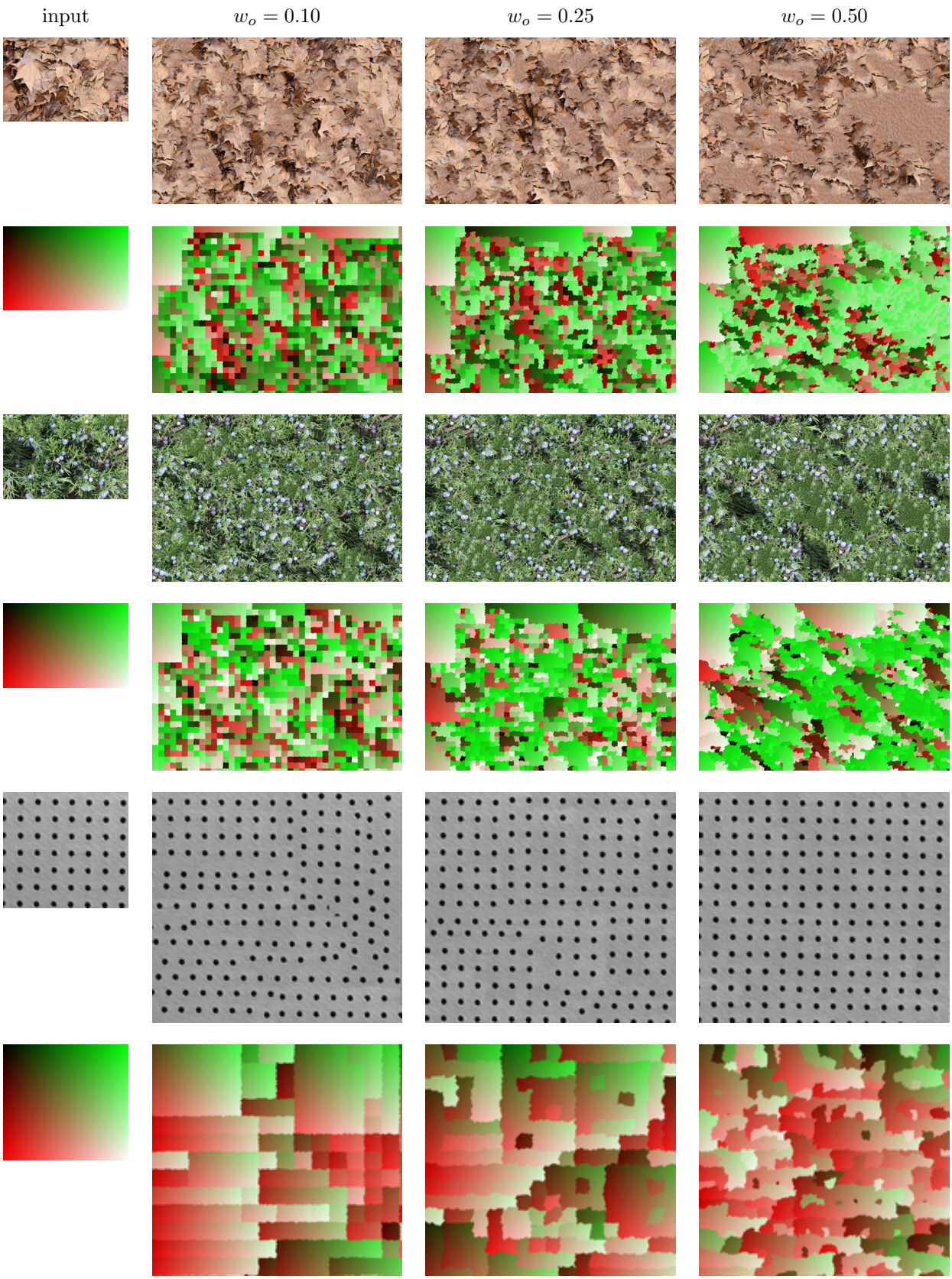


Figure 10: Overlap size influence. Each pair of rows shows from left to right the input, the synthesis results for $w_o = 0.1, 0.25, 0.5$ and the corresponding color and synthesis maps. For all the examples $w_p = 40$ and $\varepsilon = 0.1$.

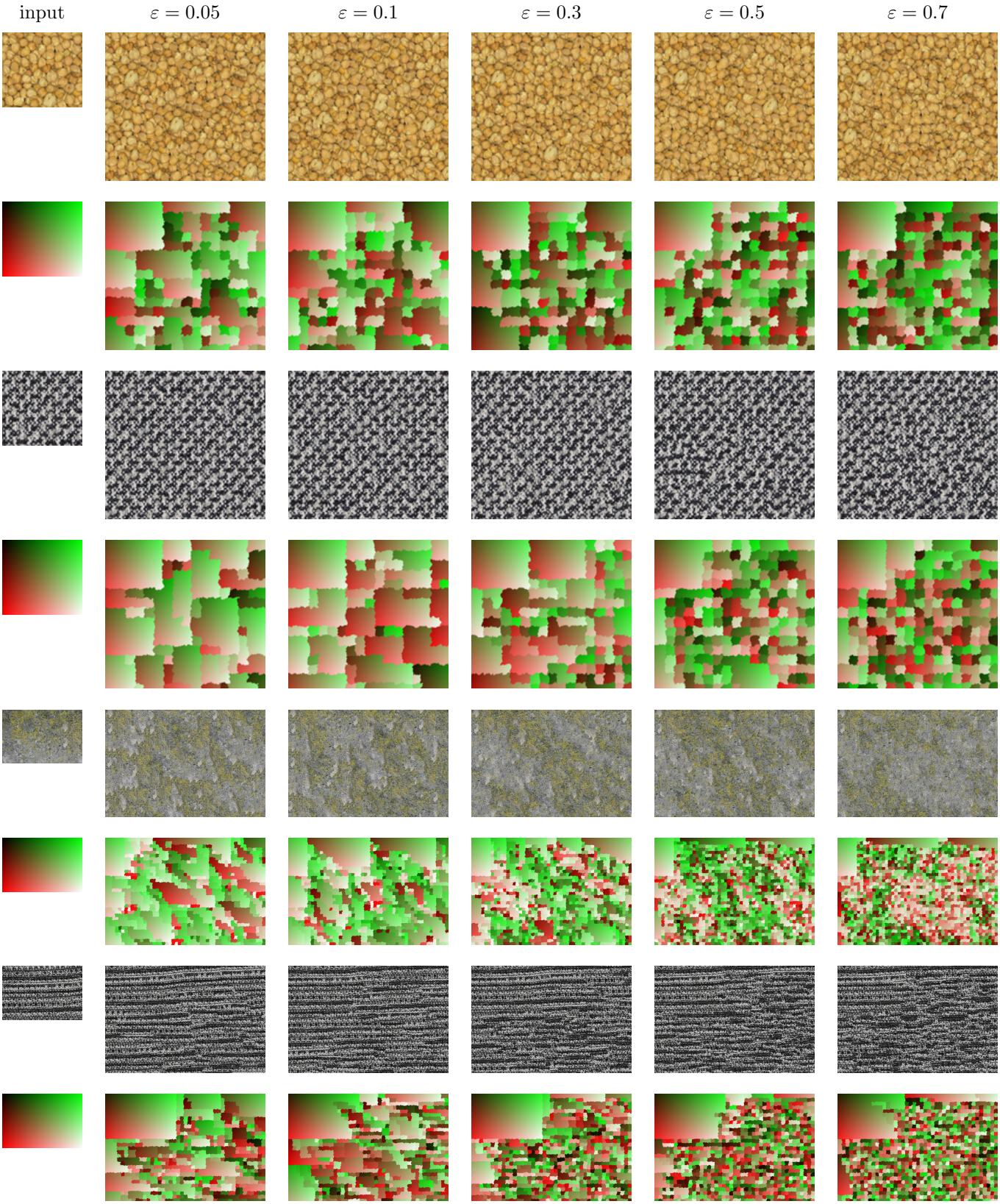


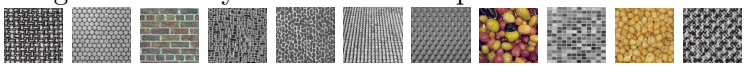
Figure 11: Tolerance error influence. Each couple of rows shows from left to right the input, the synthesis results for $\varepsilon = 0.05, 0.1, 0.3, 0.5, 0.7$ and the corresponding color and synthesis maps. For all the examples $w_p = 40$ and $w_o = 0.25$.

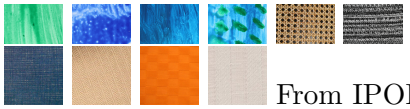
scan order, the quilting algorithm is not suitable to generate very large texture images since the quality tends to decrease with the distance to the top left corner of the image.

We provide with this analysis a strategy to partially parallelize the method who was initially introduced as essentially sequential. This allows a significant speed up when running with multi-core processors.

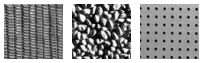
Image Credits

Images CC-BY by the authors except:

 By Eero P. Simoncelli[2].

 CC-BY Bruno Galerne and Yann Gousseau.

From IPOL archive for article [5].

 Fabric.0013, Food.0001 and Tile.0007 from the VisTex database[3].

References

[1] C. Aguerrebere, Y. Gousseau, and G. Tartavel, *Exemplar-based texture synthesis: the Efros-Leung algorithm*, Image Processing On Line, (2013). https://doi.org/10.5201/ipol.2013.59.

[2] T. Briand, J. Vacher, B. Galerne, and J. Rabin, *The Heeger & Bergen pyramid based texture synthesis algorithm*, Image Processing On Line, 4 (2014), pp. 276–299. https://doi.org/10.5201/ipol.2014.79.

[3] A. A. Efros and W. T. Freeman, *Image quilting for texture synthesis and transfer*, in SIGGRAPH, ACM, 2001, pp. 341–346. https://doi.org/10.1145/383259.383296.

[4] A. A. Efros and T. K. Leung, *Texture synthesis by non-parametric sampling*, in Proceedings of IEEE International Conference on Image Processing (ICIP), 1999, pp. 1033 – 1038. https://doi.org/10.1109/ICCV.1999.790383.

[5] B. Galerne, Y. Gousseau, and J.-M. Morel, *Micro-texture synthesis by phase randomization*, Image Processing On Line, (2011). https://doi.org/10.5201/ipol.2011.ggm_rpn.

[6] ——, *Random phase textures: Theory and synthesis*, IEEE Transactions on Image Processing, 20 (2011), pp. 257 – 267. https://doi.org/10.1109/TIP.2010.2052822.

[7] D. J. Heeger and J. R. Bergen, *Pyramid-based texture analysis/synthesis*, in SIGGRAPH, ACM, 1995, pp. 229–238. https://doi.org/10.1145/218380.218446.

[8] A. Kaspar, B. Neubert, D. Lischinski, M. Pauly, and J. Kopf, *Self tuning texture optimization*, in Computer Graphics Forum, vol. 34, Wiley Online Library, 2015, pp. 349–359. https://doi.org/10.1111/cgf.12565.

[2] http://www.cns.nyu.edu/~lcv/texture/

[3] http://vismod.media.mit.edu/vismod/imagery/VisionTexture/vistex.html

[9] V. Kwatra, A.Schödl, I. Essa, G. Turk, and A. Bobick, *Graphcut textures: image and video synthesis using graph cuts*, in SIGGRAPH, ACM, 2003, pp. 277–286. https://doi.org/10.1145/1201775.882264.

[10] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, , and H.-Y. Shum, *Real-time texture synthesis by patch-based sampling*, ACM Transactions on Graphics, 20 (2001), pp. 127–150. https://doi.org/10.1145/501786.501787.

[11] G. Peyré, *Texture synthesis with grouplets*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 4 (2010), pp. 733–746. https://doi.org/10.1109/TPAMI.2009.54.

[12] J. Portilla and E. P. Simoncelli, *A parametric texture model based on joint statistics of complex wavelet coefficients*, International Journal on Computer Vision, 40 (2000), pp. 49–71. https://doi.org/10.1023/A:1026553619983.

[13] L. Raad, A. Desolneux, and J.-M. Morel, *Locally Gaussian exemplar based texture synthesis*, in Proceedings of IEEE International Conference on Image Processing (ICIP), 2014, pp. 4667–4671. https://doi.org/10.1109/ICIP.2014.7025946.

[14] J. Rabin, G. Peyré, J. Delon, and M. Bernot, *Wasserstein barycenter and its application to texture mixing*, in Scale Space and Variational Methods in Computer Vision, vol. 6667 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2012, pp. 435–446. https://doi.org/10.1007/978-3-642-24785-9_37.

[15] G. Tartavel, Y. Gousseau, and G. Peyré, *Variational texture synthesis with sparsity and spectrum constraints*, Journal of Mathematical Imaging and Vision, (2014). https://doi.org/10.1007/s10851-014-0547-7.

[16] J. J. van Wijk, *Spot noise texture synthesis for data visualization*, in SIGGRAPH, ACM, 1991, pp. 309–318. https://doi.org/10.1145/122718.122751.

[17] L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk, *State of the art in example-based texture synthesis*, in Eurographics State of the Art Report, EG-STAR, 2009.

[18] L. Y. Wei and M. Levoy, *Fast texture synthesis using tree-structured vector quantization*, in SIGGRAPH, 2000, pp. 479–488. https://doi.org/10.1145/344779.345009.