Sparse Modeling of Textures

Gabriel Peyré
CNRS and Ceremade, Université Paris-Dauphine,
Place du Maréchal De Lattre De Tassigny,
75775 Paris Cedex 16 France
gabriel.peyre@ceremade.dauphine.fr

Abstract

This paper presents a generative model for textures that uses a local sparse description of the image content. This model enforces the sparsity of the expansion of local texture patches on adapted atomic elements. The analysis of a given texture within this framework performs the sparse coding of all the patches of the texture into the dictionary of atoms. Conversely, the synthesis of a new texture is performed by solving an optimization problem that seeks for a texture whose patches are sparse in the dictionary. This paper explores several strategies to choose this dictionary. A set of hand crafted dictionaries composed of edges, oscillations, lines or crossings elements allows to synthesize synthetic images with geometric features. Another option is to define the dictionary as the set of all the patches of an input exemplar. This leads to computer graphics methods for synthesis and shares some similarities with non-local means filtering. The last method we explore learns the dictionary by an optimization process that maximizes the sparsity of a set of exemplar patches. Applications of all these methods to texture synthesis, inpainting and classification shows the efficiency of the proposed texture model.

1 Introduction

The analysis and synthesis of textures is a central topic in computer vision and graphics. Various methods have been proposed to model textures and to sample new textures from the corresponding set of constraints. This paper proposes a framework for texture modeling based on a linear generative model for the set of patches extracted from the texture. Such a model is constrained by imposing sparsity in the decomposition of patches. The overlap of these patches turns the synthesis of a new texture into an optimization that is solved iteratively. Depending on the precise way to compute the sparse expansion of patches, one retrieves some previously proposed models that now fit into a common framework.

1.1 Sparse Models for Images and Textures

Spatial domain modeling. The works of Efros and Leung [19] and Wei and Levoy [55] pioneered a whole area of greedy approaches to texture synthesis. These methods copy pixels one by one, enforcing locally the consistency of the synthesized image with the exemplar. Later enhancements on this idea led to patch-wise copying, see for example the work of Efros and Freeman [20], Kwatra et al. [29] and Ashikhmin [3]. Recent approaches such as the methods of Lefebvre and Hoppe [31] and Kwatra et al. [28] are fast and use a multiscale strategy.

Section 3 presents a non-local computation of the sparse expansion of patches. These non-local weights generalize the idea of pixel recopy to perform average of pixels belonging to similar patches. The resulting iterative synthesis algorithm is similar to the texture optimization process of Kwatra et al. [28]. Section 3.3shows how these ideas relate to non-local means filtering as proposed by Buades et al. [11]. Brox and Cremers [10] have introduced an iterated non-local means algorithm that is used to perform denoising and differs from the non-local synthesis described in Section 3.3.

Transformed domain modeling. Julesz [26] stated simple axioms about the probabilistic characterization of textures. A texture is described as a realization of a random process characterized by the marginals of responses to a set of linear filters. Zhu, Wu and Mumford [58] setup a Gibbs energy to learn both the filters and the marginals. They use a Gibbs sampler to draw textures from this model.

A fast synthesis can be obtained by fixing the analyzing filters to be steerable wavelets as done by Heeger and Bergen [24] and by wavelet noise [15]. The resulting textures are similar to those obtained by Perlin [42]. They exhibit isotropic cloud-like structures and fail to reproduce long range anisotropic features. This is because wavelets decompositions represent sparsely point wise singularities but do not compress enough long edge features. Higher order statistics such as multiscale correlations [8] and local correlations [45] are used to synthesize high quality textures.

Dictionary learning. Wavelets and more recent tools from harmonic analysis [37] have proven to be efficient for image compression and denoising. It is however difficult to design efficient dictionaries for complex textures, as explained in the review paper of Simoncelli and Olshausen [47] on wavelet-based models for textures.

Olshausen and Field [40] proposed to learn a dictionary adapted to the processing of patches extracted from natural images. They have applied this learning to patches p_i extracted from natural images. The major conclusion of this line of research is that learning over a large set of disparate natural images leads to localized oriented edge filters. Other approaches to sparse coding have been applied with success using independent components analysis [6] and different sparsity priors on the coefficients [33, 27, 21, 2].

Specific properties of images are captured using constrained non-linear models, such as a decomposition using positive atoms and positive coefficients, see [30]. This non-negative generative process is used for texture modeling, synthesis and inpainting by Peyré [43]. Independent component analysis and sparse dictionaries have been applied in texture modeling mainly for features extraction in classification [56, 48]. An ICA decomposition is used as a post-processing step by Manduchi and Portilla [38] to enhance the synthesis results of Heeger and Bergen multiscale approach [24].

Section 4 presents a texture model based on a sparse expansion of patches in a learned dictionary. In contrast to Zhu et al. [57] that select the dictionary from a library of fixed atoms, a non-parametric approach is used and the atoms are optimized to enhance the synthesis result. The iterative synthesis process is similar to the iterative projection on constraints used by Portilla and Simoncelli [45]. Dictionary learning has been used by Mairal et al. [35] to perform color image denoising and inpainting. Our sparse texture model allows to use this dictionary learning scheme for texture synthesis.

Manifold models for textures. The set of patches extracted from a texture can have a complex geometric structure that reflects the interactions between the patterns of the image. A simple model for such a set is a low dimensional manifold embedded in a high dimensional space. The dimension of this manifold measures the number of intrinsic parameters that govern the patches formation and layout in the image. Peyré [44] studies this manifold structure for simple image models such as locally parallel textures and periodic tillings of patterns. Recent approaches to image synthesis in computer graphics use manifold modeling of textures. Matusik et al. define a manifold from a set of textures [39]. Lefebvre and Hoppe introduce a mapping of an image into a higher dimensional appearance space [32]. This embedding allows a synthesis with high fidelity and spatial variations.

This paper proposes a sparse model for texture patches. This model assumes that the set of patches extracted from an image lives in the union of low dimensional vector spaces. This model can be used to perform perform texture processing.

1.2 Texture Processing

Texture synthesis is achieved by sampling at random a texture model. More elaborated texture processing can be devised to perform texture restoration, mixing and segmentation.

Texture inpainting. The inpainting problem consists in filling a set of missing pixels of a damaged image. Non-textured inpainting is solved using evolution equations derived from fluid dynamics by Bertalmio et al. [7] and Ballester et al. [4]. Anisotropic diffusion along a tensor field is used by Tschumperlé and Deriche [52].

Texture inpainting is closely related to texture synthesis with the additional constraint that the synthesis should be coherent with the available set of pixels. Starck et al. [49]and Fadili et al. [22] inpaint both edges and oscillatory textures using sparsity in a set of fixed bases such as curvelets and local DCT. Criminisi et al. [16] inpaint regions with complex textures using pixel recopy and patch comparisons. Section 3.4 describes an iterative scheme that brings together both sparsity and patch-based methods using our sparse texture model.

Texture mixing The problem of texture mixing consists in synthesizing a texture that blends seamlessly the geometric structures of two input exemplars. Bar-Joseph et al. [5] and Portilla and Simoncelli [45] have proposed architectures for texture mixing that use a multiscale wavelet decomposition. Section 4.3 extends these approaches by using a dictionary learned from the two input textures.

Texture classification Texture segmentation has been studied extensively in computer vision. The unsupervised segmentation problem is usually solved by computing local texture descriptors for each pixel and then applying a standard clustering algorithm. For instance, early works on texture analysis use outputs from a set of Gabor filters [25, 9, 34, 13] and local moments of pixel values have been used by Tuceryan [54]. In contrast, supervised texture segmentation uses a set of exemplars to build some statistical model for each texture class. Active contours methods have been extended to textures using statistical multiscale descriptors by Paragios and Deriche [41]. Section 4.4uses our sparse texture model to perform texture classification, and is similar to the approach proposed by Skretting et al. [48].

2 Sparse Decompositions of Texture Patches

This section introduces a new model for textures which is based on a sparse expansion of image patches in a local dictionary. This model is sampled for texture synthesis purpose with a iterative algorithm that optimizes a sparsity-promoting energy.

2.1 Patch Domain Modeling

This article focusses on the local geometry of textures through the extraction of local patches. An image $f \in \mathbb{R}^N$ of N pixels is processed by extracting patches $p_x(f)$ of size $\tau \times \tau$ around each pixel position $x \in \{0, \dots, \sqrt{N}-1\}^2$

$$\forall\, t \in \{-\tau/2+1, \dots, \tau/2\}^2, \qquad p_x(f)(t) = f(x+t). \tag{1}$$

A patch $p_x(f)$ is handled as a vector of size $n = \tau^2$. In the following we also consider color images f of N pixels that can be handled as vectors of dimension $3N$. This article uses periodic boundary conditions to ease notations, but symmetric conditions with reflecting boundaries can be used with slight modifications.

In the following, $\Phi : f \mapsto \{p_x(f)\}_x$ is the linear operator that extracts all the patches from an image. This patch extraction is a mapping $\Phi : \mathbb{R}^N \to \mathbb{R}^{n \times N}$ where n is the dimension of each patch and N is the number of patches in an image of N pixels. The matrix whose columns are all the patches $p_i = p_{x_i}(f)$ is denoted as $\Phi(f) = P = (p_0, \dots, p_{N-1}) \in \mathbb{R}^{n \times N}$ where x_i indexes the N pixels of f.

An image $\tilde{f}$ is recovered from a given set of patches $P = \{p_i\}_i$ using the pseudo-inverse Φ^+ defined as

$$\Phi^+(P) = \tilde{f} \qquad \text{where} \qquad \tilde{f} = \operatorname*{argmin}_{g \in \mathbb{R}^n} \sum_{i=0}^{N-1} \|p_{x_i}(g) - p_i\|^2. \tag{2}$$

This pseudo-inverse reconstruction corresponds to an averaging of overlapping patches

$$\forall\, x \in \{0,\dots,\sqrt{N}-1\}^2, \qquad \tilde{f}(x) = \frac{1}{n} \sum_{|x_i - x| \leqslant \tau/2} p_i(x - x_i). \tag{3}$$

2.2 Sparse Expansion of Patches

Linear forward generative model. A linear generative model assumes that a patch $p \in \mathbb{R}^n$ of $n = \tau \times \tau$ pixels is approximated as a linear superposition

$$p \approx \sum_{k=0}^{m-1} w(k)\, d_k = D\, w. \tag{4}$$

Each $w(k) \in \mathbb{R}$ is a coefficient associated to the atom $d_k \in \mathbb{R}^n$, and these atoms are stored in a dictionary $D = \{d_k\}_k$ which is a matrix $D \in \mathbb{R}^{n \times m}$.

The collection of patches $P = \{p_i\}_i = \Phi(f)$ extracted from an image $f \in \mathbb{R}^N$ is decomposed with this linear generative model as

$$\Phi(f) = P = DW \qquad \text{where} \qquad W = \{w_i\}_i \in \mathbb{R}^{m \times N} \quad \text{and} \quad p_i = D w_i. \tag{5}$$

This dictionary D is the main feature of our texture model and its atoms d_k should be carefully chosen to represent efficiently geometric patterns of the textures to analyze and synthesize. Sections 2.4, 3 and 4 explore different ways to handle this dictionary learning problem.

Sparse decomposition. Equation (4) describes a forward process that generates a patch given a set of coefficients. The problem of analyzing a given image f using the local dictionary D is more complex and involves a modeling stage that enforces constraints on the set of coefficients. In particular, since both the mapping Φ and the dictionary D are highly redundant, there is many ways to perform this analysis.

The patches are modeled by requiring that they are well approximated by a sparse expansion using the dictionary D. This means that for a given patch $p \approx Dw$, only a few atoms d_k are active to describe p. This requires that the ℓ^0 pseudo-norm of w is small, where $\|w\|_{\ell^0}$ counts the number of non-zero coefficients of w

$$\|w\|_{\ell^0} = \#\left\{k \setminus w(k) \neq 0\right\}. \tag{6}$$

Such a sparse set of coefficients w approximating a given patch p is obtained by solving

$$w = \operatorname*{argmin}_{c \in \mathbb{R}^m} \|p - Dc\|_{\ell^2} \quad \text{subject to} \quad \|c\|_{\ell^0} \leqslant s, \tag{7}$$

where s is the sparsity constant of our model.

The optimization problem (7) is combinatorial and thus intractable. In practice, several approximation algorithms allows one to compute sparse coefficients w:

– *Convexification of the objective:* relaxing the problem (7) replaces the ℓ^0 pseudo-norm by the ℓ^1 norm $\|w\|_{\ell^1} = \sum_k |w(k)|$. This leads to the following convex program

$$w = \operatorname*{argmin}_{c \in \mathbb{R}^m} \|p - Dc\|_{\ell^2}^2 + \lambda \|c\|_{\ell^1}, \tag{8}$$

where λ is a parameter that controls the sparsity of w, and should be tuned so that $\|w\|_{\ell^0} \leqslant s$. This convex optimization (8) is the basis pursuit denoising problem introduced by Chen et al. [12]. It can be solved using interior point algorithms [12] or iterative thresholdings, see for instance [17]. Under restrictive conditions on both D and p, this relaxed optimization actually solves (7), see [51] for instance.

– *Greedy approximation:* approximate algorithms such as matching pursuit or orthogonal matching pursuit compute in a greedy manner the coefficients of w, see [37]. Such greedy approximations are usually less accurate than the ℓ^1 relaxation (8) but offers a faster way to compute w. Under restrictive conditions, these greedy methods can be proved do solve the original problem (7), see for instance [50].

The numerical experiments use the matching pursuit algorithm, which is faster than both orthogonal matching pursuit and basis pursuit, and works well for low values of the sparsity parameter s. Table 1 gives the details of this algorithm.

Table 1 Matching pursuit algorithm for solving approximately (7).

1. *Initialization:* $w \leftarrow 0$, set $i = 0$.
2. *Best correlation:* compute the best matching atom
$$k^\star = \operatorname*{argmax}_k \frac{1}{\|d_k\|}\langle r,\, d_k\rangle.$$
3. *Update:* modify the residual and the coefficients:
$$r \leftarrow r - \frac{1}{\|d_{k^\star}\|^2}\langle r,\, d_{k^\star}\rangle d_{k^\star} \qquad \text{and} \qquad w(k^\star) \leftarrow w(k^\star) + \frac{1}{\|d_{k^\star}\|^2}\langle r,\, d_{k^\star}\rangle.$$
4. *Stop:* if $i < s$, go back to 2.

2.3 Sparse Texture Synthesis

Given a fixed dictionary D for patches of size $w \times w$, the texture synthesis processes by searching for an image whose patches are sparse in D. This amounts to solve the following optimization problem

$$\min_{f \in \mathbb{R}^N} E_D(f) \quad \text{subject to} \quad f \in \mathcal{C} \tag{9}$$

where the energy is defined as

$$E_D(f) = \min_{W \in \mathbb{R}^{N \times m}} \|\Phi(f) - DW\|_{\ell^2} \qquad \text{subject to} \qquad \forall\, i,\ \|w_i\|_{\ell^0} \leqslant s. \tag{10}$$

Each w_i corresponds to the coefficients of the patch $p_{x_i}(f)$ which has to be sparse in D. The additional constraint $f \in \mathcal{C}$ forces the synthesized f to move away from the trivial solution $f = 0$ and is detailed in the next paragraph.

Although the energy $\|\Phi f - DW\|_{\ell^2}$ in (10) is convex as a function of f and W, the additional ℓ^0 and $\mathcal{C}$ constraints make this minimization non-convex. In our framework, a valid synthesized texture is defined as a stationary point of this energy. The energy E_D has typically many local stationary points. To sample quite uniformly this set of local minima, E_D is optimized using a descent algorithm that starts from a random initial texture.

Histogram constraints. Arbitrary meaningful constraints $f \in \mathcal{C}$ can be imposed in the synthesis optimization (9). This article considers a constraint on the histogram of the set of pixels

$$\mathcal{C} = \left\{ f \in \mathbb{R}^N \setminus H(f) = H(\tilde{f}) \right\} \tag{11}$$

where $\tilde{f}$ is a given input exemplar, and where $H(f)$ is the discrete histogram of the values of f. The goal of this constraint is to enforce f to have the same gray-level repartition as f.

Histogram-matching computes the orthogonal projection $f_0 = \mathcal{P}_{\mathcal{C}}(f)$ of f on $\mathcal{C}$. If both f and $\tilde{f}$ have N pixels, this projection is computed by first sorting the values of f and $\tilde{f}$, which corresponds to the computation of indexes $\alpha(i)$ and $\tilde{\alpha}(i)$ such that

$$\forall\, i = 1, \ldots, N-1, \quad \begin{cases} f(\alpha(i-1)) \leqslant f(\alpha(i)), \\ \tilde{f}(\tilde{\alpha}(i-1)) \leqslant \tilde{f}(\tilde{\alpha}(i)), \end{cases} \tag{12}$$

and then the copy of the sorted values from $\tilde{f}$ to f_0

$$\forall\, i, \quad f_0(\alpha(i)) = \tilde{f}(\tilde{\alpha}(i)). \tag{13}$$

In the case where $\tilde{f}$ and f do not have the same number of samples, this formula requires interpolation.

Texture synthesis algorithm. Energy (9) is minimized with an iterative texture synthesis algorithm. It optimizes the energy sequentially on the image f and on the patch coefficients W. Table 2 details this texture synthesis algorithm. Such a block-coordinates relaxation has been proved to converge by Tseng [53] to a stationary point of E_D if one replaces the ℓ^0 pseudo-norm by the ℓ^1 norm. The texture obtained at convergence is a valid sample from the sparse texture model.

The following sections apply this synthesis algorithm in various situations where the dictionary D is either created in an had-oc manner (Section 2.4) or learned from some input data (Sections 3 and 4).

Table 2 Texture synthesis algorithm for minimizing (9).

1. *Initialization:* set $f \leftarrow$ random.
2. *Computing the patches:* $P = \Phi f$.
3. *Sparse coding:* perform the matching pursuit, table 1, to compute
$$\forall\, i = 0, \ldots, N-1, \quad w_i = \operatorname*{argmin}_{w \in \mathbb{R}^m} \|p_i - Dw\|_{\ell^2} \quad \text{subject to} \quad \|w\|_{\ell^0} \leqslant s$$
4. *Texture reconstruction:* reconstruct the patches $p_i = Dw_i$ and $f \leftarrow \Phi^+ P$, where the pseudo-inverse is defined in equation (3).
5. *Imposing histograms:* Perform equalization $f \leftarrow \mathcal{P}_{\mathcal{C}}(f)$.
6. *Stop:* while not converged, go back to 2.

2.4 Examples with Synthetic Dictionaries

Before detailing in the next sections how to learn dictionaries from some given texture exemplar, this section explores the synthesis algorithm with hand-crafted synthetic dictionaries. The dictionaries D we consider are parameterized by a small number of parameters as follow

$$D = \{d_i\}_{i=0}^m \quad \text{where} \quad \forall\, t \in \{-\tau/2+1, \ldots, \tau/2\}^2, \quad d_i(t) = \varphi_{\lambda_i}(2t/\tau), \tag{14}$$

where each λ_i is drawn uniformly at random in some set of parameter $\lambda_i \in \Lambda$. In the remaining of this section, we consider various kind functions $\varphi_\lambda : [-1,1]^2 \to \mathbb{R}$, for $\lambda \in \Lambda$. Similar ensembles of low dimensional set of patches are described in details by Peyré [44].

Dictionary of edges. A simple model of geometric images is the cartoon model introduced by Donoho [18]. A cartoon function is regular outside a set of edge curves which are themselves regular. A typical patch extracted from such a geometric cartoon image is well approximated by a binary straight edge. A dictionary composed of binary edges is generated from the following set of functions

$$\varphi_\lambda(t) = P\big(R_\theta(x - (\delta, 0))\big), \quad \text{where} \quad \lambda = (\theta, \delta) \in [0, 2\pi) \times \mathbb{R}^+, \tag{15}$$

where R_θ is the planar rotation of angle θ and the step is $P = h * \tilde{P}$ where $\tilde{P}(t) = 0$ if $t_1 < 0$ and $\tilde{P}(t) = 1$ otherwise. In this edge model, the local geometry of the image is described by θ which parameterizes the orientation of the closest edge and δ which is the distance to that edge. Figure 1 shows an example of these edge patches.

Figure 2 shows the iterations of the synthesis algorithm of table 2 for this dictionary of edges. The resulting synthesized image is a cartoon image with smooth edges.

Dictionary of local oscillations. The following set of functions

$$\varphi_\lambda(t) = \sin\big(R_\theta(t - (\delta, 0))/\nu\big), \quad \text{and} \quad \lambda = (\theta, \delta) \in [0, 2\pi) \times \mathbb{R}^+ \tag{16}$$

is used to synthesized oscillating textures. The local frequency ν controls globally the width of the oscillations whereas θ is the local orientation of these oscillations.

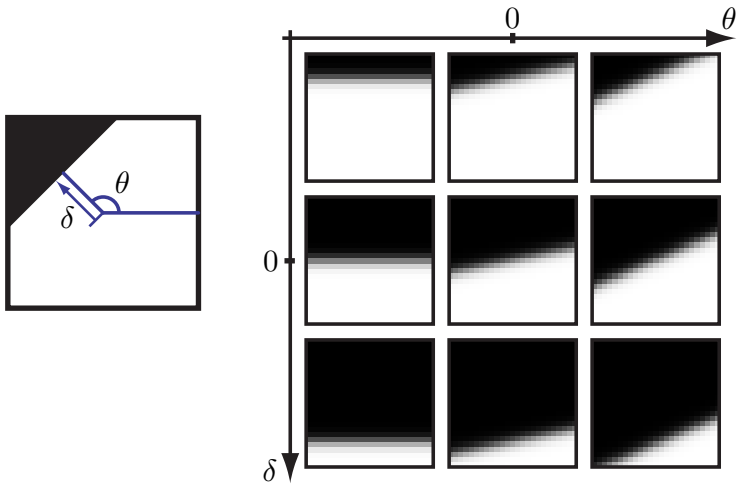


Figure 1: *Parameterization of the dictionary of edge patches and some examples.*

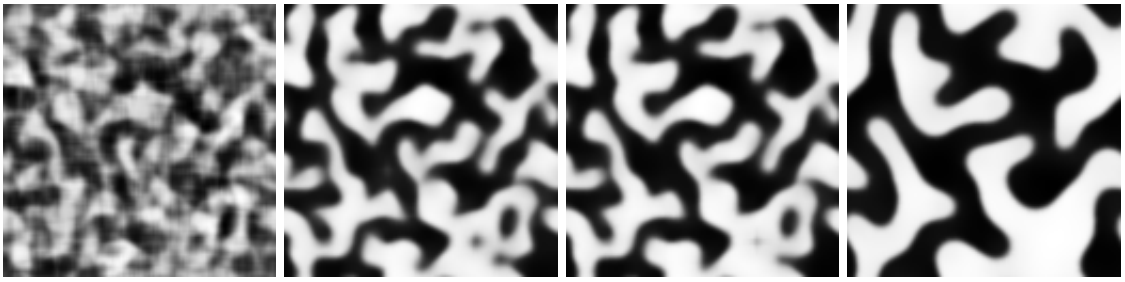


Figure 2: *Iterations of the synthesis algorithm with the dictionary of edges (sparsity $s = 2$).*

Dictionaries of lines. Similarly to the edge dictionary (15), a dictionary of lines is obtained by rotating and translating a straight line

$$\varphi_\lambda(t) = \ell_{\theta,\delta,\sigma}(t) = \exp\left(\frac{1}{2\sigma^2}\|R_\theta(t-(\delta,0))\|^2\right), \tag{17}$$

where $\lambda = (\theta, \delta) \in [0, 2\pi) \times \mathbb{R}^+$ and where σ control the width of the line pattern.

Dictionaries of crossings. A dictionary of crossings is obtained by considering atoms which contain two overlapping lines

$$\varphi_\lambda(t) = \max\left(\ell_{\theta_1,\delta_1,\sigma}(t), \ell_{\theta_2,\delta_2,\sigma}(t)\right) \quad \text{where} \quad \lambda = (\theta_1, \delta_1, \theta_2, \delta_2). \tag{18}$$

Figure 3 shows examples of synthesis for the four dictionaries generated by the set of functions (15), (16), (17) and (18).

3 Strict Sparsity and Non-local Expansions

Most approaches for texture synthesis in computer graphics [19, 55, 20, 29, 3, 31, 28] perform a recopy of patches from an original input texture f to create a new texture $\tilde{f}$ with similar structures. These processings can be re-casted into the sparse texture model. This section considers our texture model in a restricted case where one seeks a strict sparsity with $s = 1$ in a highly redundant dictionary.

3.1 Strict Sparsity Model

Considering the special case $s = 1$ means that each patch of the synthesized image f should be close to a patch in the original exemplar texture $\tilde{f}$. Within this assumption, it makes sense to

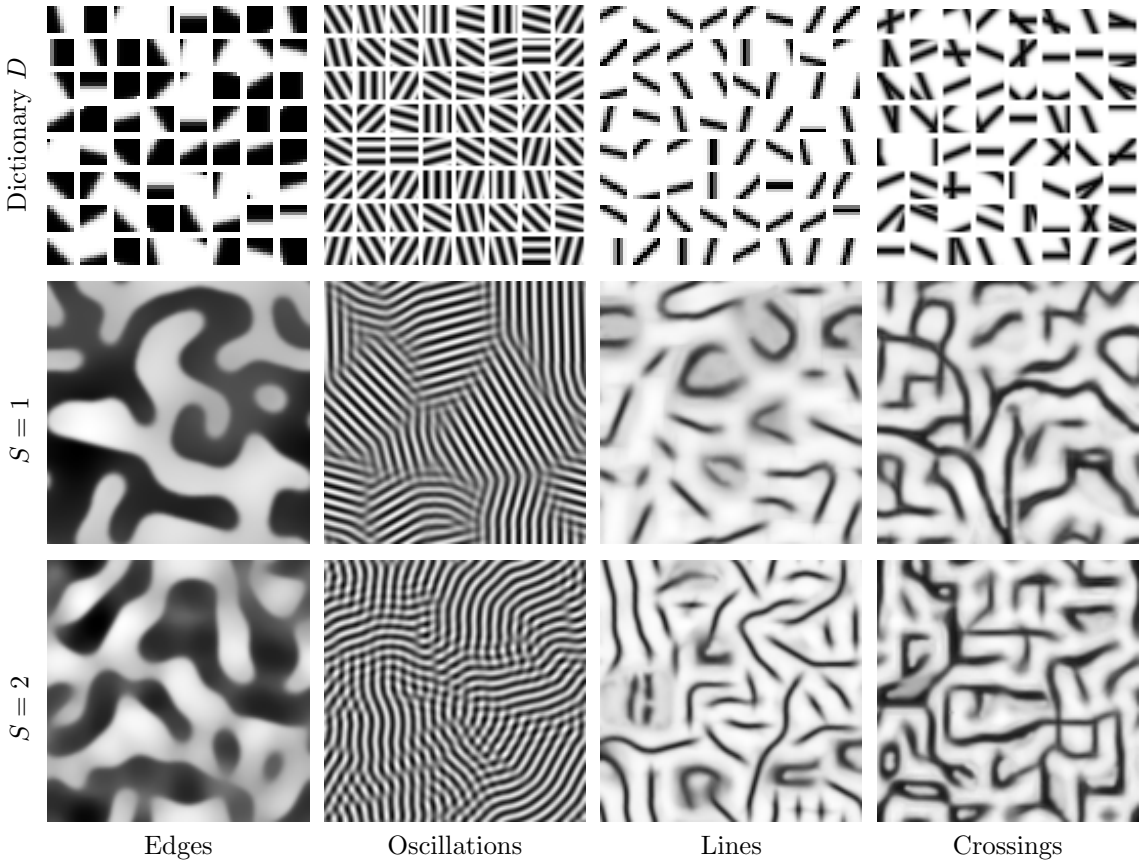


Figure 3: *Examples of synthesis for two sparsity levels s for the four kinds of dictionaries considered.*

define the dictionary D as the set of all the patches extracted from $\tilde{f}$

$$D = \{p_{x_i}(\tilde{f})\}_{i=0}^{N-1} = \Phi(\tilde{f}). \tag{19}$$

This dictionary is highly redundant and the synthesis algorithm looks for a matching between patches of f and $\tilde{f}$

$$\forall\, i, \quad p_{x_i}(f) = \lambda_i\, p_{\gamma(x_i)}(\tilde{f}), \qquad \text{where} \qquad \lambda_i \in \mathbb{R}, \tag{20}$$

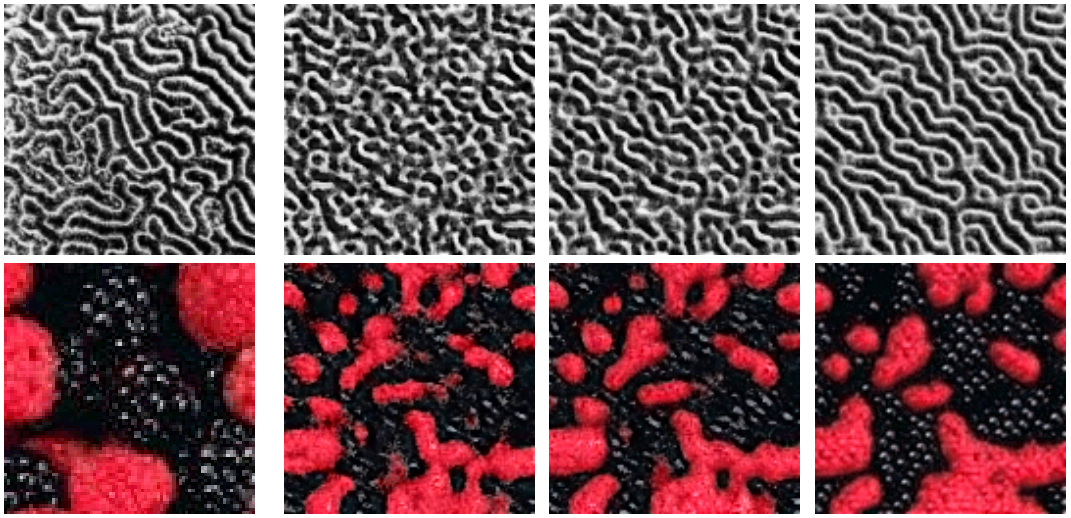
where $\gamma : \{0, \dots, \sqrt{N}-1\}^2 \to \{0, \dots, \sqrt{N}-1\}^2$ maps the pixel locations of the synthesized f to the pixel locations of $\tilde{f}$.

A further simplifying assumption done frequently in computer graphics assumes that $\lambda_i = 1$, which leads to the following definition of the mapping γ

$$\forall\, x, \quad \gamma(x) = \operatorname*{argmin}_y \|p_x(f) - p_y(\tilde{f})\|. \tag{21}$$

In this setting, the algorithm described in table 2 iterates between the best-fit computation (21) (step 3) and the averaging of the patches (step 4). This is similar to the optimization procedure of Kwartra et al. [28].

The iterative algorithm described in table 2 is used to draw a random texture that minimizes E_D. Figure 4 shows the iterations of texture synthesis using the highly redundant dictionary (19). For these examples, the size of the patches is set to $\tau = 6$ pixels. Figure 5 shows other examples of synthesis and compares the results with texture quilting [20]. Methods based on pixels and regions copy like [20] tend to synthesize images very close to the original. Large parts of the input are often copied verbatim in the output, with sometime periodic repetitions. In contrast, and similarly to [28], our method treats all the pixels equally and often leads to a better layout of the structures, with less global fidelity to the original.



Exemplar $\tilde{f}$ Iter. #1 Iter. #3 Iter. #20

Figure 4: *Iterations of nearest-neighbors matching for texure synthesis.*

3.2 Multiscale synthesis

The choice of the size parameter τ is non-trivial and requires some prior knowledge of the typical width of the structures one would like to maintain during the synthesis process. A multiscale synthesis strategy is used to cope with the fixed scale τ. A fixed number of pixels τ_0 is used but textures are synthesized with an increasing resolution. This process captures first elongated structures and then fine scale geometric details. An interpolation is used to switch between the various resolutions. At each scale, the synthesis algorithm manipulates only small patches of size $\tau_0 \times \tau_0$. This leads to the algorithm described in table 3 that handles J scales.

Figure 5 shows examples of multiscale synthesis using three scales $\tau \in \{4, 8, 16\}$. It shows in particular a comparison between the single scale algorithm and the multiscale extension, which is able to better recover elongated structures. Another advantage of this approach is that it speeds up the computation since the synthesis algorithm converges faster than with a single patch size.

Table 3 Multiscale synthesis algorithm.

1. *Initialization:* Set $j = J$ to be the coarser scale. Initialize the synthesis with a random noise f_J of $N/2^J \times N/2^J$ pixels.

2. *Compute the dictionary:* Set $\tau = 2^j \tau_0$. Smooth the exemplar $\tilde{f}_j = \tilde{f} * h_j$ where h_j is a gaussian kernel of width 2^j pixels. The current dictionary D_j is composed of the patches extracted from $\tilde{f}_j$ and subsampled by a factor 2^j. The elements of D_j have thus $\tau_0 \times \tau_0$ pixels.

3. *Synthesis:* perform the synthesis by minimizing E_{D_j} using algorithm described in table 2. The algorithm is initialized with the current f_j and this estimate is updated by the optimization

$$f_j = \underset{g \in \mathbb{R}^{N/2^j}}{\operatorname{argmin}} E_{D_j}(g) \quad \text{subj. to} \quad g \in \mathcal{C}.$$

4. *Up-sample:* if $j = 0$, then stop the algorithm and return $f = f_0$. Otherwise, upsample the current synthesized texture f_j to obtain f_{j-1} using linear interpolation from $N/2^j \times N/2^j$ pixels to $2N/2^j \times 2N/2^j$ pixels.

5. *Stop:* while $j > 0$, set $j \to j - 1$ and go back to step 2.

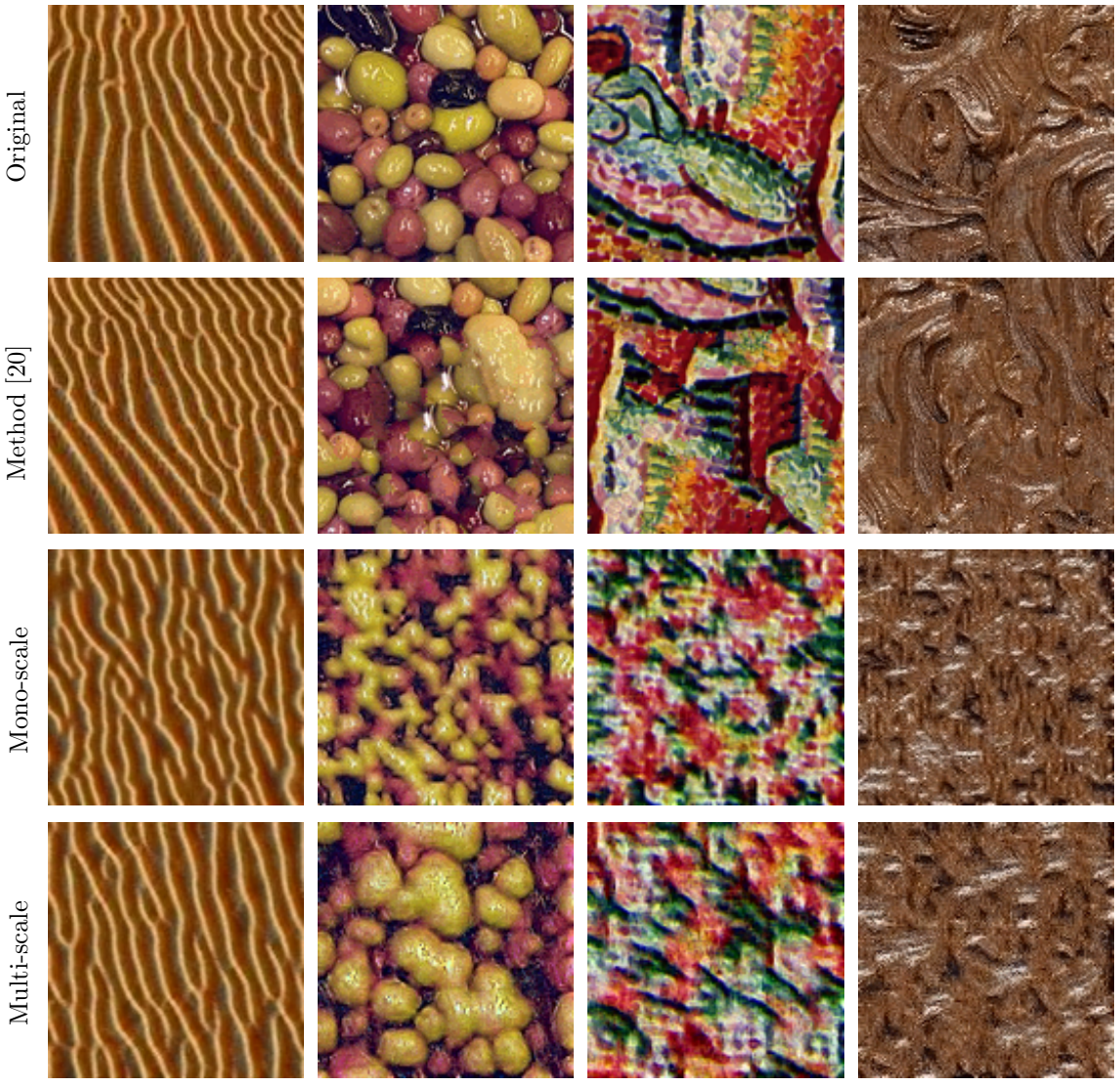


Figure 5: *Examples of synthesis using [20] and with our method with a single-scale* $\tau = 6$ *and a multiscale* $\tau \in \{4, 8, 16\}$ *(algorithm of table 3).*

3.3 Connexions with Non-local Means Filtering

Instead of performing the exact recopy of one best fitting patch $p_{\gamma(x_i)}(\tilde{f})$, as defined in equation (21), one can select several patches and use them to do the reconstruction. This shares some similarities to the non-local means algorithm introduced by Buades et al. [11] that performs image denoising using a spatially varying filter.

Using some input exemplar image $\tilde{f}$, an image f is filtered using

$$\mathcal{N}_{\tilde{f}}(f)(x) = \sum_y \xi(x,y)\tilde{f}(y) \qquad \text{where} \qquad \xi(x,y) = \frac{1}{Z_x} G_\sigma(p_x(f) - p_y(\tilde{f})) \tag{22}$$

where the weights depends on the distance between patches in the image

$$G_\sigma(a) = \exp\left(-\frac{\|a\|^2}{2\sigma^2}\right) \qquad \text{and} \qquad Z_x = \sum_y G_\sigma(p_x(f) - p_y(\tilde{f})). \tag{23}$$

The original non-local means algorithm [11] corresponds to the filtering of f itself and produces an estimation $\mathcal{N}_f(f)$.

This non-local filtering suggests to replace the sparse coding (step 3 of table 2) by an averaging based on the weights $w(x,y)$ of equation (22). Each patch $p_i = p_{x_i}(f)$ is sparsely approximated as

$$p_i = Dw_i \qquad \text{where} \qquad w_i(j) = \xi(x_i, x_j). \tag{24}$$

The width σ controls the sparsity of the expansion in a manner similar to s in the sparse optimization (7). The computation of the expansion (24) can be interpreted as a crude 1-step matching pursuit that compute at once all the weights to approximate p_i. When the parameter σ tends to 0, the non-local sparse coding is equivalent to the best match (21), because for $\sigma = 0^+$,

$$w_i(j) = \begin{cases} 1 & \text{if} \quad x_j = \gamma(x_i), \\ 0 & \text{otherwise.} \end{cases} \tag{25}$$

Figure 6, left and center, shows the comparison of the texture synthesis with $\sigma = 0$ and $\sigma > 0$. Another option to perform texture synthesis is to replace the reconstruction step 4 of table 2 by the non-local mean weighted average 22. This alternate synthesis algorithm corresponds to the iteration of the non-local means filtering $f \leftarrow \mathcal{N}_{\tilde{f}}(f)$ starting from a random noise. The resulting synthesis shown on Figure 6, right, is noisier than the synthesis with the sparse texture model. This is because the denoising effect of the non-local means reconstruction is only achieved by using a larger value of $\sigma > 0$

Brox and Cremers [10] propose an iterated version of non-local means process to solve the related fixed point equation $f = \mathcal{N}_f(\tilde{f})$. This iterated process is relevant for denoising problems since it enforces the denoised image to use its own patches to do the averaging. This is however different from our synthesis iterations.

$\sigma = 0$ $\sigma = 0.15$ NL-means, $\sigma = 0$

Figure 6: *Left: synthesis using $\sigma = 0$ (perfect recopy, sparse coding using (21)). Center: synthesis using a larger value for $\sigma > 0$ (averaged recopy, sparse coding using (24)). Right: synthesis using iterations of the non-local means filter $f \leftarrow \mathcal{N}_{\tilde{f}}(f)$.*

3.4 Application to Texture Inpainting

The inpainting problem consists in filling a set Ω of missing pixels of a given image $\tilde{f}$. The region $\Omega \subset \{0, \ldots, \sqrt{N}-1\}^2$ might represent damaged pixels of an old photograph or some object to erase to achieve a special effect. The sparse texture model is used to perform inpainting by modifying the algorithm of table 2. At each iteration, the reconstruction only modifies the missing pixels in Ω, leaving the known pixels unchanged. The resulting algorithm is detailed in table 4.

Table 4 Image inpainting algorithm.

1. *Initialization:* Set as initial inpainted image f the original $\tilde{f}$ with values at random inside Ω.
2. *Update the dictionary:* The dictionary is computed from the patches of f: $D \leftarrow \Phi(f)$.
3. *Analysis:* For each $x \in \Omega$, compute the best fitting patch $\gamma(x)$ using equation (21).
4. *Synthesis:* The value of each pixel is replaced using the pseudo-inverse (3) restricted to un-known pixels in Ω:
$$\forall\, x \in \Omega, \quad f(x) = \sum_{|x-y| \leqslant \tau, x+\gamma(y)-y \notin \Omega} p_{\gamma(y)}(x-y)$$
5. *Stop:* While not converged, go back to step 2.

Figure 7 shows some steps of this inpainting process and figure 8 shows additional results and a comparison with [16]. The approach of Criminisi et al. [16] explicitly favors the recopy of salient structures by progressively filling-in the missing pixels. In contrast, our method does not enforce any priority and process all the pixels in parallel. It seems to give similar or better results over homogeneous textured areas, but tends to give poor results if Ω intersects a broad range of different structures.

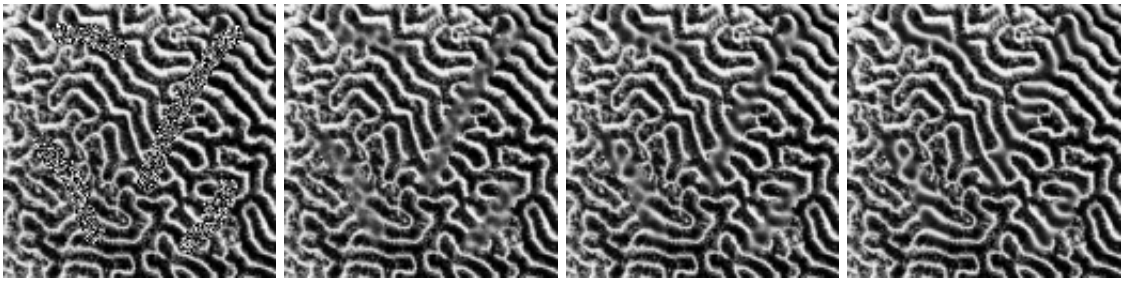


Figure 7: *Evolution of the inpainting process.*

4 Dictionary Learning for Synthesis

In the previous section, the dictionary D was obtained by selecting all the patches $p_x(\tilde{f})$ from some given exemplar $\tilde{f}$. This approach is related to computer graphics methods and does not leads to a compact model for textures, which could be useful beyond the problem of strict recopy of texture. For example, texture classification, texture mixing and modification require an efficient texture model to reach good performances. The underlying problem is the learning of this sparse representation to achieve both good approximation of the original texture and good generalization to capture patterns slightly different from the input exemplar.

4.1 Learning the Dictionary

Image compression, denoising and even synthesis is most often performed using a fixed dictionary D such as for instance wavelets or Gabor atoms. Such processing can be enhanced by learning a dictionary D to sparsify a set $P = \{p_i\}_i$ of typical texture patches. This set of patches is extracted from some input exemplar $\tilde{f}$ so that $p_i = p_{x_i}(\tilde{f})$, where $\{x_i\}_{i=0}^{N-1}$ is the set of pixel locations. This set of input patches is conveniently stored in a matrix $P = \Phi(\tilde{f}) \in \mathbb{R}^{n \times N}$ as detailed in section 2.1.

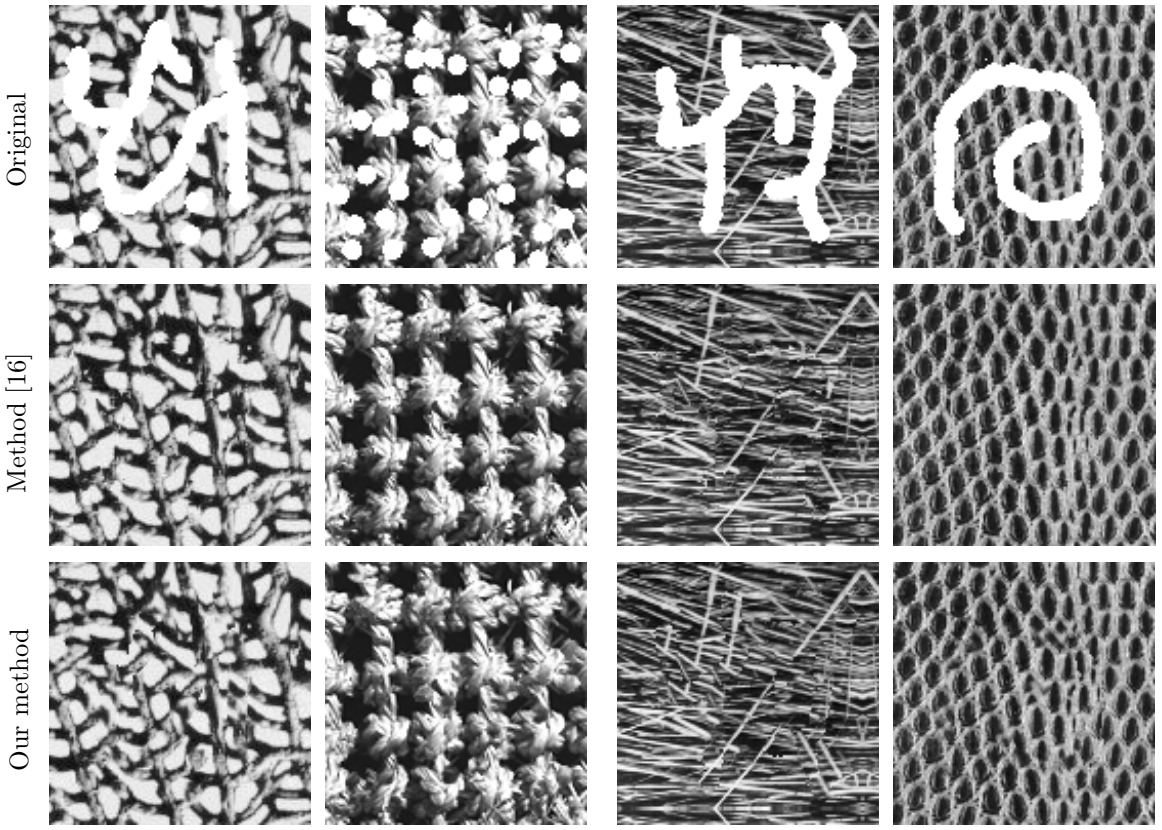


Figure 8: *Examples of inpainting using [16] and using the proposed method.*

Table 5 Dictionary learning for minimizing (29).

1. *Initialization:* set D as a random matrix with unit norm columns.
2. *Sparse coding:* for each exemplar p_i, solve for the coefficients w_i by optimizing
$$w_i \leftarrow \underset{w \in \mathbb{R}^m}{\operatorname{argmin}} \|p_i - Dw\|_{\ell^2} \quad \text{subject to} \quad \|w\|_{\ell^0} \leqslant s. \tag{26}$$
This optimization can be solved approximately using the matching pursuit algorithm, table 1.
3. *Dictionary update:* the dictionary D is updated using either a MOD update [21] or a K-SVD update [2].
MOD update: D is computed as a linear over-determined best fit
$$D \leftarrow PW^+ \qquad \text{where} \qquad W^+ = (W^{\mathrm{T}}W)^{-1}W^{\mathrm{T}} \tag{27}$$
K-SVD update: each atom d_k is updated once at a time. Let $I_k = \{i \setminus w_i(k) \neq 0\}$ the signals using atom d_k. The atom d_k and its coefficients $w_{\cdot}(k) = \{w_i(k)\}_i$ are updated according to
$$(d_k, w_{\cdot}(k)) = \underset{g,a}{\operatorname{argmin}} \sum_{j \in I_k} \|\tilde{p}_j - a_j g\|_{\ell^2} \qquad \text{where} \qquad \tilde{p}_j = p_j - \sum_{\ell \neq k} w_\ell(j) d_\ell. \tag{28}$$
This minization is equivalent to a rank-1 approximation of the matrix containing the signals $\tilde{p}_j$, which can be solved with an SVD.
4. *Normalization:* set for all k, $d_k \leftarrow d_k / \|d_k\|_{\ell^2}$.
5. *Stop:* while not converged, go back to 2.

An optimal D is selected by requiring that the synthesis algorithm works optimally when initialized with the exemplar $\tilde{f}$. It means that D should minimize the synthesis energy $E_D(\tilde{f})$ introduced in equation (9). This leads to a search for the dictionary D that solves the following optimization problem

$$\min_{D\in\mathbb{R}^{n\times m}, W\in\mathbb{R}^{m\times N}} \|P - DW\|_{\ell^2} \quad \text{subject to} \quad \begin{cases} \forall\, i,\ \|w_i\|_{\ell^0} \leqslant s, \\ \forall\, k,\ \|d_k\|_{\ell^2} = 1, \end{cases} \tag{29}$$

where $P = \Phi(\tilde{f}) = \{p_i\}_i$ is the set of patch exemplars. In this optimization problem, each w_i is a column of W that store the coefficients of an exemplar patch p_i and each column vector d_k is a unit norm atom of D. The energy minimized in (29) is the same as the one in (9), but this time the image $\tilde{f}$ is fixed and the optimization is performed on (D, W).

The optimization problem (29) is both non-smooth and non-convex in (D, W) and one can compute a stationary point of this energy using either the MOD algorithm [21] or K-SVD [2]. Table 5 details both algorithms. In practice, both K-SVD and MOD iterations give similar results for synthesis. The iterations of K-SVD are usually faster to converge due to the sequential update of the atoms.

Figure 9 (right) shows an example of dictionary learned with this iterative algorithm. When one applies the learning stage to patches from a single homogenous texture, patterns from the original texture emerge in the trained dictionary.

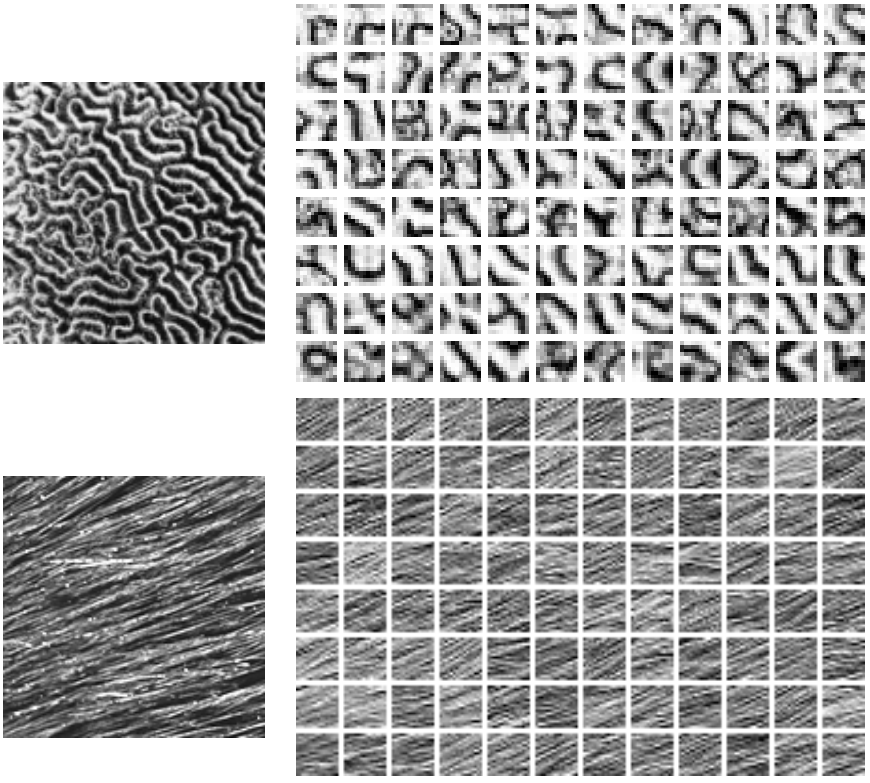
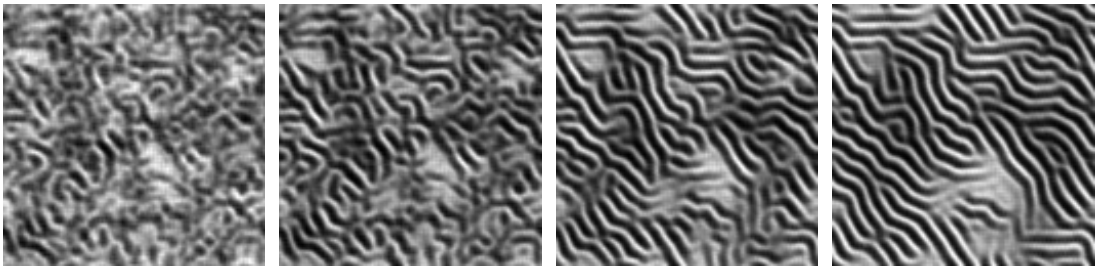


Figure 9: *Left: an input texture $\tilde{f}$, right: the dictionary D learned from this texture.*

4.2 Sparse Texture Synthesis

Starting from some input texture $\tilde{f}$, the algorithm of table 5 is used to learn a dictionary optimized to approximate the patches $\Phi(\tilde{f})$. The algorithm of table 2 is then used to perform the synthesis of a new texture f whose patches are sparse in D. Figure 10 shows some iterations of the synthesis algorithm using a fixed sparsity.

This texture synthesis algorithm depends on two parameters



Iter. #1 Iter. #2 Iter. #4 Iter. #20

Figure 10: *Iteration of the synthesis process for $s = 2$.*

- The redundancy $r = m/n$ of the dictionary. More redundancy provides more geometric fidelity during the synthesis since patches of the original texture $\tilde{f}$ is better approximated in D. In contrast, using a small m leads to a compact texture model that compresses the geometric patterns of the original texture with a few atoms. Such a model allows good generalization performance for task such as texture discrimination or classification when the data to process is unknown but close to f.
- The sparsity $s \geqslant 1$ of the patch expansion. Increasing the sparsity s is a way to overcome the limitations inherent to a compact dictionary (low redundancy m/n) by providing more complex linear combination. In contrast, for very redundant dictionaries (such as the non-local expansion presented in section 3) one can even impose that $s = 1$. Increasing the sparsity also allows to have blending of features and linear variations in intensity that leads to slow illumination gradients not present in the original texture.

Figure 11 shows the influence of both parameters.

Features of various sizes can be captured using the multiscale synthesis algorithm presented in table 3. Note that this synthesis algorithm implicitly considers a set D_j of highly redundant dictionaries at various resolution. Other approaches have been proposed to learn a multiscale dictionary, see for instance [46, 36].

4.3 Application to Texture Mixing

Texture mixing consists in synthesizing a texture f that blends seamlessly the geometric structures of two input exemplars f_1 and f_2. A dictionary D is learned to sparsify the patches of both f_1 and f_2. The set of patches is defined as the concatenation of the two patch matrices $P = (P_1, P_2)$ where $P_i = \Phi(f_i)$. Note that this is different from computing the union $\tilde{D} = (D_1, D_2)$ of two dictionaries trained independently on each texture, which leads to poor results because no atoms in $\tilde{D}$ is able to mix features of both textures.

The algorithm 12 uses D to perform a synthesis that mixes the features of both f_1 and f_2. The set of constraints $\mathcal{C}$ is defined using the histogram from pixels of both f_1 and f_2. Using a sparsity $s > 1$ helps to blend the features of the two textures together.

4.4 Application to Sparse Texture Classification

Our sparse texture model can be used to segment a given texture f into components corresponding to patterns similar to exemplars $\{f^1, \ldots, f^L\}$. A texture dictionary D^ℓ is learned for each texture f^ℓ using the algorithm of table 5 with patches $\Phi(f^\ell)$. The texture f to analyze is modified according to each dictionary D^ℓ. This corresponds to the synthesis of a projected texture f_ℓ by applying the synthesis algorithm of table 2 with f as the initial texture (in step 1, the random noise is replaced by f).

The goodness of fit of the texture model of class ℓ around a pixel x is measured by the projection error

$$\tilde{A}_\ell(x) = |f(x) - f_\ell(x)|^2. \tag{30}$$

Since the boundary separating the textured regions are assumed to be smooth, the classification error is reduced by considering a smoothed projection error

$$A_\ell = \tilde{A}_\ell * G_{\sigma_0} \tag{31}$$

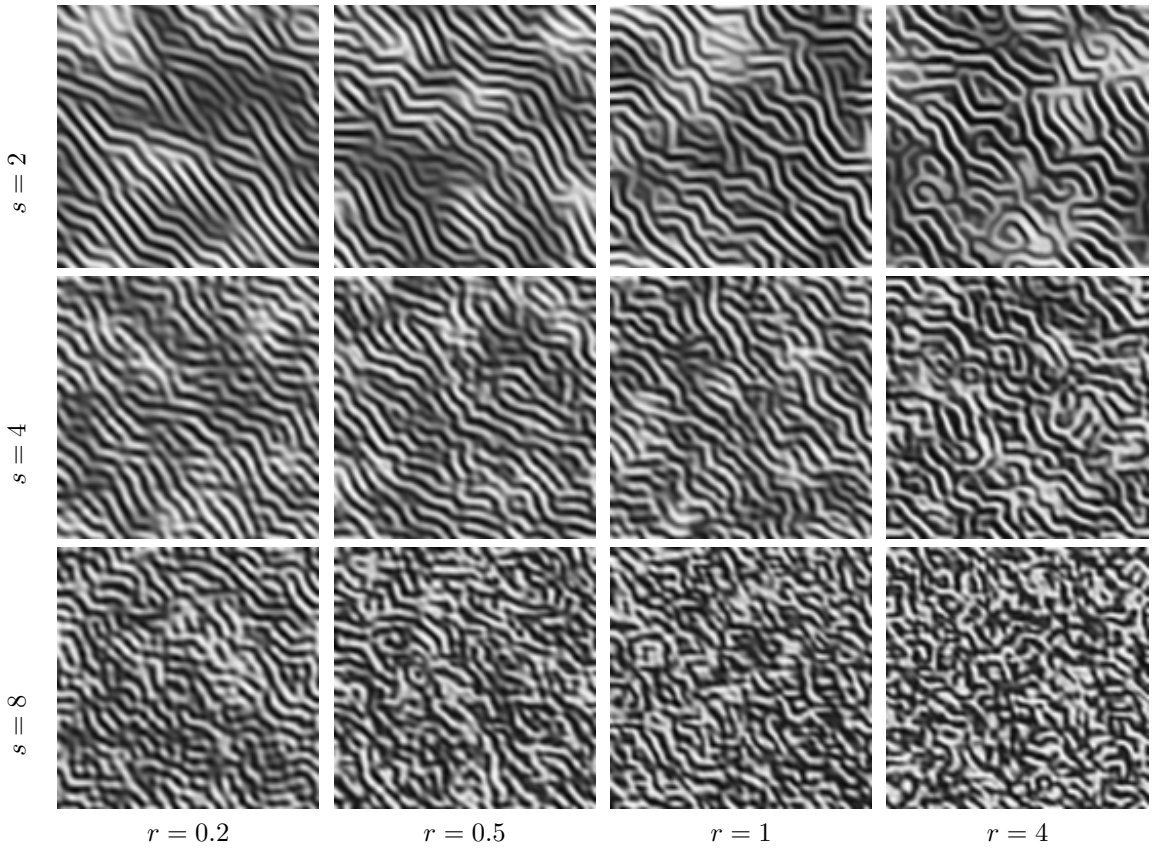


Figure 11: *Influence of the redundancy $r = m/n$ and sparsity s.*

where G_{σ_0} is a 2D Gaussian filter of width σ_0. The parameter σ_0 should be adapted to match the smoothed of the boundary of the regions to reduce as much as possible the classification error.

The classification $\ell(x) \in \{1, \ldots, L\}$ of a pixel x is defined as

$$\ell(x) = \underset{\ell \in \{1,\ldots,L\}}{\operatorname{argmin}} \; A_\ell(x). \tag{32}$$

Table 6 summarize the main step of the segmentation algorithm.

Figure 13 shows the result of classification on a set of $L = 5$ exemplar textures. The input image f of 256×256 pixels is a patchwork of L textures extracted from large images. The exemplars f^ℓ are similar textures extracted from the same set of images but at other locations.

Table 6 Texture classification algorithm.

1. *Learning:* the texture dictionary D^ℓ is learned for each class $\ell = 1, \ldots, L$.
2. *Projection:* compute f_ℓ by texture synthesis with dictionary D^ℓ applied to f.
3. *Error:* for each ℓ compute A_ℓ as defined using (31)..
4. *Classification:* compute the classifier $\ell(x)$ using (32).

4.5 Texture Signature Model

The dictionary learning procedure detailed in section 4.1 allows one to parameterize the texture model with a single dictionary $D \in \mathbb{R}^{n \times m}$ learned from an input exemplar $\tilde{f}$. This representation, while being efficient, is not particularly compact since the original exemplar $\tilde{f}$ of N pixels is replaced by a matrix of size $n \times m$ which becomes large when the redundancy of the dictionary m/n increases.

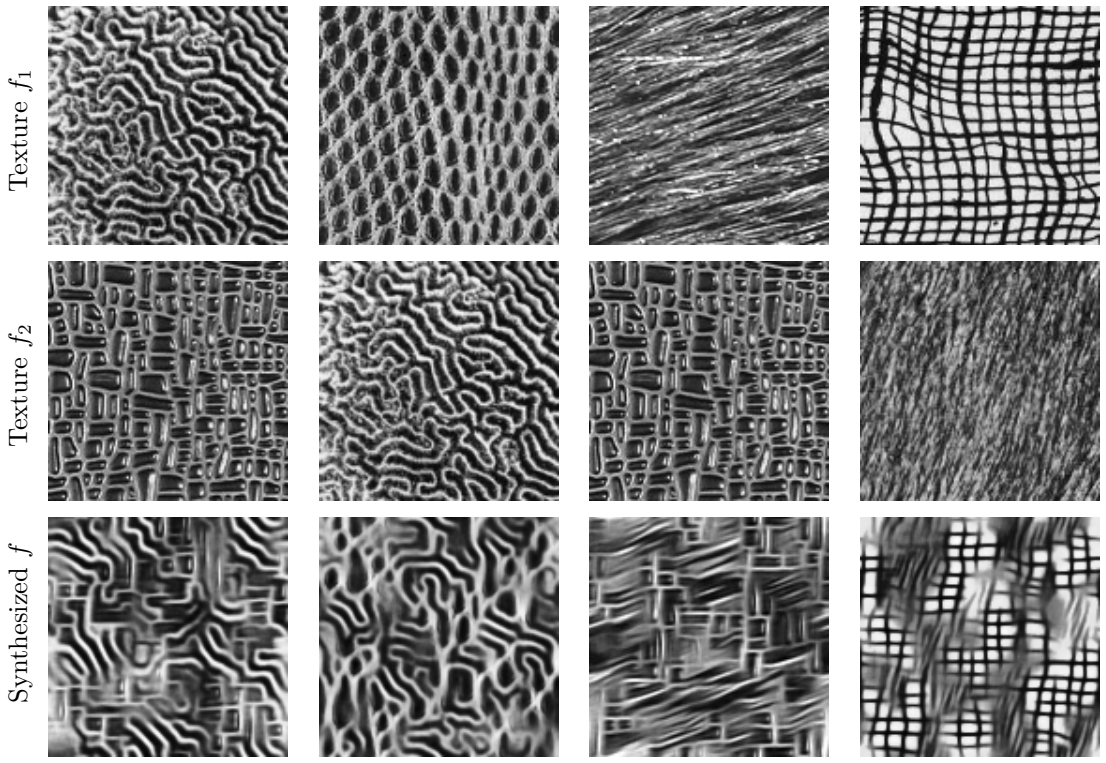


Figure 12: *Examples of texture mixing with sparsity $s = 4$.*

This increase of dimensionality is reduced by taking into account the spatial relationship between the patches $p_x(\tilde{f})$ extracted from the input exemplar. This is achieved by assuming that each atom d_i in the dictionary is a patch $d_i = p_{x_i}(F)$ extracted from a image signature dictionary $F \in \mathbb{R}^m$ of $\sqrt{m} \times \sqrt{m}$ pixels.

This image signature dictionary has been proposed by Aharon and Elad [1]. A similar procedure has been introduced in computer graphics [23], but it is restricted to a strict sparsity $s = 1$. An early approach to solve a similar problem of vector quantization is [14].

Using the patch extraction operator Φ defined in equation (2.1), a dictionary $D = \Phi(F) \in \mathbb{R}^{n \times m}$ is optimized by minimizing (29)

$$\min_{F \in \mathbb{R}^m, W \in \mathbb{R}^{m \times N}} \frac{1}{2} \|P - \Phi(F)W\|_{\ell^2} \quad \text{subject to} \quad \begin{cases} \forall\, i,\ \|w_i\|_{\ell^0} \leqslant s, \\ \forall\, k,\ \|d_k\|_{\ell^2} = 1. \end{cases} \tag{33}$$

where P stores the set of exemplar patches. If the size m of the signature F is smaller than N, the resulting texture ensemble is parameterized with a reduced set of parameters. The procedure introduced by Aharon and Elad [1] minimizes iteratively (33) using a block coordinate descent, see table 7.

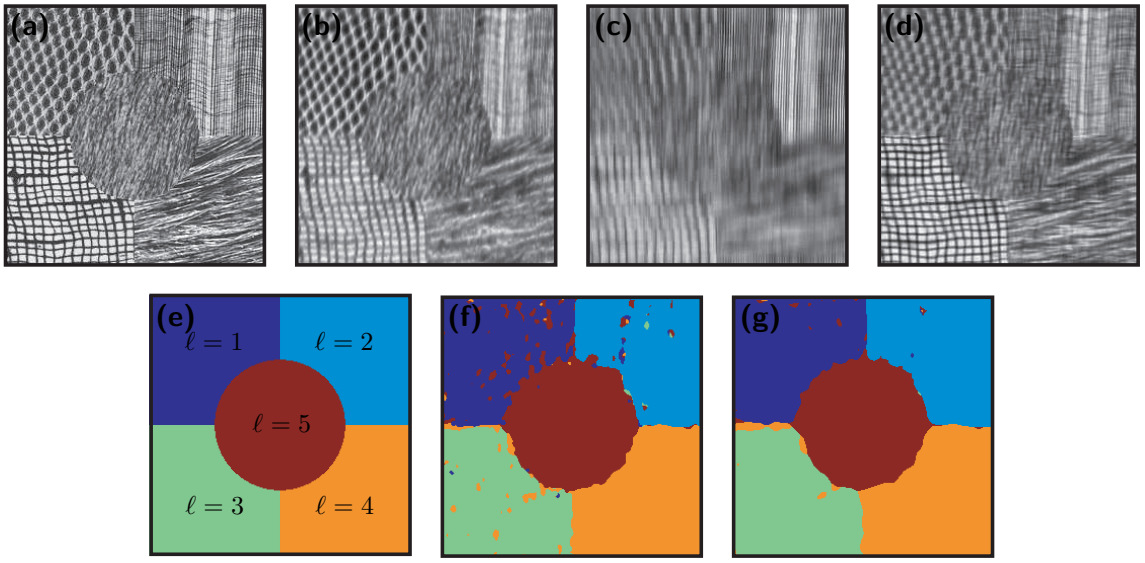


Figure 13: *(a) Original texture f. (b) Projected texture f_1 of f. Note how the upper left corner is well preserved. (c) Projected texture f_2. (d) Projected texture f_3. (e) Ground trust classification. (f) Classification $\ell(x)$ computed with $\sigma_0 = 3$ pixels. (g) Classification computed with $\sigma_0 = 6$ pixels.*

Table 7 Image signature learning algorithm.

1. *Initialization:* set $F \leftarrow$ random.
2. *Dictionary extraction:* define $D = \Phi F$.
3. *Sparse coding:* compute W by optimizing

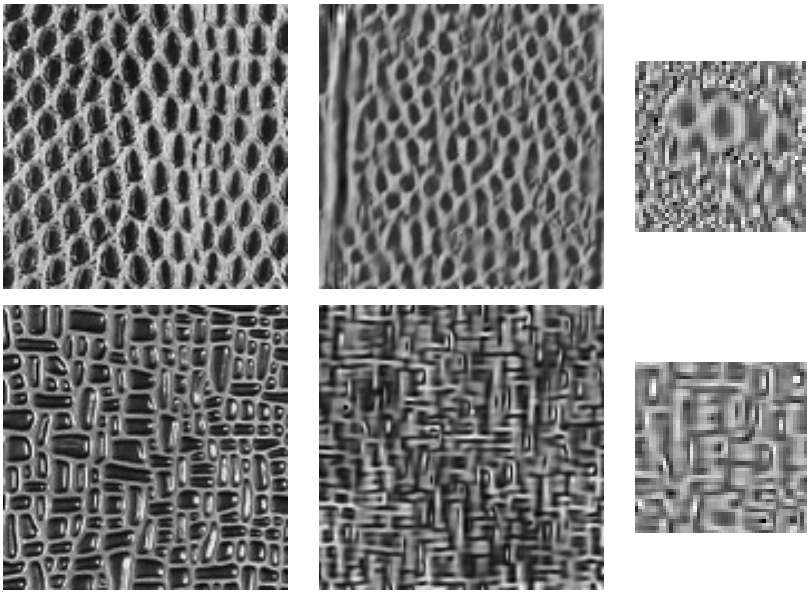
$$w_i \leftarrow \underset{w \in \mathbb{R}^m}{\operatorname{argmin}} \|p_i - Dw\|_{\ell^2} \quad \text{subject to} \quad \|w\|_{\ell^0} \leqslant s. \tag{34}$$

   which can be solved approximately with matching pursuit [37].
4. *Dictionary update:* update the dictionary D using either the MOD update or the K-SVD update (see step 3 of table 5).
5. *Signature update:* Compute $F \leftarrow \Phi^+ D = \frac{1}{n}\Phi^{\mathrm{T}} D$.
6. While not converged, go back to 2.

Once a compact image signature F has been learned with this algorithm, the dictionary $D = \Phi(F)$ is used to perform texture synthesis with the iterative method exposed in section 2.3. Figure (14) shows examples of synthesis using image signature dictionaries.

Conclusion

This paper describes a model to capture the geometric structures of homogeneous textures. This model is based on a sparse expansion of the patches of the texture into a redundant dictionary. Classical texture synthesis methods are equivalent to a strict sparsity of $s = 1$ into the highly redundant dictionary of all the patches from some input exemplar. More refined models are learned by optimizing a dictionary that sparsify a set of input patches. Such a compressed representation can be tuned to achieve an increasing fidelity to the exemplar and is useful to perform texture mixing and texture classification.



Original $\tilde{f}$ Synthesized f Signature F

Figure 14: *Examples of texture synthesis using an image signature dictionary.*

References

[1] M. Aharon and M. Elad. Sparse and redundant modeling of image content using an image-signature-dictionary. *SIAM Journal on Imaging Sciences*, 1(3):228–247, July 2008.

[2] M. Aharon, M. Elad, and A.M. Bruckstein. The k-svd: An algorithm for designing of overcomplete dictionaries for sparse representation. *IEEE Trans. On Signal Processing*, 54(11):4311–4322, 2006.

[3] M. Ashikhmin. Synthesizing natural textures. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 217–226, New York, NY, USA, 2001. ACM Press.

[4] C. Ballester, V. Caselles, and J. Verdera. Disocclusion by joint interpolation of vector fields and gray levels. *SIAM Multiscale Modeling & Simulation*, 2(1):80–123, 2003.

[5] Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman. Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):120–135, 2001.

[6] A. J. Bell and T. J. Sejnowski. The independent components of natural scenes are edge filters. *Vision Research*, (37):3327–3338, 1997.

[7] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proc. of Siggraph 2000*, pages 417–424, 2000.

[8] J. S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proc. of SIGGRAPH '97*, pages 361–368. ACM Press/Addison-Wesley Publishing Co., 1997.

[9] A. C. Bovik. Analysis of multichannel narrow-band filters for image texture segmentation. *IEEE Transactions on Signal Processing*, 39(9):2025, 1991.

[10] T. Brox and D. Cremers. Iterated nonlocal means for texture restoration. In *Proc. International Conference on Scale Space and Variational Methods in Computer Vision*, LNCS, Ischia, Italy, May 2007. Springer.

[11] A. Buades, B. Coll, and J. M. Morel. A review of image denoising algorithms, with a new one. *SIAM Multiscale Modeling & Simulation*, 4(2):490–530, 2005.

[12] S. S. Chen, D.L. Donoho, and M.A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1998.

[13] D. A. Clausi and M. E. Jernigan. Designing gabor filters for optimal texture separability. *Pattern Recognition*, 33(11):1835–1849, November 2000.

[14] Laurent D. Cohen. A new approach of vector quantization for image data compression and texture detection. *In International Conference on Pattern Recognition*, pages II: 1250–1254, 1988.

[15] R.L. Cook and T. DeRose. Wavelet noise. *ACM Trans. Graph.*, 24(3):803–811, 2005.

[16] A. Criminisi, P. Pérez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing*, 13(9):1200–1212, 2004.

[17] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Comm. Pure Appl. Math*, 57:1413–1541, 2004.

[18] D. Donoho. Wedgelets: Nearly-minimax estimation of edges. *Ann. Statist*, 27:353–382, 1999.

[19] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, page 1033. IEEE Computer Society, 1999.

[20] A.A. Efros and W.T. Freeman. Image quilting for texture synthesis and transfer. *Proc. of SIGGRAPH 2001*, pages 341–346, August 2001.

[21] K. Engan, S. O. Aase, and J. Hakon Husoy. Method of optimal directions for frame design. In *Proc. ICASSP '99*, pages 2443–2446, Washington, DC, USA, 1999. IEEE Computer Society.

[22] M.J. Fadili, J.-L. Starck, and F. Murtagh. Inpainting and zooming using sparse representations. *The Computer Journal*, 2006. Revised.

[23] J. Han, L. Wei, K. Zhou, H. Bao, B. Guo, and H-Y. Shum. Inverse texture synthesis. *ACM Trans. Graph*, 27(3), 2008.

[24] D. J. Heeger and J. R. Bergen. Pyramid-Based texture analysis/synthesis. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 229–238. ACM SIGGRAPH, Addison Wesley, August 1995.

[25] A. K. Jain and F. Farrokhnia. Unsupervised texture segmentation using gabor filters. *Pattern Recogn.*, 24(12):1167–1186, 1991.

[26] B. Julesz. Visual pattern discrimination. *IRE Trans. Inform. Theory*, 8(2):84–92, 1962.

[27] K. Kreutz-Delgado, J. F. Murray, B. D. Rao, K. Engan, T-W. Lee, and T.J. Sejnowski. Dictionary learning algorithms for sparse representation. *Neural Comput.*, 15(2):349–396, 2003.

[28] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. Texture optimization for example-based synthesis. *ACM Transactions on Graphics, Proc. of SIGGRAPH 2005*, 24(3):795–802, August 2005.

[29] V. Kwatra, A. Schdl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003*, 22(3):277–286, July 2003.

[30] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems 13*, Cambridge, MA, 2001. MIT Press.

[31] S. Lefebvre and H. Hoppe. Parallel controllable texture synthesis. *ACM Trans. Graph.*, 24(3):777–786, 2005.

[32] S. Lefebvre and H. Hoppe. Appearance-space texture synthesis. *ACM Trans. Graph.*, 25(3):541–548, July 2006.

[33] M. S. Lewicki and T. J. Sejnowski. Learning overcomplete representations. *Neural Comput.*, 12(2):337–365, 2000.

[34] S. Y. Lu, J. E. Hernandez, and G. A. Clark. Texture segmentation by clustering of Gabor feature vectors. In *Proc. IJCNN'91, International Joint Conference on Neural Networks*, volume I, pages 683–688, Piscataway, NJ, 1991. IEEE; Int. Neural Network Soc, IEEE Service Center.

[35] J. Mairal, M. Elad, and G. Sapiro. Sparse representation for color image restoration. *IEEE Trans. Image Processing*, 17(1):53–69, January 2008.

[36] J. Mairal, G. Sapiro, and M. Elad. Learning multiscale sparse representations for image and video restoration. *Preprint*, 2007.

[37] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, San Diego, 1998.

[38] R. Manduchi and J. Portilla. Independent component analysis of textures. In *ICCV*, pages 1054–1060, 1999.

[39] W. Matusik, M. Zwicker, and F. Durand. Texture design using a simplicial complex of morphable textures. *ACM Trans. Graph.*, 24(3):787–794, 2005.

[40] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive-field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, June 1996.

[41] N. Paragios and R. Deriche. Geodesic active regions and level set methods for supervised texture segmentation. *International Journal of Computer Vision*, 46(3):223–247, February 2002.

[42] K. Perlin. An image synthesizer. In *Proc. of SIGGRAPH '85*, pages 287–296, New York, NY, USA, 1985. ACM Press.

[43] G. Peyré. Non-negative sparse modeling of textures. *Proc. of SSVM'07*, pages 628–639, 2007.

[44] G. Peyré. Manifold models for signals and images. *to appear in Computer Vision and Image Understanding*, 2008.

[45] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Comput. Vision*, 40(1):49–70, 2000.

[46] P. Sallee and B.A. Olshausen. Learning sparse multiscale image representations. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *NIPS*, pages 1327–1334. MIT Press, 2002.

[47] E. P. Simoncelli and B. A. Olshausen. Natural image statistics and neural representation. *2001*, 24:1193–1215, Annual Review of Neuroscience.

[48] K. Skretting and J.H. Husoy. Texture classification using sparse frame based representations. *EURASIP Journal on Applied Signal Processing, to appear*, 2006(1):102–102, 2006.

[49] J.-L. Starck, M. Elad, and D.L. Donoho. Redundant multiscale transforms and their application for morphological component analysis. *Advances in Imaging and Electron Physics*, 132, 2004.

[50] J. A. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004.

[51] J. A. Tropp. Just relax: convex programming methods for identifying sparse signals in noise. *IEEE Transactions on Information Theory*, 52(3):1030–1051, 2006.

[52] D. Tschumperlé and R. Deriche. Vector-valued image regularization with PDEs: A common framework for different applications. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(4):506–517, April 2005.

[53] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109(3):475–494, 2001.

[54] M. Tuceryan. Moment-based texture segmentation. *Pattern Recognition Letters*, 15(7):659–668, July 1994.

[55] L-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488. ACM Press/Addison-Wesley Publishing Co., 2000.

[56] X-Y. Zeng, Y-W. Chen, D. van Alphen, and Z. Nakao. Selection of ICA features for texture classification. In *ISNN (2)*, pages 262–267, 2005.

[57] S. C. Zhu, X. W. Liu, and Y. N. Wu. Exploring texture ensembles by efficient markov chain monte carlo-toward a 'trichromacy' theory of texture. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(6):554–569, June 2000.

[58] S. C. Zhu, Y. Wu, and D. Mumford. Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling. *Int. J. Comput. Vision*, 27(2):107–126, 1998.