PatchWorks: Example-Based Region Tiling for Image Editing

Patrick Pérez, Michel Gangnet, Andrew Blake

January 13, 2004

Technical Report
MSR-TR-2004-04

Filling regions of an image in a seamless manner is an important task for editing digital images. We propose to address it by using an example-based image generation technique. Such techniques have recently proved very effective in the context of unconstrained texture synthesis. We show that they can also be surprisingly effective to fill large regions in images, even within complex mixes of textures and structures. We combine this approach with a block-based architecture within an interactive image editing tool, named *PatchWorks*. This tool performs faster and better than earlier pixel-based counterparts, and faster than most recent related techniques with similar capabilities. The ease of use and versatility of our system is illustrated on a number of real image editing tasks.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
`http://www.research.microsoft.com`

1 Introduction

Filling regions of an image in a seamless manner is important both for photograph and movie editing. It is indeed a key tool for images restoration (repairing images physically damaged by tears, stains, scratches, writing, etc.), correction (fixing undesirable artifacts such as red-eyes, flash reflections, blemish, wrinkles, etc., and removing unwanted occluding objects such as wires, persons, etc.), and manipulation (producing photo-montages and playing visual tricks). One real example for each of these types of image editing tasks is presented in Fig. 1, along with the results obtained effortless with the tool presented in this report.

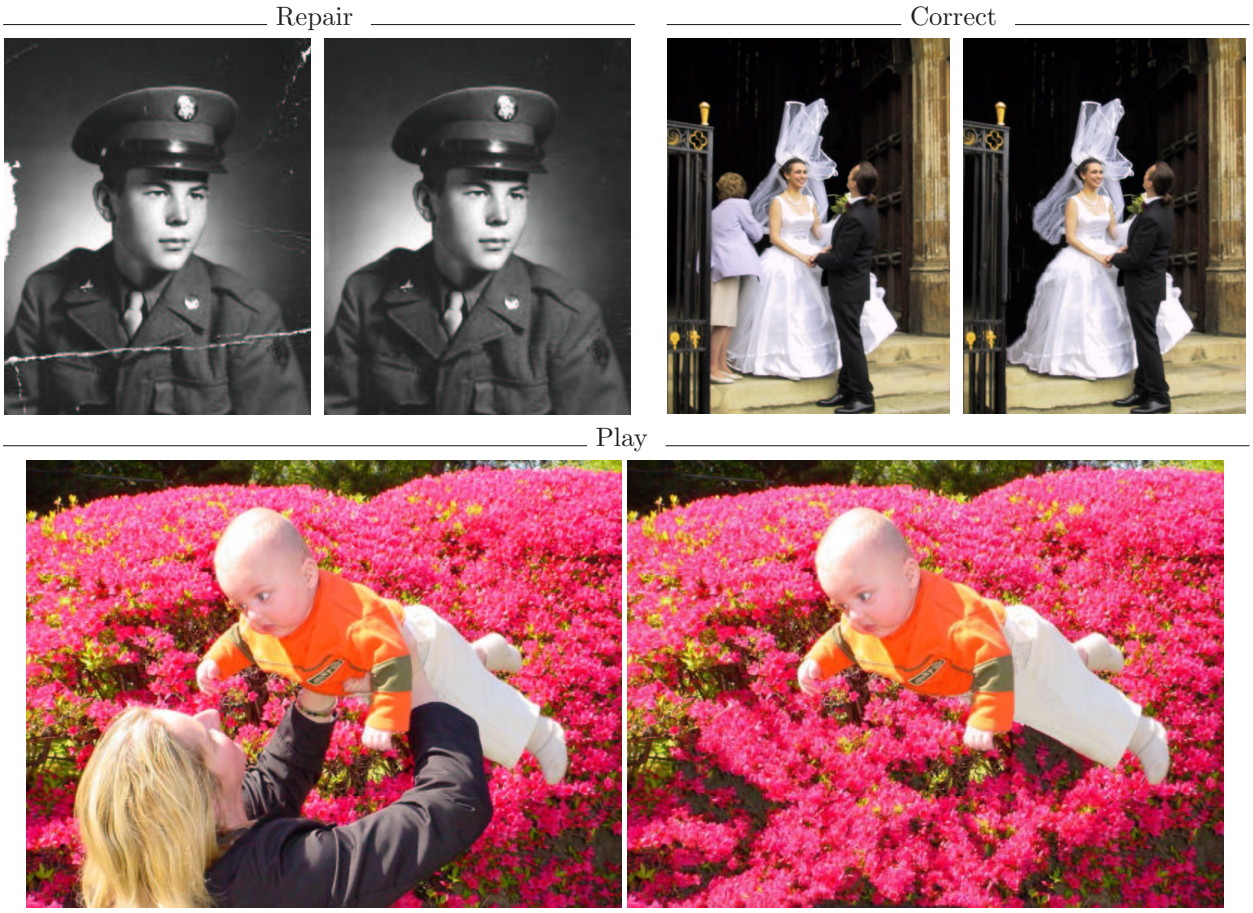


Fig 1. Three examples of image editing tasks using PatchWorks: digital repairing of damaged old photographs, contents correction by removing undesirable "objects", and visual effects. All these tasks require to re-invent plausible image patches to fill-in regions manually selected by the user.

A natural way to address the task of region filling in images is to resort to classic interpolation techniques: the boundary values along the border of the destination region are interpolated inside, based on a certain smoothness criterion. The most

sophisticated and successful attempts along those lines are those first proposed in (Bertalmio, Sapiro, Caselles & Ballester 2000). The so-called *Inpainting* approach is a PDE-based framework in which different diffusion schemes are interleaved to iteratively extend, as smoothly as possible, the isophotes arriving at the region boundary (Bertalmio et al. 2000, Bertalmio, Bertozzi & Sapiro 2001, Ballester, Bertalmio, Caselles, Sapiro & Verdera 2001*a*, Ballester, Caselles, Verdera, Bertalmio & Sapiro 2001*b*). The authors report impressive results of removing scratches or text overlays from photographs. By design, their system is good at bridging small gaps in rather texture-free regions. Unfortunately, the technique does not seem suitable in terms of both synthesis versatility and computational cost for a number of image manipulation tasks where the destination regions can be large, and the surrounding image highly textured.

As for synthesis capabilities, image-based (also known as example-based or non-parametric [NP]) image generation techniques appear as an appealing alternative to variational interpolation. Indeed, since the seminal works by Efros & Leung (1999) and Freeman, Pasztor & Carmichael (1999), a common and extremely simple framework has been developed to tackle with impressive success a number of tasks: unconstrained texture re-synthesis (Efros & Leung 1999, Guo, Shum & Xu 2000, Wey & Levoy 2000, Bar-Joseph, El-Yaniv, Lischinski & Werman 2001, Efros & Freeman 2001, Harrison 2001, Hertzmann, Jacobs, Oliver, Curless & Salesin 2001, Ashikhmin 2001, Liang, Liu, Xu, Guo & Shum 2001, Kwatra, Schödl, Essa, Turk & Bobick 2003), texture transfer (Ashikhmin 2001, Efros & Freeman 2001, Hertzmann et al. 2001), texture-by-numbers (Ashikhmin 2001, Efros & Freeman 2001, Harrison 2001, Hertzmann et al. 2001), generic super-resolution (Freeman et al. 1999, Freeman, Pasztor & Carmichael 2000, Hertzmann et al. 2001), and object-specific super-resolution (Baker & Kanade 2000). The work on so-called *Image Analogies* by Hertzmann et al. (2001) unifies elegantly all these techniques and covers all the above-mentioned applications (as well as artistic style transfer).

As soon as (Efros & Leung 1999), the specific task of filling regions in images with such techniques is mentioned as an instance of constrained texture synthesis, i.e., synthesis with boundary conditions, and very briefly illustrated on a single mono-texture example. In (Wey & Levoy 2000, Harrison 2001, Liang et al. 2001), the task of region filling is similarly mentioned as an aside with a single experimental illustration on a mono-texture example.[1] To our knowledge the work of Bornard, Lecan, Laborelli & Chenot (2002) is the first one to focus specifically on the use of example-based synthesis techniques à la Efros & Leung (1999) to address the region filling problem. This pixel-based technique is shown to give very good results at low cost on rather narrow regions corresponding to scratches, thin structures, or text overlays. More recently, sophisticated techniques have been proposed to ensure a robust NP filling of *large* image regions surrounded by multiple textures and structures (Bertalmio, Vese, Sapiro & Osher 2003, Criminisi, Pérez & Toyama

[1] Harrison's work has nonetheless originated an explicit region filling tool in the form the GIMP plug-in called *Resynthesizer*. This tool will be used for comparison purpose in this report.

2003, Drori, Cohen-Or & Yeshurun 2003, Jia & Tang 2003). They all mix classic NP texture synthesis with some form of smooth interpolation: of the low frequencies in (Bertalmio et al. 2003, Drori et al. 2003), of the priority map in (Criminisi et al. 2003), and of the texture segmentation of the surrounding in (Jia & Tang 2003). Their sophistication however comes at a price with computation times ranging from several minutes to several hours for regions of order 100×100 pixels.

In order to get a really versatile interactive image editing tool, good results and reasonable computation times should be obtained on large regions with complex surrounding. This is achieved with the patch-based filling system, named *PatchWorks*, that we present in this report. The salient features of this system are:

- **Patch-based generation**: in the spirit of (Efros & Freeman 2001, Guo et al. 2000, Liang et al. 2001), the image generation is performed on a patch basis, as opposed to pixel-based region filing techniques described in (Efros & Leung 1999, Wey & Levoy 2000, Harrison 2001, Bornard et al. 2002). As we shall show, using patches dramatically speeds-up the process, while often improving the quality. This is a key ingredient to deal with large regions, e.g., larger than 100×100 pixels.

- **Local patch search**: unless specified otherwise by the user, the set of sample patches is gathered in the surrounding of the destination region, similarly to (Bornard et al. 2002), but in contrast with (Efros & Leung 1999, Wey & Levoy 2000, Harrison 2001). This impacts dramatically the speed of the filling process and its quality. When situation requires it, the source of sample patches can also be provided manually, in a form of one or several regions of any size and shape.

- **Inward layered filling**: following the first suggestion of Efros & Leung (1999), concentric layers are generated from the boundary of the destination region inwards, with no need for one of the "highest confidence first" approaches proposed in (Harrison 2001, Bornard et al. 2002, Criminisi et al. 2003, Drori et al. 2003).

The generic algorithm for example-based image generation is presented in Section 2, along with our patch-based version for region filling. First experiments on toy examples will illustrate the merit of using patches and concentric layers.

In Section 3, we demonstrate the power and versatility of the image editing tool thus obtained. In particular, we demonstrate the ability of the algorithm to deal with complex photographs as well as with images of single textures. We conclude this demonstration section by showing complete image editing tasks conducted with several successive uses of *PatchWorks*.

The only downside of patches is the occasional persistence of visible seams regularly organized along the tilling structure. We propose in Section 4 a simple post-processing technique that helps removing photometric and colorimetric seams. It complements the optimal irregular boundary approach proposed by Efros & Freeman (2001) and recently extended by Kwatra et al. (2003), which is more adapted

to removing texture seams. Our technique is based on the idea of slightly modifying the whole content of every other image patch of the final tiling to increase further its compatibility with its neighborhood.

In Section 5, we conclude and indicate further research directions.

2 The algorithm

2.1 Generic example-based image generation

The principle behind example-based image generation is extremely simple and intuitive. Given sample image portions, a new image portion that is visually similar to those in the sample set can be generated in a jigsaw fashion: little pieces taken from the sample pool are stitched together as seamlessly as possible. One step of the generation procedure consists in (i) choosing an empty location at the boundary of the existing synthesized image portion, and (ii) filling it in with a patch of pixels borrowed from the source such as to match well the part of neighborhood that already exists at this location. This way, a complete new image can be generated from scratch within a lexicographical procedure, an existing image can be extended (extrapolating out in a spiraling-out procedure), or a region of an existing image can be filled-in by spiraling inwards from its boundary. In addition to the spatial consistency, complementary information can also influence the choice of the sample piece. When used, such an information is available from the start for all the locations to be synthesized over, and for all the image patches in the data set. In super-resolution, as well as in coarse-to-fine texture synthesis, this information corresponds to a low resolution view of the image at concerned locations. In texture-by-numbers, this information consists of texture labels present both in the source data and in the image to be synthesized. With emphasis on this auxiliary guidance, the *Image Analogies* framework (Hertzmann et al. 2001) expresses its principle nicely: one tries to generate an image J which is to the input auxiliary image $\tilde{J}$ what I is to $\tilde{I}$, where the image pair $(I, \tilde{I})$ defines the sample set.

Let us set up formally the scene, using a patch-based formalism. Sample image I and its counterpart J to be (partly) generated are indexed respectively by sets of pixels Γ and Ω. In case of auxiliary guidance, two other images $\tilde{I}$ and $\tilde{J}$, possibly different in nature from the two previous ones, are available and indexed also by Γ and Ω respectively. At any moment in the process of generating/completing J, a binary mask $M = \{m_{\mathbf{x}}, \mathbf{x} \in \Omega\}$ assigns label 0 to pixel locations still unoccupied, and 1 to the occupied ones.

The elementary building brick is a square patch[2] of pixels defined through a set P of shifts with respect to a reference position, say the upper left corner. The patch can reduce to a singleton, as in most works. A neighborhood defined through the shift set N is also assigned to P. If for example the patch is 2×2 and its neighborhood is constituted of the 12 pixels surrounding it, we have $P = \{0, 1\}^2$ and

[2]It could be any tileable pattern, starting with rectangles.

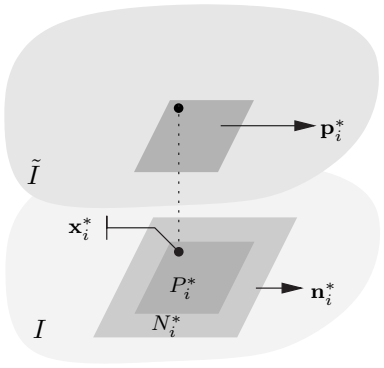


Fig 2. Notations at the source. Main source image I is tiled with patches $\{P_i^*\}$ with origins $\{\mathbf{x}_i^*\}$ and neighborhoods $\{N_i^*\}$. The latter give rise to image value vectors $\{\mathbf{n}_i^*\}$. When an auxiliary image $\tilde{I}$ is also used, an auxiliary image value vector $\mathbf{p}_i^*$ is associated to each patch P_i^*.

$N = \{-1, 0, 1, 2\}^2 - P$. For any location $\mathbf{x}$, the associated patch and neighborhood are respectively $\mathbf{x} + P = \{\mathbf{x} + \mathbf{u}, \mathbf{u} \in P\}$ and $\mathbf{x} + N = \{\mathbf{x} + \mathbf{u}, \mathbf{u} \in N\}$.

A sample dictionary $\mathcal{D} \doteq \{\mathbf{d}_i^*\}_{i=1\ldots n^*}$ is built by considering n^* patches $P_i^* \doteq \mathbf{x}_i^* + P$ at locations $\mathbf{x}_i^*$, $i = 1 \ldots n^*$ such that their neighborhoods $N_i^* \doteq \mathbf{x}_i^* + N$ all belong to the sample support Γ (Fig. 2).

Each entry in the dictionary is a vector formed in the following manner:

$$\mathbf{d}_i^* = [\mathbf{n}_i^*, \mathbf{p}_i^*, \mathbf{x}_i^*, \mathbf{i}_i^*], \tag{1}$$

with:

- $\mathbf{n}_i^*$, the vector of image values $I(N_i^*)$ in the patch neighborhood;[3]
- $\mathbf{p}_i^*$, the vector of auxiliary image values $\tilde{I}(P_i^*)$ in the patch;
- $\mathbf{i}_i^*$, the vector of dictionary indices corresponding to neighboring patches. They are eight possible such neighbors,[4] each of them associated to one shift in the set T. In case of 2×2 patches and 8-connexity, $T = \{-2, 0, 2\}^2$. We denote $\mathbf{i}_i^* = [\gamma_{\mathbf{t}}^*(i)]_{\mathbf{t} \in T}$. For a given patch, some of these neighbors might either not exist at all (border of the sample image support Γ) or not be associated to an entry in the dictionary. In the two cases the corresponding entry in $\mathbf{i}_i^*$ is assigned null index: $\gamma_{\mathbf{t}}^*(i) = 0$ iff $\nexists j : \mathbf{x}_j^* = \mathbf{x}_i^* + \mathbf{t}$.

The support Ω of J is tiled with n patches $P_j \doteq \mathbf{x}_j + P$, $j = 1 \ldots n$. If the patches are not singletons, the tiling $\Omega \subset \cup_{j=1}^n P_j$ can be periodic or not, overlapping or not. Similarly to dictionary entries, a vector $\mathbf{d}_j = [\mathbf{n}_j, \mathbf{p}_j, \mathbf{x}_j, \mathbf{i}_j]$ can be associated to each patch P_j, with a slight twist: for a given state of the generation procedure

[3] Given a set of pixels A, notation $I(A)$ stands for $\{I(\mathbf{x}), \mathbf{x} \in A\}$.

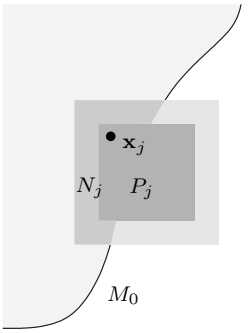
[4] In the general case, this number depends on the geometry of the elementary patch and of its neighborhood.

only part of J image values exist, and $\mathbf{n}_j$ might be only partly populated. This is also th case if P_j falls on the border of the image support Ω. This will be captured by a binary sieve $\mathbf{m}_j$ of same size as $\mathbf{n}_j$, with null entries indicating empty entries in $\mathbf{n}_j$. Finally, at any moment the dictionary index of the patch copied over P_j will be denoted $\alpha(j) \in \{0, 1, \ldots, n^*\}$, with $\alpha(j) = 0$ for patch location not yet filled-in.

Using these notations, a high level synopsis of example-based image generation is proposed below (Algorithm 1). Up to slight twists, this generic description encompasses all works mentioned earlier, irrespective to their application domains. In this synopsis, `search` stands for a procedure that returns a dictionary index i such that $I(N_i^*)$ is "similar" to $J(N_j)$ over the populated part of N_j, and, if auxiliary guidance is used, $\tilde{I}(P_i^*)$ and $\tilde{J}(P_j)$ are similar. Step (a) in Algorithm 1 simply amounts to picking a location whose associated patch is at least partially empty while its neighborhood is at least partially filled.

Algorithm 1 Generic example-based image generation

- Build sample dictionary $\mathcal{D} \doteq \{\mathbf{d}_i^*\}_{i=1\ldots n^*}$.
- Initialize completion mask $M = \{M(\mathbf{x}), \mathbf{x} \in \Omega\}$.
- While $M_0 \doteq \{\mathbf{x} \in \Omega : M(\mathbf{x}) = 0\} \neq \emptyset$ do:
  (a) Pick location j s.t. $P_j \cap M_0 \neq \emptyset$ and $N_j \not\subset M_0$.
  (b) Assemble $\mathbf{d}_j$ and $\mathbf{m}_j$.
  (c) Search: $\alpha(j) \leftarrow \texttt{search}(\mathbf{d}_j, \mathbf{m}_j; \mathcal{D})$.
  (d) Copy: $J(P_j) \leftarrow I(P^*_{\alpha(j)})$, $M(P_j) \leftarrow 1$.



Let us briefly review the different ingredients:

- Elementary patch: apart from unconstrained texture synthesis approaches in (Efros & Freeman 2001, Guo et al. 2000, Liang et al. 2001, Kwatra et al. 2003), and resolution enhancement in (Freeman et al. 1999), all example-based image generation techniques proceed one pixel at a time: P is thus a singleton, and N stands for a few-pixel wide band around (typically 2-10 pixel wide). Note however that at copy time, it is not only the candidate location but also the empty part of its neighborhood that are filled with source materials in (Criminisi et al. 2003, Drori et al. 2003, Jia & Tang 2003).

- Similarity and search: the similarity measure on which the search relies is usually captured by a distance of the form

$$\texttt{dist}(\mathbf{d}_j, \mathbf{d}_i^*; \mathbf{m}_j) = \left(\|\mathbf{n}_j - \mathbf{n}_i^*\|^q_{q,\Delta_n \mathbf{m}_j} + \beta \|\mathbf{p}_j - \mathbf{p}_i^*\|^q_{q,\Delta_p \mathbf{1}} \right)^{1/q} \qquad (2)$$

  where $q = 1$ or 2, Δ_p and Δ_n are two diagonal matrices weighting the contributions of each location in the patch and in the neighborhood respectively

according to its distance to the center of the patch, notation $\|\mathbf{u}\|_{q,\mathbf{w}}$ stands for $(\sum_k w_k|u_k|^q)^{1/q}$, and $\mathbf{1}$ stands for vector with unit entries. Based on this distance which depends on sieve $\mathbf{m}_j$, the procedure $\texttt{search}(\mathbf{d}_j, \mathbf{m}_j; \mathcal{D})$ may return the nearest neighbor from the dictionary, or one randomly picked among either the K nearest neighbors, or among all points within the ball of radius $(1+\varepsilon)d_{\min}$ around the query vector $\mathbf{d}_j$, or within a ball of radius ε around the nearest neighbor. Due to the complexity of the nearest neighbor search in a large dictionary of high-dimensional vectors, dimension reduction via PCA and approximate heuristics (kd-tree in particular) have also been considered by most authors. To both speed-up the image generation and reduce the risk of unrealistic results, Ashikhmin (2001) also introduced a consistency principle: the first sample patches to be searched are those that continue *verbatim* what has already been copied in one of the neighboring locations (Fig. 3):

if $\arg\min\{\texttt{dist}[\mathbf{d}_j, \mathbf{d}^*_{\gamma^*_{-\mathbf{t}}\alpha\gamma_{\mathbf{t}}(j)}],\ \mathbf{t}\in T,\ \alpha\gamma_{\mathbf{t}}(j)\neq 0,\ \gamma^*_{-\mathbf{t}}\alpha\gamma_{\mathbf{t}}(j)\neq 0\} < \text{threshold}$

the shift $\mathbf{t}\in T$ for which the minimum is reached yields $\alpha(j) = \gamma^*_{-\mathbf{t}}\alpha\gamma_{\mathbf{t}}(j)$ and the search is stopped for location j.

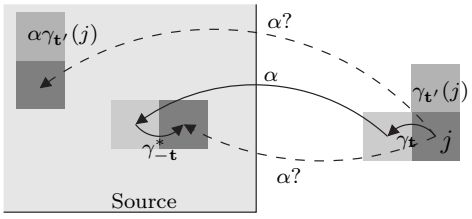


Fig 3. Consistency-driven search. Best patch $\alpha(j)$ for location j is first searched such as to continue *verbatim* one of the source patches that has already been used to populate a neighboring location $\gamma_{\mathbf{t}}(j)$.

In the region-filling technique of Bornard et al. (2002), this consistency strategy is coupled with a proximity principle, according to which the candidate sample patches for a given location are only searched within a certain maximum distance from this location.

Location-dependent search is also used in (Jia & Tang 2003) where a texture-based segmentation of the image is performed beforehand and interpolated within the selected region such that each location to be filled is associated to a certain texture (and associated image patches).

- Auxiliary guidance: multi-resolution texture synthesis, super-resolution, texture-by-number are specific applications of NP image generation where auxiliary guidance is employed (low resolution image in the two former, class labels in the latter). In (Drori et al. 2003), the region filling problem is addressed within a multiresolution setting where previous level is used to drive the detail generation at the current one. Also of interest for the region filling problem, a depth

map can be used as auxiliary guidance to mimic the changes of texture scale caused by depth changes in the scene (Harrison 2001, Hertzmann et al. 2001).

- Tiling geometry and order of visit: apart from the hole-filling technique in (Barret & Cheney 2002), where irregular micro-regions are used to manipulate the image, only regular tilings have been considered in the literature on example-base image generation. The issue of synthesis order is mostly relevant when initial boundary conditions have to be taken into account, as we shall discuss later for our region filling problem. For other unconstrained image generation, complete new images are generated usually in the lexicographical order.

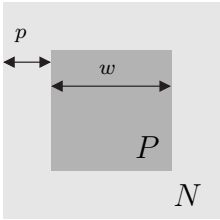
2.2 PatchWorks

The basic version of *PatchWorks* is summarized in Alg. 2. It is a particular instance of the generic framework described above with the following settings.

Algorithm 2 Basic *PatchWorks*

Input

p	width of square patches
q	width of neighborhood
ℓ	max. distance of patches from destination
J	original image of support Ω
$D \subset \Omega$	destination to be filled-in ($D \neq \emptyset$)
$\Gamma \subset \Omega - D$	source of image patches



- If $\Gamma = \emptyset$, $\Gamma \leftarrow \{\mathbf{x} \in \Omega - D : d(\mathbf{x}, D) \leq \ell\}$.
- For each $\mathbf{x}$ s.t. $\mathbf{x} + N \subset \Gamma$, associate one entry $\mathbf{d}_i^*$ in the dictionary.
- Initialize binary completion mask M s.t. $M(\mathbf{x}) = 0$ iff $\mathbf{x} \in D$.
- Tile Ω with step p: $\Omega \subset \cup_j P_j$ with P_j having upper left corner $\mathbf{x}_j$.
- Label recursively inner layers: $L^0 \doteq \emptyset$ and for $k = 0 \ldots K-1$, $L^{k+1} = \left\{ j \in \{1 \ldots n\} - \cup_{u=1}^{k} L^u : \ \mathbf{i}_j \cap L^k \neq \emptyset \right\}$.
- For $k = 1 \ldots K$ visit L^k lexicographically, and for current location index j:
  - Assemble $\mathbf{d}_j$ and $\mathbf{m}_j$.
  - Search: $\alpha(j) \leftarrow \arg\min_{i=1 \cdots n^*} \|\mathbf{n}_j - \mathbf{n}_i^*\|_{1,\mathbf{m}_j}$
  - Copy: $J(P_j) \leftarrow I(P^*_{\alpha(j)})$, $M(P_j) \leftarrow 1$

Images J is a color image.

Destination The destination region to be filled-in, $D \subset \Omega$, composed of one or several disjoint parts, is provided manually using any appropriate selection tool, e.g., drawing-based, brushing-based, color-based. The latter type of selection is crucial when it comes to select convoluted and/or numerous regions, as in the first examples in Section 4.1.

Source By default the sample source is defined as a band of width ℓ around selection D (Fig. 4). This default mode works well in most circumstances, and results in an effortless process from the user's perspective. In some situations however, it is useful to select manually the source region $\Gamma \subset \Omega - D$, to prevent for example the growing of a given structure or texture from the surrounding (Fig. 4). The user can thus control the nature of synthesized textures through the choice of texture "buckets". He/she can as well as control the trade-off between the computational complexity of the filling-in process and its quality through the size and richness of the sample source.

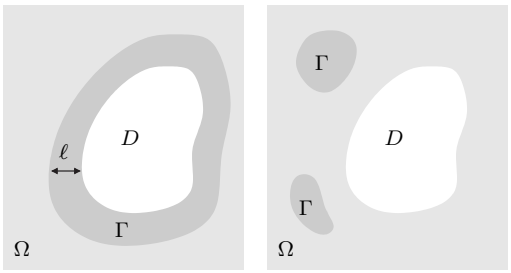


Fig 4. Source and destination in PatchWorks. The destination region $D \subset \Omega$ is provided in some way by the user. (Left) By default the source of image samples, Γ, is the band of width ℓ around it. (Right) It can also be manually provided as one or several connected regions.

Fig. 5 shows an example where the manual selection of the source region has prevented the undesirable bleeding of one of the textures neighboring the destination while speeding up the filling process.

Patches Whereas most non-parametric image generation technique proceed one pixel at a time (but generally with rather thick neighborhood), a few authors have demonstrated that using copying entire square patches instead improves dramatically both the speed and the quality of unconstrained texture synthesis (Guo et al. 2000, Efros & Freeman 2001, Liang et al. 2001, Kwatra et al. 2003).

This is the choice we make for our region-filling tool: the building bricks are square patches of size p, where p typically ranges from 4 to 20. Their neighborhood is formed by a w-pixel wide band around it, where w ranges typically from 2 to 4. The impact of these two parameters in terms of result quality and computational load is illustrated on a few single-texture examples in Figs. 6, 7, and 8. Such single-texture examples have been thoroughly investigated in the context of unconstrained

Fig 5. Interest of switching to manual source input. (Left) When removing an object at the boundary of another one, e.g., the person highlighted in yellow behind the hat of the baby in this example, the use of the default source often results in an undesirable bleeding of the latter object in the background. (Right) Alternatively, the user can specify the source of sample patches, a small chunk of sky highlighted in green here. This not only prevents the above mentioned bleeding, but also permits to control the size of the dictionary, hence the computational load: in this example the manually selected sample source is 10 times smaller than the default one for $\ell = 20$.

texture synthesis. In these studies, it has been shown that the the width w of the neighborhood in case of pixel-wise generation ($p = 1$) must get rather large to capture correctly certain textural patterns. In fact the neighborhood must be larger than the maximum textel size in the image (Efros & Leung 1999). However, the very high-dimensional search associated to large neighborhoods, combined with the intrinsic slowness of pixel-wise procedures, results in prohibitive computational loads. Using patches instead of single pixels, not only reduces the number of queries to the dictionary, but also often limits the need for large neighborhoods. Hence the computational gain is two-fold for at least similar quality, as shown in Fig. 6. Even in extreme cases, e.g., in Fig. 8, pixel-wise synthesis will fail to capture the correct patterns, no matter how large the neighborhood gets. By contrast, very large patches, with modest neighborhoods can still work well, at very low cost.

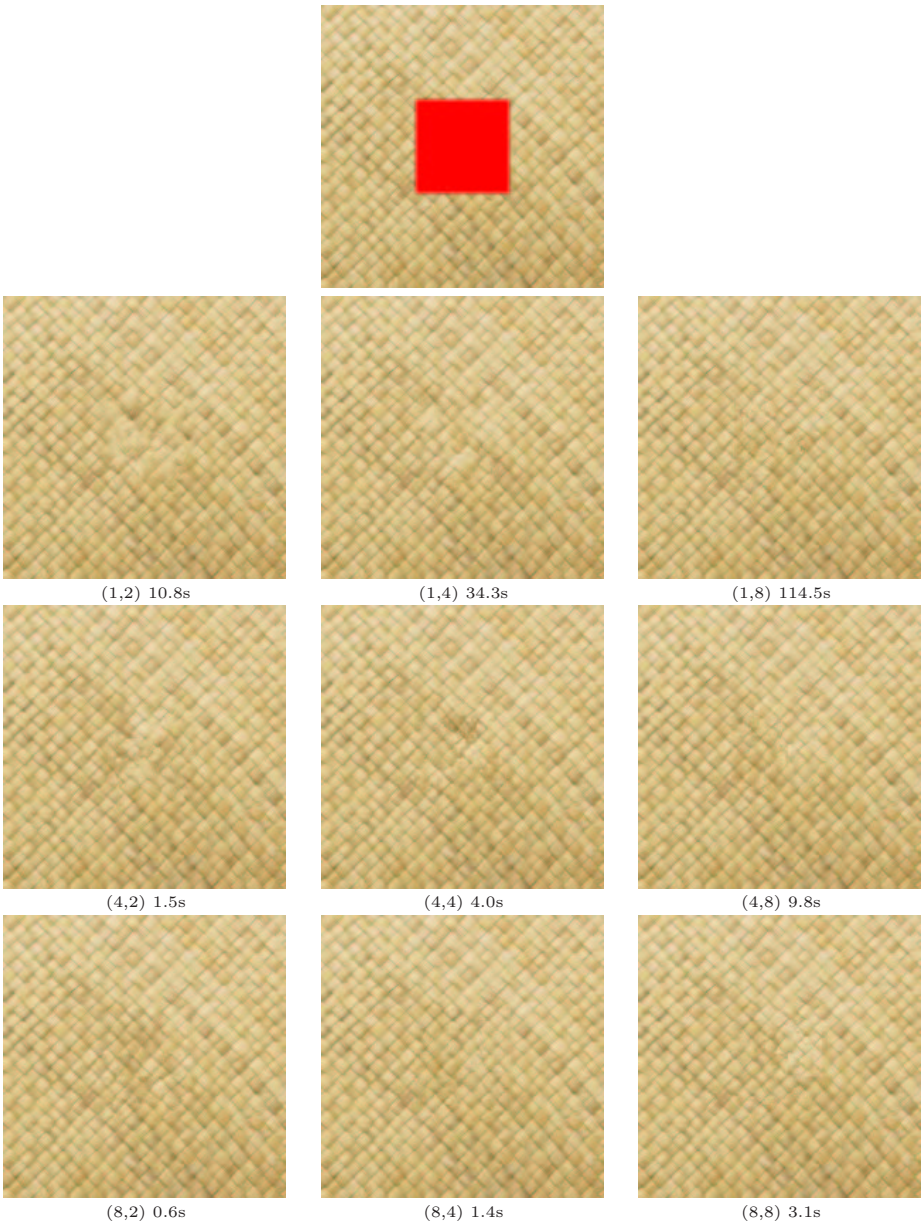


Fig 6. Impact of patch size p and neighborhood width w. The caption under each result indicates the parameter couple (p, w) followed by the cpu times on a Pentium IV, 2.3GHz, 1MByte of Ram. While in pixel-wise synthesis the size of the neighborhood is a critical parameter, it appears that the size of the patch can have an even more dramatic impact. This first example illustrates the fact that a few pixels wide patches with slim neighborhoods allow us to get similar or better results than single-pixel generation with large neighborhoods, while offering orders of magnitude of speed-ups. Pixel-wise filling is only satisfactory when w gets to 8, whereas large 8×8 patches do similarly well 200 times faster with only $w = 1$. On this example, Harrison's *Resynthesizer* yields results similar to (1,2) after 50s. [200×200 image, 66×66 destination region]

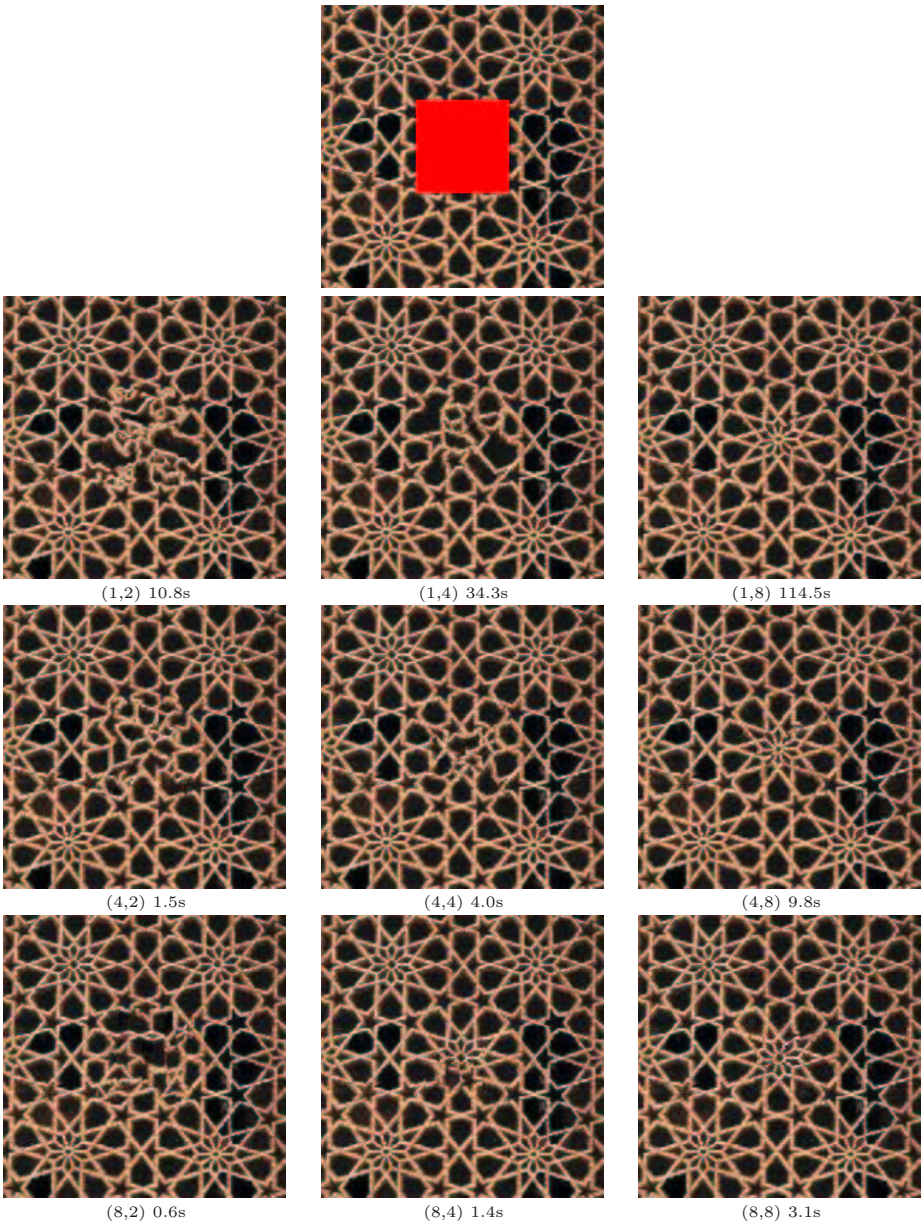


Fig 7. Impact of patch size p and neighborhood width w (continued). In this second example, with more complex structured patterns, good results were only obtained with a large neighborhood, irrespective to the size of the patch. Although now reduced, the speed-up offered by large patches for a similar quality remains nonetheless dramatic. On this example, Harrison's *Resynthesizer* yields results similar to $(p, w) = (1, 2)$ after 50s. [200×200 image, 66×66 destination region]

Synthesis order In (Efros & Leung 1999, Wey & Levoy 2000, Liang et al. 2001), constrained texture synthesis for region-filling is conducted in a concentric way, e.g.,

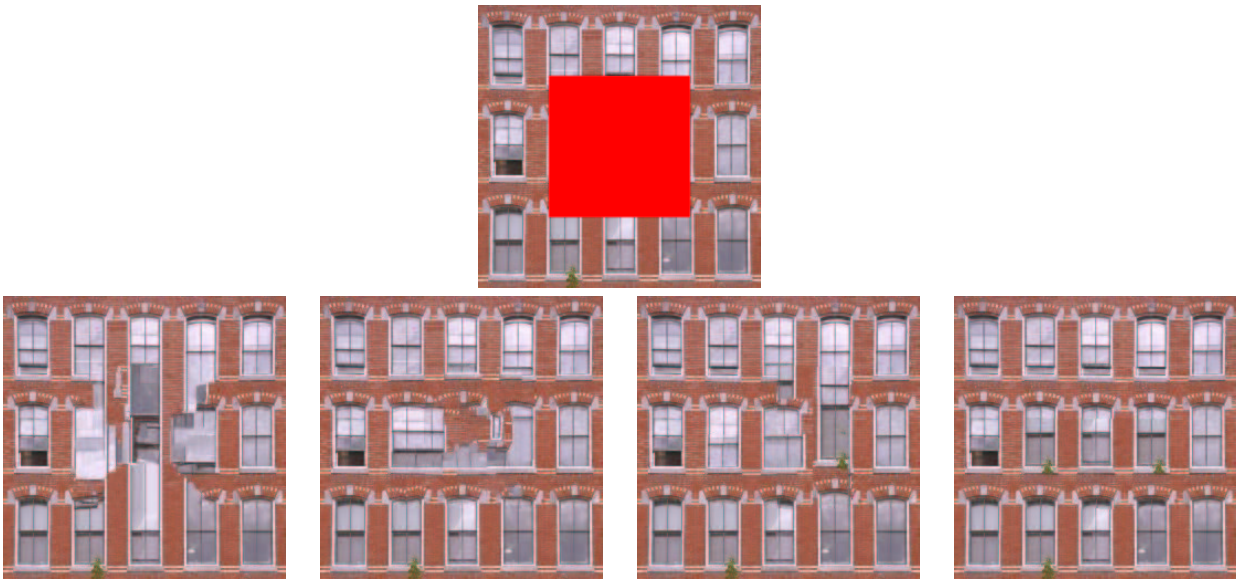


Fig 8. Impact of neighborhood width w in presence of highly structured patterns. On this extreme example of highly structured patterns, increasing the size of patches up to 50 improves dramatically both the quality and the speed. An excellent fill is finally obtained with no more than a two pixel-wide neighborhood. By contrast, increasing a lot the size of the neighborhood hardly improved the result of single-pixel (or small-patch) generation, while increasing prohibitively the computational complexity [e.g., 45mn for (1,16)]. [256×256 image, 128×128 destination region]

one layer after another starting from the inner boundary of the region inwards. This choice seems indeed the simplest and more natural way to treat all boundary conditions on an equal footing, as opposed to raster-scan filling that makes bottom right boundary conditions less likely to be well met. A careful examination of this statement showed us that in many single texture situations encountered in natural images, no noticeable improvements were obtained by using concentric as opposed to lexicographical filling-in. However, when the texture under consideration either is substantially non-stationary, or exhibits oblique patterns, a concentric filling proves better (Fig. 9, top row). More importantly, when the destination straddles two regions characterized by different colors and/or texture, the concentric ordering offers a clear improvement (Fig. 9, bottom row): with a lexicographical synthesis not only the top region grows un-naturally inside the bottom one, but also visible seams might appear at the bottom/right part of the destination region. More sophisticated synthesis orders have been introduced in (Harrison 2001, Bornard et al. 2002, Criminisi et al. 2003, Drori et al. 2003). The one proposed by Harrison (2001) relies on a preliminary statistical analysis of the anisotropies of the textures in the image. Apart from the overhead associated to this preliminary computations conducted over the whole image, we found often that either it was making no noticeable difference (as compared to concentric pixel-wise filling-in à la Efros &

Leung (1999)), or it was overshooting stronger structures in an undesirable fashion. The one considered by Bornard et al. (2002) is purely geometric (filling-in first the most surrounded locations) and is likely to make only little difference compared to concentric filling, especially for the square regions used in Fig. 9. Finally the ones proposed in (Criminisi et al. 2003, Drori et al. 2003) rely on two different ways of mixing geometry and image content and both allow to continue nicely interfaces between differently colored or textured regions. This is an appealing feature (obtained as well by Jia & Tang (2003) thanks to an interpolated texture segmentation), but comes at a substantial computational cost.

Search heuristics We use exhaustive search for exact nearest-neighbor. Whereas randomness is useful to inject variety in unconstrained texture synthesis, we found that it was more of a nuisance (along with a computational overhead) than anything else in the constrained case we are interested in. Initial boundary conditions are sufficient to avoid *verbatim* synthesis, and noise is simply making seams more visible. As for the consistency principle of Ashikhmin (2001), it is in a sense subsumed by the use of square patches. Patch-based synthesis amounts to systematically enforcing the consistency principle on a fixed geometry basis. We found no need for further enforcing this principle at the level of patches themselves. As for the proximity principle used by (Bornard et al. 2002) in the context of pixel-based region-filling with source taken around the destination, we found that it degraded noticeably the quality of the results in our patch-based implementation.

Distance We found that the L1 norm with no weighting, i.e, $q = 1$ and Δ_n =Id in (2), computed directly in RGB space, behaves similarly to other, more expensive, distances proposed in the literature (L2 norm weighted by a Gaussian kernel, possibly within alternative color spaces). In particular, switching from $q = 2$ to $q = 1$ provided a 40% speed gain for similar results (even slightly better in certain cases, although no systematic behavior differences were noticed).

Overall complexity In the absence of auxiliary guidance, the size of a vector in the dictionary is $12w(p+w)$. An estimate of the number n^* of vectors in the dictionary is given by[5] $4(|D|^{1/2} + \ell + p + 2w - 1)(\ell - p - 2w - 1)$. Finally the number of time the dictionary will be searched is approximately $p^{-2}|D|$. For a 100×100 square region to be tiled with 5×5 patches surrounded by 2-pixel wide neighborhoods and chosen not further than 50 pixels from the destination region, a 168-dimensional dictionary of approximately 26,000 vectors will be searched 400 times. This amounts to compute 1.7 million scalar differences. Fig. 10 provides similar cost estimations for $(p, w) \in \{1 \dots 16\}^2$. It makes clear the drastic computational advantage offered by the patch-based architecture.

[5] Assuming D is a square, the number of square patches of size $p + 2w$ in a band of width ℓ around D is $(|D|^{1/2} + 2\ell)^2 - [|D|^{1/2} + 2(p + 2w - 1)]^2$, hence the result.

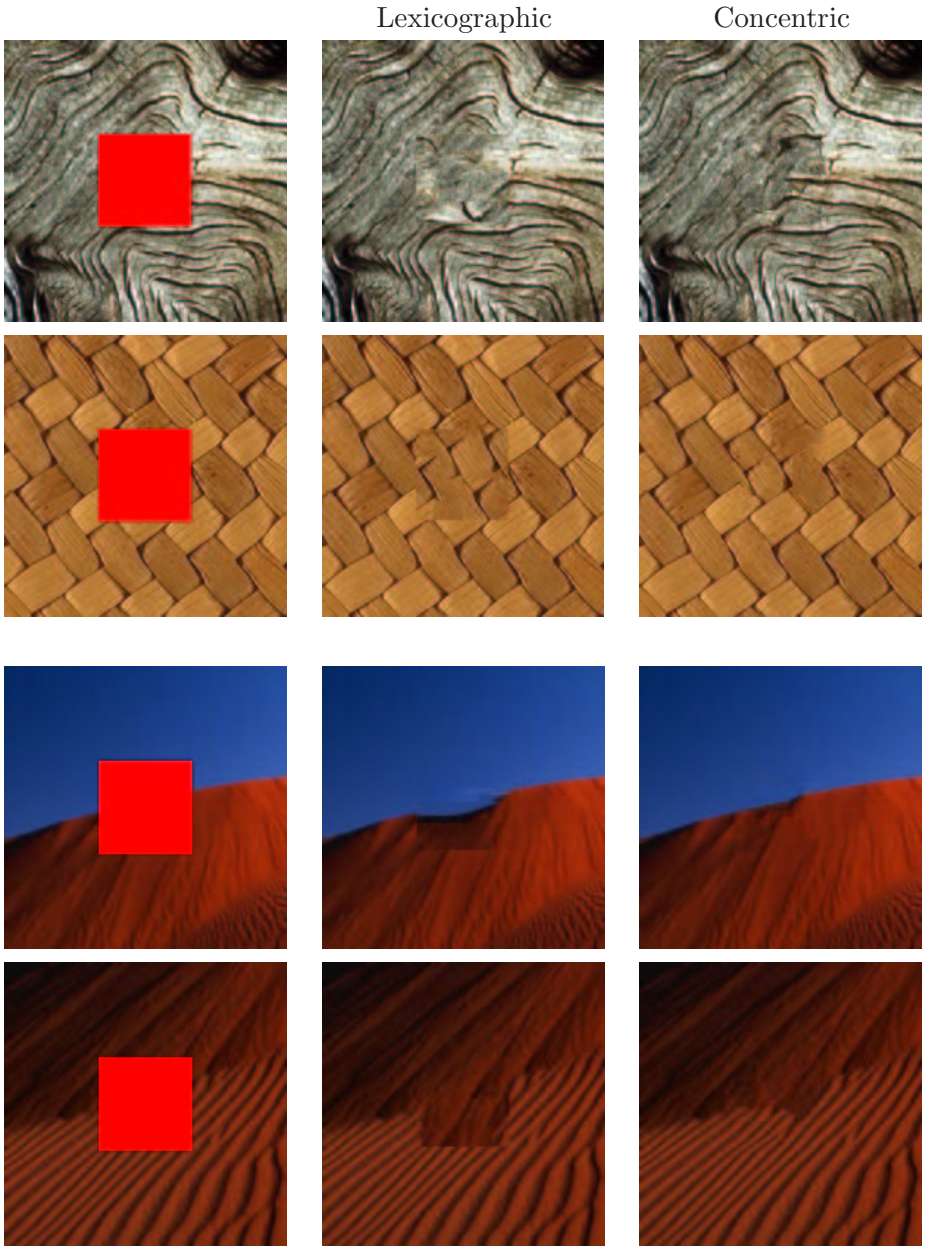


Fig 9. Two types of situations where concentric filling does substantially better than lexicographical filling. (Top rows) when filling a region surrounded by a "texture" which is either not stationary or with oblique patterns, a concentric ordering yields less visible seams than the lexicographical one. (Bottom rows) when filling a selection that straddles two very different regions, the top one is unnaturally favored by the lexicographical order and the boundary conditions at the bottom are visibly not well respected. [200 × 200 image, 66 × 66 destination region]

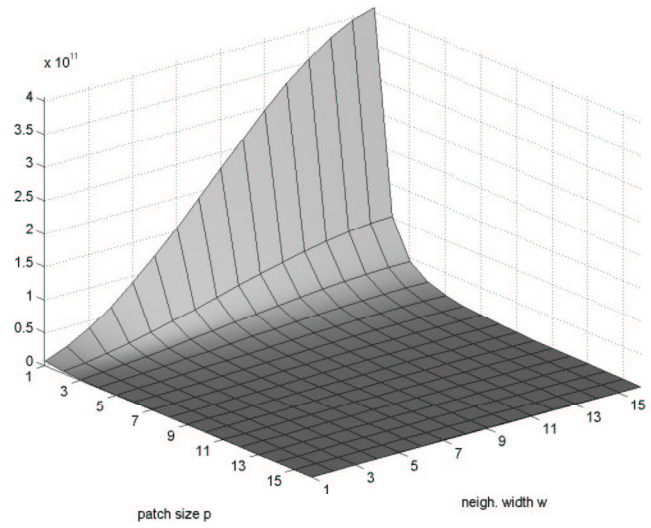


Fig 10. Complexity assessment as a function of (w, p)**.** This plot indicates the number of scalar distances, as a function of the patch size p and of the neighborhood width w, to be computed to fill a 100×100 pixel region, using the surrounding 50-pixel band as a source. It shows clearly the massive complexity reduction obtained as soon as patches are a few pixels wide. It also shows that the cost increase induced by the extension of the neighborhood is kept reasonably low within a patch-based system.

Auxiliary guidance All examples presented in this report make no use of auxiliary guidance: $\tilde{I}$ and $\tilde{J}$ do not exist and vectors $\mathbf{d}$ do not have an inner-patch component $\mathbf{p}$. The depth guidance proposed by Harrison (2001) can however be invoked within *PatchWorks* when the region to be filled corresponds to a textured surface with strong perspective distortion (typically the ground in scenes with large depth variations). Fig. 11 shows the impact of depth-based guidance in this situation. Note however that auxiliary guidance might result in a substantial overhead since the dimension of the search space is increased by $|P|$ times the number of channels of the auxiliary image (1 for depth). Also, in the absence of an automatic way of switching on this mode when appropriate, the use of this feature requires an additional input from the user.

3 Results

3.1 Single input image editing

We first present examples of one shot removal, i.e., one destination region is filled at once using the default source. In case the selection is composed of several disconnected components, these components are filled independently. In particular, each one is associated with its own surrounding source.

Fig 11. Advantage of depth-based guidance in presence of perspective distortion. When the object to be removed lies on a textured surface under strong perspective distortion, such as the gravel ground in this example, plain *PatchWorks* is usually unable to recreate the progressive change of scale in the synthesized image region (Middle). Using a depth-based complementary term in the similarity measure allows to circumvent this problem (Right).

Thin convoluted regions Using a color-based selection tool, thin convoluted structures of consistent color can be easily removed from an image. This can be useful to remove scribbling, scratches, logos, or any kind of obstructing text overlays. This is illustrated in Fig. 12. When dealing with such thin regions, the size of used patches should be roughly of the order of the gap width, e.g., $p = 2$ in the examples. Note that it is in this type of situation that *Inpainting* techniques have proved very effective. The results obtained with *PatchWorks*, as well as with the NP approaches of Harrison (2001) and Bornard et al. (2002), are similar to those of *Inpainting* for a presumably lower computation load.

Image extension Using *PatchWorks*, it is very easy to extend images from their external boundary outwards. This is useful to complete digitized old photographs with missing parts at the border. A first example of this task was presented in Fig. 1. A more dramatic illustration is offered in Fig. 13.

An other interesting application of this partly constrained image generation (only part of the destination region boundary is assigned boundary conditions) is the completion of the panoramas that are now routinely produced by users of digital cameras. Instead of cropping the original panorama into a smaller rectangular one, and hence loosing part of the original scene, the panorama can be easily extended to fill completely its rectangular bounding box (Fig. 14).

Fig 12. Removing thin convoluted structures. When restoring or correcting images, one might come across the task of removing thin structures of complex forms, such as text overlays. If these structures come in a consistent color, they are easily selected at once based on their color, and can then be removed using *PatchWorks* with default sources taken around each individual connected component of the selection (e.g., each letter). A close examination of the final image in this example shows that most of the structures, even those thin and complex, of the original image are convincingly completed.

Fig 13. Completing old photographs. It is quite common for old photographs to have torn borders. After digitization, *PatchWorks* can be used to re-invent the missing parts. On this 276×476 digitized old photograph, the 8,200 missing pixels in the lower left corner were filled in 0.2s. After 2min45s, *Resynthesizer* filled the same area in a much less satisfactory way, with artifacts appearing in the grass.

Fig 14. Completing panoramas. (Top) Typical panorama have irregular lay-outs due to the geometric correction of overlapping constitutive images. Getting back to a rectangular format is usually done by cropping, which results in a loss of original material, and in a restriction of the panorama scope. (Bottom) This is avoided by filling-in with *PatchWorks* the missing portions of the bounding box of the original panorama. On this 1058×311 panorama, 20,000 pixels associated to missing parts, as well as to the dark window frame in the upper right corner, were filled in 0.4s.

Filling large regions The patch-based architecture of *PatchWorks* makes easy the removal of large undesirable objects. The user only provides a rough outline of the destination region, and automatic filling is obtained fast enough for a comfortable use in an interactive image editing context. Filling large regions is useful in a number of image correction tasks where characters or objects have to be removed from the scene (Fig. 15), but also for other photo-editing tasks, such as photo-montages (Fig. 16).

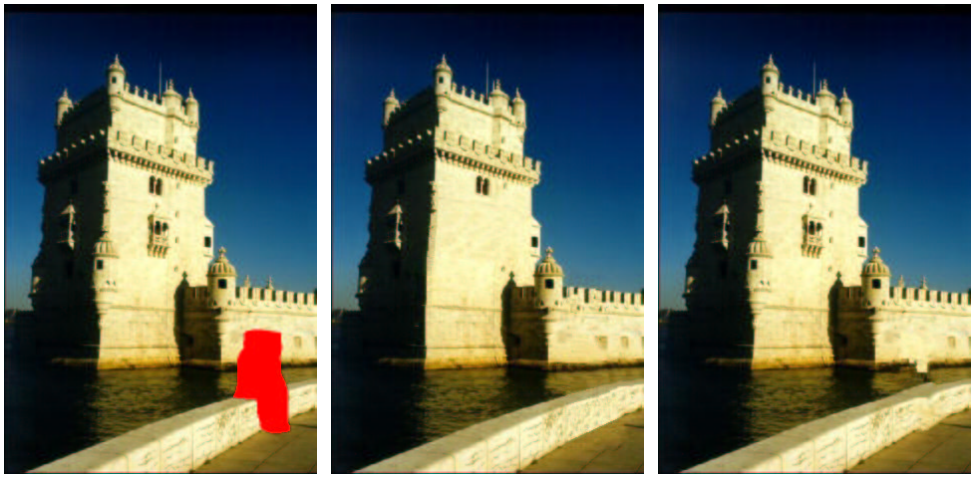


Fig 15. Removing large objects at once. The original 318 × 500 image on the left was corrected by removing a tourist. (Middle) *PatchWorks* fills the 4,000 pixels of the destination region in 0.2s by satisfactorily continuing the different wall and water regions surrounding it. (Right) By contrast, *Resynthesizer* performs less satisfactorily despite its 45s of computation.

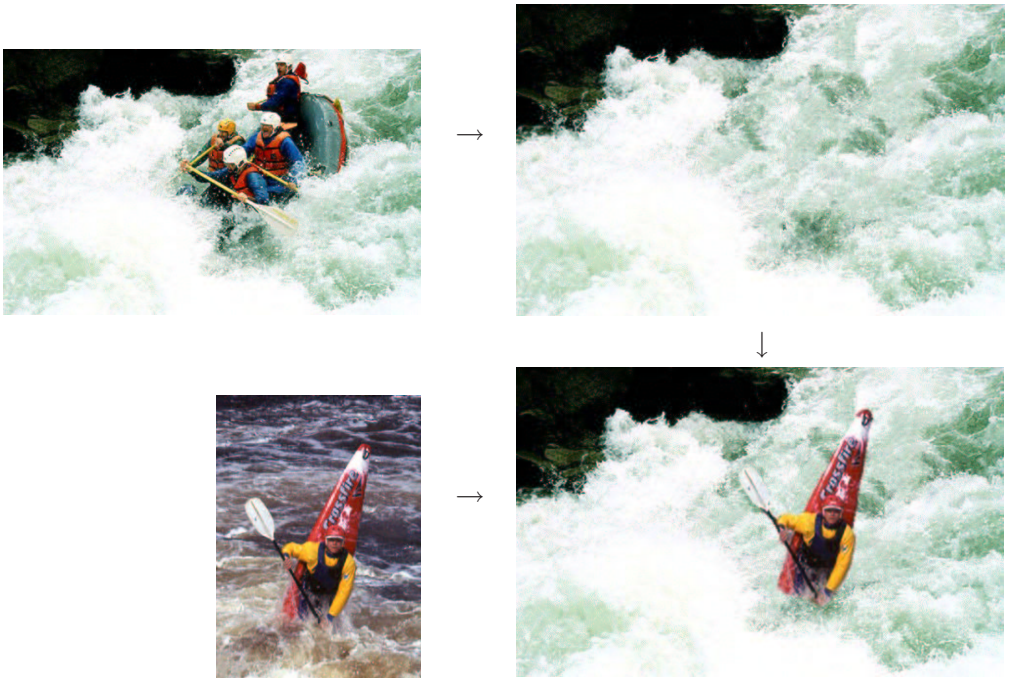


Fig 16. Clearing background of photo-montages. New images can be produced by importing cutout objects from different sources within a common background. In this context, *PatchWorks* proves useful to clear any undesirable objects from the original scene, the raft to be replaced by the kayak in this example. It took 1.5s with $(p, w, \ell) = (8, 2, 20)$ to fill a region amounting to 20% of the original 318 × 500 image. *Resynthesizer* fills less satisfactorily the same region, with dark chunks in the middle, after 8min.

3.2 Multiple input image editing

We conclude this section with a few examples of more elaborate image editing works. In these examples, *PatchWorks* was used in successive steps, often with the source manually specified, and with a few trials. All these results were nonetheless produced in just a few minutes.

Two of these more elaborate works were already shown in Fig. 1 to illustrate respectively the tasks of correcting the content of images and playing visual tricks. Figs. 17-19 present other examples of complex image corrections with one or several undesirable objects removed to make the scene more pleasing.

Fig 17. Small image corrections. *PatchWorks* may be employed to slightly change the esthetic or semantic content of a scene by removing small undesirable objects. In these examples, the removed objects although relatively small were occluding a great deal of texture and structure. Re-inventing these complex regions required a few trials, playing with the manual selection of the sources.

Fig 18. Not so small image corrections. This other example further illustrates how *PatchWorks* can be used to remove easily undesirable objects from a scene even when the part of background that is originally occluded is large and exhibits a fairly complex structure.

Fig 19. Major image corrections. This last example shows how a complete range of objects can be easily removed from the same single scene: this picture, taken from a moving car, has indeed been enhanced by removing the large blurred elements in the foreground, and the dim uninteresting structures in the far background.

4 Making seams invisible

By design, the patches that are used to tile the destination region are meant to produce as few seams as possible. As a result, no block artifacts are usually visible if the patches are not too large and the dictionary is rich enough. This is the case in all experiments presented in the previous section. When patches are large (e.g., larger than 5×5 pixels) and/or the dictionary is rather restricted, both situations occurring when trying to fill extremely large regions in a reasonable time, visible residual seams can show up: at each step of the filling process, it is unlikely that even the best source patch matches very precisely the existing neighborhood at the concerned location. These seams can be of three different types:

- Color/intensity seams,
- Texture seams,
- Structure seams.

The two first types are often made particularly visible by their lying along the regular grid of patches boundaries. Efros & Freeman (2001) suggest to break this geometric regularity by finding *a posteriori* optimal jagged boundaries in between the patches using a dynamic programming technique. This approach has been extended recently by Kwatra et al. (2003) to find the best irregular border of a patch at pasting time thanks to a max-cut/min-flow technique. These approaches based solely on the carving of patches frontiers are mostly effective on seams of type 2.When seams of type 1 occur in rather uniform regions simply because the dictionary is not rich enough, these techniques are not helpful.

This kind of seams are well handled in (Drori et al. 2003) using the Laplacian pyramid blending of Burt & Adelson (1983). We propose an alternative technique well adapted to our patch-based architecture. It relies on the use of the image blending technique proposed in (Pérez, Gangnet & Blake 2003), which corrects smoothly an image patch of any shape such that it complies exactly with new prescribed boundary conditions. It can be applied to any of the square patches used to tile the selection, with is neighborhood providing the new boundary conditions.

Let P_j a patch location in the destination region. It has been filled by source patch $I(P_i^*)$, with $i = \alpha(j)$. This copied patch can be modified by an additive correction f defined over $N_j \cup P_j$ by:

$$Lf = 0 \text{ over } N_j \cup P_j \text{ and } I(N_i^*) + f(N_j) = J(N_j), \tag{3}$$

where L is a discrete Laplacian operator. Hence the correction is as smooth as possible while ensuring that the modified source patch neighborhood complies exactly with the actual neighborhood in J. The computation of this correction amounts to solve a sparse linear system of order p^2. These corrections only need to be applied to every other patch of the tiling (one color of the checkerboard), at least for the inner layers. An example of this post-processing if shown in Fig. 20.

As for the seams of the third type, they are the most visible and it is not clear how they could be removed *a posteriori*. Using some kind of warping to deform concerned patches so that structures are better aligned might be envisaged but seems complex. Alternatively, part of these seams might be avoided at synthesis time by designing either a match distance penalizing them particularly or a synthesis order favoring their continuing (Harrison 2001, Criminisi et al. 2003, Drori et al. 2003). The former might however result in more seams of the two other types, whereas the latter comes at a non negligible computational cost.

PatchWorks Corrected *PatchWorks* Corrections

Fig 20. Removing seams by post-processing. The tiling obtained with a small dictionary of 20×20 blocks exhibits a few visible seams in this example. These seams are removed by applying an appropriate smooth additive correction to all the patches of the first inner layer, and to every other patch on the other layers. The corrections shown on the right are those applied in the green color channel. [size of destination region: 140×120 pixels; $w = 20$, $p = 2$, $\ell = 40$.]

5 Conclusions

We have shown that the *simplest* use of Efros and Leung's example-base image generation technique can be turned into a fast, robust, and flexible image completion tool using a block-based architecture. If results are in some specific cases not as good as those reported in recent works (Criminisi et al. 2003, Drori et al. 2003, Jia & Tang 2003), comparable results are obtained in most classic image completion tasks, but orders of magnitude faster. This latter aspect is crucial in the context of interactive image editing.

The main problem, partly attacked with different angles in (Bertalmio et al. 2003, Criminisi et al. 2003, Drori et al. 2003, Jia & Tang 2003), remains the issue of mixing convincingly structures, textures, and boundaries in the synthesis process. Higher level image analysis tools might be necessary to address this problem.

Another issue of interest concerns the reduction of the complexity when it comes to edit very large images, e.g, as large as 6 Mpixels. Because we are handling a different sample dictionary for each individual region filling, standard dimension reduction and clustering techniques are less appealing in our context and alternative solutions should be sought.

Acknowledgments

The first author thanks Raphaël Bornard and Antonio Criminisi for numerous discussions on example-based region-filling techniques as well as for their comments on early versions of this document.

References

Ashikhmin, M. (2001), Synthesizing natural textures, *in* 'Proc. ACM Symp. Interactive 3D Graphics', Research Triangle Park, NC, pp. 217–226.

Baker, S. & Kanade, T. (2000), Hallucinating faces, *in* 'proc. IEEE Int. Conf. Automatic Face and Gesture Recog.', pp. 83–88.

Ballester, C., Bertalmio, M., Caselles, V., Sapiro, G. & Verdera, J. (2001*a*), 'Filling-in by joint interpolation of vector fields and gray levels', *IEEE Trans. Image Processing* **10**(8), 1200–1211.

Ballester, C., Caselles, V., Verdera, J., Bertalmio, M. & Sapiro, G. (2001*b*), A variational model for filling-in gray level and color images, *in* 'Proc. Int. Conf. Computer Vision', Vancouver, Canada, pp. I: 10–16.

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

Barret, A. & Cheney, A. (2002), 'Object-based image editing', *ACM Transactions on Graphics* **21**(3), 777–784. (SIGGRAPH'02).

Bertalmio, M., Bertozzi, A. & Sapiro, G. (2001), Navier-stokes, fluid dynamics, and image and video inpainting, *in* 'Proc. Conf. Comp. Vision Pattern Rec.', Hawai, pp. I:355–362.

Bertalmio, M., Sapiro, G., Caselles, V. & Ballester, C. (2000), Image inpainting, *in* E. Fiume, ed., 'Proceedings of ACM SIGGRAPH 2000', Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, New-York, pp. 417–424.

Bertalmio, M., Vese, L., Sapiro, G. & Osher, S. (2003), Simultaneous structure and texture image inpainting, *in* 'Proc. Conf. Comp. Vision Pattern Rec.', Madison, WI, pp. II.707–714.

Bornard, R., Lecan, E., Laborelli, L. & Chenot, J.-H. (2002), Missing data correction in still images and image sequences, *in* 'Proc. ACM Int. Conf. on Multimedia', Juan-les-Pins, France.

Burt, P. & Adelson, E. (1983), 'A multiresolution spline with application to image mosaics', *ACM Trans. Graphics* **2**(4), 217–236.

Criminisi, A., Pérez, P. & Toyama, K. (2003), Object removal by exemplar-based inpainting, *in* 'Proc. Conf. Comp. Vision Pattern Rec.', Madison, WI, pp. II.721–728.

Drori, I., Cohen-Or, D. & Yeshurun, H. (2003), 'Fragment-based image completion', *ACM Transactions on Graphics* **22**(3), 303–312. (SIGGRAPH'03).

Efros, A. & Freeman, W. (2001), Image quilting for texture synthesis and transfer, *in* E. Fiume, ed., 'Proceedings of ACM SIGGRAPH 2001', Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, New-York, pp. 341–346.

Efros, A. & Leung, T. (1999), Texture synthesis by non-parametric sampling, *in* 'Proc. Int. Conf. Computer Vision', Kerkoa, Greece, pp. 1033–1038.

Freeman, W., Pasztor, E. & Carmichael, O. (1999), Learning low-level vision, *in* 'Proc. Int. Conf. Computer Vision', Kerkoa, Greece, pp. 1182–1189.

Freeman, W., Pasztor, E. & Carmichael, O. (2000), 'Learning low-level vision', *Int. J. Computer Vision* **40**(1), 25–47.

Guo, B., Shum, H. & Xu, Y.-Q. (2000), Chaos mosaic: Fast and memory efficient texture synthesis, Technical Report MSR-TR-2000-32, MSR.

Harrison, P. (2001), A non-hierarchical procedure for re-synthesis of complex texture, *in* 'Proc. Int. Conf. Central Europe Comp. Graphics, Visua. and Comp. Vision', Plzen, Czech Republic.

Hertzmann, A., Jacobs, C., Oliver, N., Curless, B. & Salesin, D. (2001), Image analogies, *in* E. Fiume, ed., 'Proceedings of SIGGRAPH 2001', Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, New-York, pp. 327–340.

Jia, J. & Tang, C.-K. (2003), Image repairing: robust image synthesis by adaptive nd voting, *in* 'Proc. Conf. Comp. Vision Pattern Rec.', Madison, WI, pp. I643–650.

Kwatra, V., Schödl, A., Essa, I., Turk, G. & Bobick, A. (2003), 'Graphcut textures: image and video synthesis using graph cut', *ACM Transactions on Graphics* **22**(3), 277–286. (SIGGRAPH'03).

Liang, L., Liu, C., Xu, Y.-Q., Guo, B. & Shum, H. (2001), 'Real-time texture synthesis by patch-based sampling', *ACM Transactions on Graphics* **20**(3), 127–150.

Pérez, P., Gangnet, M. & Blake, A. (2003), 'Poisson image editing', *ACM Transactions on Graphics* **22**(3), 313–318. (SIGGRAPH'03).

Wey, L.-W. & Levoy, M. (2000), Fast texture synthesis using tree-structured vector quantization, *in* E. Fiume, ed., 'Proceedings of ACM SIGGRAPH 2000', Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, New-York.