Non-local filtering for denoising and inpainting

Marc Szafraniec
Master MVA
ENS Paris-Saclay

Abstract

This project report presents an implementation of the PatchMatch algorithm introduced in [Barnes et al., 2009], used here for inpainting. All the techniques described here come from this paper, unless stated otherwise. Inpainting consists in filling in masked regions of an image, in photo editing or film restoration for example, with coherent content with its surroundings. This algorithm aims, as its name says it, to find other patches of the same image that match the missing regions. This is done here using a nearest-neighbor algorithm, and is made really efficient by the idea that we can find accurate matches by random sampling among the rest of the image and then propagate them to all of the masked regions. This method is an important improvement compared to previous algorithms, in efficiency and mostly in speed, by the use of techniques such as propagation of matches across the image, while retaining a significant part of the underlying principles of previously existing algorithms. This report also describes experiments that show the efficiency of this method on multiple problems, even if its running time proved to be longer than expected.

1 Introduction

Ever since photography was created, and even more so since the apparition of computers and image editing software, numerous techniques have been implemented in order to realistically modify an image, first for propaganda purposes, now accessible to everyone on household computers. Be it for replacing lost or corrupted parts of images, or even simply removing red eyes on photographs, inpainting is an important component of today's image editing.

Figure 1: A famous early use of inpainting, back in the USSR

1.1 Previous Work

Inpainting is a method for completion of large "holes" in images. A recurring idea in inpainting algorithms is to complete the unknown regions based on the known regions of the same image. First, it was defined as an edge continuation process, but this method is restricted to small missing image portions in highly structured image data. A more efficient technique is *texture synthesis*, which one of the pioneering works is [Efros and Leung, 1999]'s, which aims to fill missing pixels using values from its neighbors using Markov Random Fields. That is, they assume that the value of a pixel given the values of its neighborhood is independent of the rest of the image. Thus, this technique synthesize texture which is both stationary and *local*. [Wexler et al., 2007] obtain impressive result on video inpainting by extending this technique: they give it a global - hence *non-local* - optimization problem framework, solving a well-defined objective function (besides obviously adapting it to the time dimension). The missing image portions are still filled in by sampling patches from other image portions, but they also enforce global consistency between all patches in and around the hole. The objective function states that every local image patch should be similar to some local patch in the remaining parts of the image, while globally all these patches must be consistent with each other. It also helps to remove the local inconsistencies of greedy algorithms such as [Efros and Leung, 1999]'s, which tend to use large patches to ensure better coherence. This is an improvement, because a greedy approach requires the correct decision to be made at every step, causing the chances for errors to increase rapidly as the gap grows. This objective minimization task is related to Probabilistic Graphical Models, by the fact that it is programmed using an EM-like algorithm.

This paper is also fundamental for the PatchMatch algorithm as it provides a unified framework for various types of image completion and synthesis tasks: it also uses the idea of processing the image at multiple scales, that is beginning by reducing the resolution by an important factor, then applying inpainting on the rescaled image, increasing the scale by a factor two, and repeating the operation until we attain the original scale again, but this time with all missing parts filled in. This also allows the final image to have a better global consistency. Besides, this framework requires that the whole neighborhood around each pixel is considered, and it considers all windows containing each pixel simultaneously, thus effectively using an even larger neighborhood.

So, how to find the patches to match? The answer is Nearest neighbor search methods, which have been investigated by many researchers over the years to be as efficient as possible, because their relative slowness is a real problem for several Megapixel images. That is one of the reasons of the scale-down idea presented before.

1.2 The PatchMatch Approach

PatchMath is very widely inspired by [Wexler et al., 2007], as for example it retains the idea of global optimization function as well as the idea of pyramidal downscaling. Obviously, these techniques have to be adapted from a video framework to simple images, but the content mainly stays the same. However, PatchMath adds in important improvements. An inspiration to complete this method is the works of [Ashikhmin, 2001], with the use of a *local propagation technique* assuming the coherence of the image, which means that two neighboring missing points should be matched by two neighbor patches from the rest of the image. This assumption allows to propagate a good match to its neighbors, and one thing leading to another to a whole region of the image. This is very powerful, because once a pixel is correctly matched in a missing region, it is pretty much sure that all the region will also be, and that in only one propagation step, and a huge advantage compared to other methods where having information on a pixel did not help much for its neighbors.

In PatchMatch, the nearest-neighbors search is done by computing an *approximate* Nearest Neighbor Field (NNF), that is a function that for each pixel of the input image associates an *offset*, a difference in position between the original pixel that has to be replaced and the associated new pixel. Thus, the real stake of the algorithm is to find this NN Field. Talking about neighbors in this case can be confusing, because here nearest-neighbor means "the patch from the rest of the source image that most closely matches this one in the target image". This search is at first done using *random sampling*, which means that we first associate to each pixel of the target image a random pixel from the source image, and progressively refine this by searching closer and closer from the original pixel. Then, we use propagation as described before, and the coupling of these two ideas is what really makes the force of PatchMatch.

One of the main advantages of this method is that it is much faster than previous ones (20-100 x), so it can be used in real time and hence allows interactivity with the user, which wasn't possible before. This allows this algorithm to be used for image retargeting for example, which is the problem of displaying images without distortion on media of various sizes, as well as image reshuffling, which is the rearrangement of content within an image, based on interactive user input, and in both cases provides important improvement to previous methods. It also permits the program to deal with much bigger images in the same time frame.

This paper is organized as follows: Section 2 presents each part of the algorithm in detail: first the Nearest-Neighbor Field and the important improvements in its updating, then pyramidal downscaling, and finally the minimization of the global optimization function using an EM-like algorithm. Sections 3 is a theoretical proof justifying the use of this EM-like algorithm, while Section 4 demonstrates the application of this method to various problems. Finally, Section 5 concludes this report and gives further perspectives of work.

2 Presentation of the algorithm

The PatchMatch algorithm functions by associating each pixel of a *target* image (the output) to a pixel of the *source* image (the input). That means that the final target image will be composed of pixels from the source image, sticking to it in non-empty regions, and filling missing pixels in the other ones. Practically, this is done by inputting two images, the source, and an image which will be the mask used to identify which regions are to be reconstructed. We assume here that the mask is provided manually, but it could also be the outcome of some segmentation algorithm.

2.1 The Nearest-Neighbor Field

2.1.1 Description

As said above, the core of the algorithm is to compute an approximated NNF for the image. A NNF associates to a pixel a from the target image an offset $b - a$, where b is the corresponding pixel in the source image. More precisely in this implementation, the NNF associates the couple (a_x, a_y) to b_x, b_y and to the distance between these two pixels, which is defined below. In the following, we'll note $f(a_x, a_y)$ the pixel corresponding to a, and $D(a_x, a_y)$ the distance between a and $f(a_x, a_y)$. When we create an instance of NNF, we must also specify the patch size S, to compute distances between patches more than between single points, which wouldn't make much sense. Each pixel in the target and source images are thus the center of a square patch of width $2S$, constituted by its neighbors.

2.1.2 Initialization

When the NNF is totally empty, we associate each point from the target to a random point from the source, and set the distance between them as infinity. Then, we compute the distance between each pair of corresponding pixels. More precisely, this distance if computed between the two patches associated to these pixels: it is the sum of the SSD (sum of squared differences) between each color band of each pixel of the source and target patches (for simplicity reasons, the color space is RGB and not LAB as in the original paper). If one of the patches is at the border of the image with a part out of it, or if it contains missing values, the distance for each of these pixels is set at a prohibitive cost, the same as if one of the pixels was totally white and the other one totally black.

2.1.3 Update

Here stands the biggest idea of the PatchMatch algorithm, which allows it to be so fast and powerful compared to previous techniques. We update a NNF by an iterative process which alternates between two steps: *propagation* and *random search*.

Propagation is based on the assumption that the image is coherent, which means that two points next to each other should correspond to two points that are also close to each other. Algorithmically speaking, that means that for each pixel (x, y) in the target image, beginning by the left- and up-most one, we compare $D(x, y)$ to the distance between (x, y) and $f(x - 1, y) + (1, 0)$ and between (x, y) and $f(x, y - 1) + (0, 1)$, and if one of them is smaller we replace $f(x, y)$ by the corresponding value. That is the *left-down* propagation. The goal of this operation is that if (x, y) has a correct mapping and is in a coherent region R, then all of R below and to the right of (x, y) will be filled with the correct mapping. Once that is made, we perform *right-up* propagation, which is the same operation reversed,

beginning from the right- and down-most pixel of the target.

Random Search also aims to improve the quality of the mapping, this time by selecting a random point r around $f(a)$, a being the pixel we want to update, and comparing the distance between a and r and $D(a)$. If it is smaller, we take $f(a) = r$. We then reduce the size of the search window by a factor 2, and repeat the operation until the window size is below 1 pixel.

Note that this algorithm only gives an approximation of the NNF, but we can prove that its convergence towards the exact NNF is very fast. Indeed, if we originally have an image consisting of M pixels, the probability that a given pixel is associated to its best match is $\frac{1}{M}$, but the probability that at least one pixel is correctly assigned by random sampling is already great and equals to $1 - (1 - \frac{1}{M})^M \simeq 1 - \frac{1}{e}$ for large M. Because the random search is quite dense in small local regions (the size of the window decreasing at each iteration by a factor 2) we can also consider a "correct" assignment to be any assignment within a small neighborhood of size C pixels around the correct offset. Such offsets will be corrected in about one iteration of random search. The probability to have a correctly matched point in such a neighborhood are very high: $(1 - (1 - \frac{C}{M})^M)$ or for large M, $1 - e^{-C}$. Given that if a pixel is correctly matched, propagation will match the entire region in which it is in one iteration. Let's take the problem from another point of view: if we call m the chosen patch size, and p the probability that a given point falls into the neighborhood (of size C) of the correct offset, we have $p = 1 - \left(1 - \frac{C}{M}\right)^m$. That is the probability of convergence before any iteration. This is a Bernoulli trial, hence we know that the time of convergence t follows a geometric distribution, and the expected time of convergence is $\langle t \rangle = \frac{1}{p} - 1$, or:

$$\begin{array}{rcl} \langle t \rangle & = & \left(1 - \left(1 - \frac{C}{M}\right)^{\gamma M}\right)^{-1} - 1 \\ & \underset{M \to \infty}{\longrightarrow} & \left(1 - e^{-C\gamma}\right)^{-1} - 1 \end{array}$$

If we call $\gamma = \frac{m}{M}$ the relative patch size. By Taylor expansion for small γ, we obtain $\langle t \rangle = \frac{M}{Cm} - \frac{1}{2} = \frac{1}{C\gamma} - \frac{1}{2}$, that is the expected time of convergence is constant with γ, which means that it is constant if we choose a adapted patch size m varying linearly with the resolution M. Obviously, a smaller patch size will lead to a more accurate algorithm, but a compromise with running time has to be found.

2.2 The Pyramid

In the PatchMatch algorithm, we begin by downsampling the images by a factor 2 until they are smaller than a given radius, $r = 2$ in this implementation. We obviously must take care of resizing the mask in order to correspond to this new image, that is if 3 out of 4 of the pixels are masked, the corresponding pixel in the downscaled image will also be, in a kind of maxpooling fashion. This idea is given in [Wexler et al., 2007]: we begin by applying the inpainting to the smallest resolution version of our image, and then refine it gradually until we come back to its original size. This enforces global consistency, as the image is progressively refined at the same time for the missing regions and for the rest of the image, and it is also useful to avoid getting stuck in local minima in the optimization process. Besides, this technique doesn't involve important computational costs, as the number of pixels is divided by 4 at each level, thus only multiplying the running time by $\frac{4}{3}$.

After finding an appropriate NNF for a level of the pyramid, one must scale it up to be an initialization for the NNF of the next higher resolution image. That is simply

done by increasing the size of the NNF, and then if the corresponding point to a is b, making correspond $2b$ to $2a$ and $2a+1$ with an infinity distance in the new NNF. This is very intuitive, and is once again based on the coherence assumption. Then, distances are recomputed and updated for all points in the new NNF following the method explained above.

Figure 2: From the higher to the lower level in the pyramid

2.3 The Expectation-Maximization algorithm

Once we computed the distances between the target and the source at this scale, with the same correspondences as at the lower scale, we wish to find the target image that matches the source at best. As stated before, this equates to minimizing the following objective function:

$$d_{BDS} = \frac{1}{N_S} \sum_{s \subset S} \min_{t \subset T} d(s,t) + \frac{1}{N_T} \sum_{t \subset T} \min_{s \subset S} d(s,t) \tag{1}$$

Here N_S is the number of patches in the source, N_T in the target and $d(s,t)$ is the SSD distance (defined above) between the patches s from the source and t from the target, s being the patch of the source that is most similar to t. The measure consists of two terms: on the left, the completeness term ensures that the output image contains as much visual information from the input as possible and therefore is a good summary; and on the right the coherence term ensures that the output is coherent with the input and that new visual structures are penalized. This minimization of the distance between each pixel and its corresponding patch using an EM-like algorithm.

Each iteration of an EM algorithm consists of two processes: the E-step, and the M-step. In the expectation, or **E-step**, the missing data are estimated given the observed data and current estimate of the model parameters. Here, we update the NNF as described above, then "patch-voting" is performed to accumulate the pixel colors of each overlapping neighbor patch. The weight of each pixel is calculated by a similarity measure to his corresponding patch, based on the distance:

$$\text{sim}(a,b) = e^{-\frac{d(a,b)}{2\sigma^2}} \tag{2}$$

σ^2 should be chosen to be the 75-percentile of all distances in the current search in all locations, as advised in [Wexler et al., 2007], in order to take into account the majority of locations and reduce the overall error. However, it really slows down the process to compute this percentile when we call sim, so I used the value $\sigma^2 = 0.05 * d_{\max}$ that still gives good results, $d_{\max}$ being the maximal possible distance. Thus we can rewrite (1) as

$$\text{minimize } d_{BDS} \Leftrightarrow \text{maximize sim}_{BDS} = \left(\prod_{s \subset S} \max_{t \subset T} \text{sim}(s,t) \right)^{\frac{1}{N_S}} \left(\prod_{t \subset T} \max_{s \subset S} \text{sim}(s,t) \right)^{\frac{1}{N_T}} \tag{3}$$

Which strongly reminds equation (1) from [Wexler et al., 2007] (equation (4)), only with the separation in separate completeness and coherence terms here.

$$\text{maximize } \text{sim}_{BDS} = \prod_{s \subset S} \max_{t \subset T} \text{sim}(s,t) \tag{4}$$

In the **M-step**, the distance function is minimized under the assumption that the missing data are known. All the color "votes" are averaged to generate a new image, using the weights computed previously: the estimate of the missing data from the E-step are used in lieu of the actual missing data.

This is summed up by the equation, given in [Wexler et al., 2007]:

$$c = \frac{\sum_i w_p^i c^i}{\sum_i w_p^i} \tag{5}$$

with c the value of a color of the pixel, w_p^i the weight given to the pixel c_i, with i representing all pixels in all the sources patches matching the target patches containing the pixel we want to update. To speed up the execution, we only update pixels if the "contribution" from the source image is significant, that is if $\sum_i w_p^i > 1$ (empirically determined value). The convergence of the famous EM-algorithm has been studied in great detail by [McLachlan and Krishnan, 2008]. We only need a few EM-steps per level of the pyramid, the minimum between 5 and the level in this implementation.

As explained in [Wexler et al., 2007], in order to avoid a blurry output we can upsize the penultimate target and compute the last target from the next level source image instead of directly upsizing the final target.

2.4 The algorithm

To sum up the PatchMatch algorithm, here is a step-by-step overview of the algorithm:

1. Create pyramid of source images
2. For each level, beginning by the highest one, repeat:
   (a) Initialize the NNF, at random if this is the highest one, else adapt it from the previous level's
   (b) Apply NNF update - propagation and random search
   (c) Apply an iteration of the EM-like algorithm
   (d) Output a target image that will be the next-level's first step
3. Once back at the original scale, we have our final target

This algorithm is relatively simple in its structure, but nonetheless very efficient. This simple overview can be useful for future implementations of this algorithm, because the original paper by [Barnes et al., 2009] unfortunately lacks of a clear summary of the different steps to follow for implementation.

3 Theoretical proof of the EM-like algorithm

In this section, we use a Probabilistic Graphical Model framework to prove the relevance of our variant of the EM algorithm, following the demonstration done in [Wexler et al., 2007] but adapting it to the case of images. In this framework, the global visual coherence of (1) can be derived as a likelihood function via a graphical model and our iterative optimization process can be viewed as a variant of the EM algorithm to find the maximum-likelihood estimate. For a simpler demonstration, we will show that the algorithm optimizes (4) and not (1), however the first easily implies the second one.

Under this model, the unknown parameters, denoted by Θ, are the color values of the missing pixel locations p in the space-time hole: $\Theta = \{c_p | p \in \mathcal{H}\}$. The known pixels around the hole are the observed variables $Y = \{c_p | p \in S \backslash \mathcal{H}\}$. The windows $\{W_n\}_1^N$ are defined as small patches overlapping the hole where N is their total number. The set of patches $X = \{X_n\}_1^N$ are the hidden variables corresponding to W_n. The space of possible assignments for each X_n are the data set patches in $\mathcal{D}$. We assume that each X_n can have any assignment with some probability. For the completion examples here, the data set is composed of all the patches from the same sequence that are completely known, i.e., $X = \{X_n | X_n \subset S \backslash \mathcal{H}\}$, but the data set may also contain patches from another source. This setup can be described as a graphical model which is illustrated in the following figure.

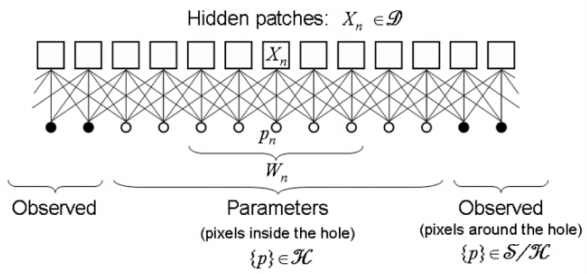


Figure 3: Our graphical model, representation by [Wexler et al., 2007]

Each patch W_n is associated with one hidden variable X_n. This, in turn, is connected to all the pixel locations it covers, $p \in W_n$. As each W_n denotes a patch overlapping the hole, at least some of these pixel locations are unknowns, i.e., they belong to Θ. Let $c_p = W_n^p$ denote the color of the pixel p in the appropriate location within the window W_n and let X_n^p denote the color at the same location within the data set windows. Note that, while the color values X_n^p corresponding to pixel p may vary for different window locations n, the actual pixel color c_p is the same for all overlapping windows W_n^p. Using these notations, an edge in the graph that connects an unknown pixel p with an overlapping window X_n has an edge potential of the form $\phi(c_p, X_n) = e^{-\frac{(c_p - X_n^p)^2}{2\sigma^2}}$. The graph in Fig. 3 with the definition of its edge potentials is equivalent to the following joint probability density of the observed boundary variables and the hidden patches given the parameters Θ:

$$f(Y, X, \Theta) = \beta \prod_{n=1}^{N} \prod_{p \in W_n} \phi(c_p, X_n) = \beta \prod_{n=1}^{N} \prod_{p \in W_n} e^{-\frac{(c_p - X_n^p)^2}{2\sigma^2}}$$

where p denotes here either a missing or observed pixel and β is a constant normalizing the product to 1. This product is exactly the similarity measure defined previously:

$$f(Y, X, \Theta) = \beta \prod_{n=1}^{N} e^{-d(X_n, W_n)} = \beta \prod_{n=1}^{N} \mathrm{sim}(X_n, W_n)$$

To obtain the likelihood function, we need to marginalize over the hidden variables. Thus, we need to integrate over all possible assignments for the hidden variables $X_1, ..., X_N$:

$$L = f(Y, \Theta) = \sum_{(X_1,...,X_n) \in \mathcal{D}^N} f(X, Y, \Theta) = \beta \prod_{n=1}^{N} \text{sim}(X_n, W_n)$$

The maximum-likelihood solution to the completion problem under the above model assumptions is the set of hole pixel values (Θ) that maximize this likelihood function. Note that, since the summation terms are products of all patch similarities, we will assume that this sum is dominated by the maximal product value (deviation of one of the patches from its best match will cause a significant decrease of the entire product term): $\max L \approx \max_{\{X\}} \beta \prod_{n=1}^{N} \text{sim}(X_n, W_n)$. Given the point values, seeking the best patch matches can be done independently for each W_n; hence, we can change the order of the max and product operators:

$$\max L \approx \beta \prod_{n=1}^{N} \max_{X_n} \text{sim}(X_n, W_n)$$

meaning that the maximum-likelihood solution is the same completion that attains best visual coherence according to (4). We will next show that our optimization algorithm fits the EM algorithm for maximizing the above likelihood function. In the E step, at iteration t, the posterior probability density $\hat{\pi}^t$ is computed. Due to conditional independence of the hidden patches, this posterior can be written as the following product:

$$\hat{\pi}^t = f(X|Y; \Theta^t) = \prod_{n=1}^{N} f(X_n|Y; \Theta^t) = \prod_{n=1}^{N} \beta_n \text{sim}(X_n, W_n)$$

Thus, we get a probability value $\hat{\pi}^t$ for each possible assignment of the data set patches. Given the above considerations, these probabilities vanish in all but the best match assignments in each pixel. This common assumption, also known as "Hard EM", justifies the choice of one nearest neighbor. In the M step, the current set of parameters are estimated by maximizing the following function:

$$\hat{\Theta}^{t+1} = \text{argmax}_{\Theta} \left(\sum_{(X_1,...,X_n) \in \mathcal{D}^N} \hat{\pi}^t \log f(X, Y, \Theta) \right)$$

Given the graphical model in Fig. 3, each unknown p depends only on its overlapping patches and the pixels are conditionally independent. Thus, the global maximization may be separated to local operations on each pixel:

$$\hat{c_p}^{t+1} = \text{argmax}_{c_p} \left(- \sum_{n|p \in W_n} (c_p - \hat{X}_n^p)^2 \right)$$

where the $\hat{X}_n$ are the best patch assignments for the current iteration. This is an L_2 distance between the point value and the corresponding color values in all covering patches and it is maximized by the mean of these values similar to (5). Similar to the classic EM, the likelihood in the "Hard EM" presented here is increased in each E and M steps. Thus, the algorithm converges to some local maxima. Pyramidal downscaling (see Section 2.2) and obviously of the propagation and random search techniques (Section 2.1) lead to a quick convergence and to a realistic solution with high likelihood.

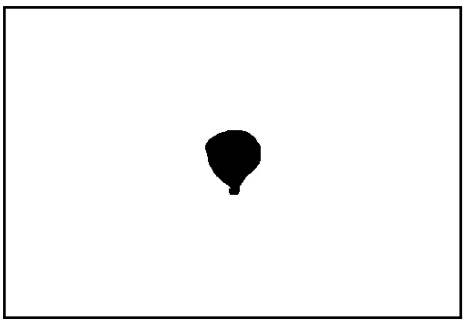
4 Experimental Results

4.1 Some experiments

As presented in the original paper, the PatchMatch algorithm works very well in most of the cases. I made some experiments on the following image:

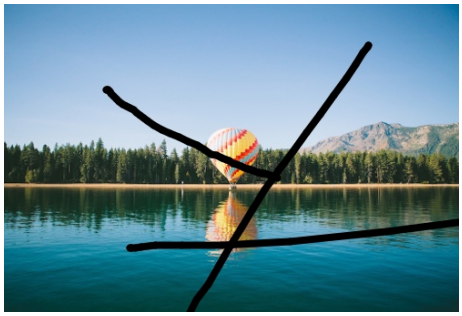
Figure 4: The test image

I will now apply object removal and image restoration to this simple air balloon. These are two different problems because the first one consist in replacing a "dense" zone of the image, while the second one has a more sparsely reparted mask.



(a) Image with mask in red

(b) Image with erased balloon

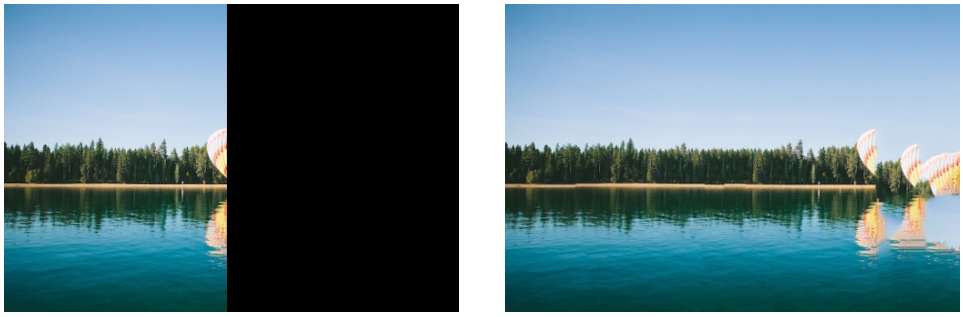


(c) Masked image (with background)

(d) Repaired image

Figure 5: Two different use cases of the algorithm

In the two first cases, PatchMatch does an excellent work. The upper-left of the balloon is a little blurry in the second case, and this can be attributed to the fact that it is a unique zone in the image, and no other region was able to replace it correctly, which is not the case in the first result. For the third case, the task is obviously way more difficult, and we can see that the program does not know how to reconstruct the right part from the balloon, so tries to stick it to the border of the image.



(a) Masked image (with background) (b) Reconstructed image

Figure 6: Reconstruction of half of the image

Another use that we can have of this algorithm is watermark removal. For this experiment I took an image on the famous site Shutterstock, which gives images with a watermark and asks a fee for removing it. PatchMatch is once again very efficient:

(a) Original image

(b) Masked image (mask in red here) (c) Result

Figure 7: Inpainting is dreadful for Shutterstock's business

However, it does not work as well on all images. When the region to inpaint is too large, the task becomes very challenging for the program, which has little to no idea of what could be a plausible completion. This is illustrated on the following images, where the algorithm fails to deliver a coherent result.

(a) Original image

(b) Mask

(c) Result of the algorithm

(d) The work of USSR artists

Figure 8: Removal of an embarrassing colleague is a difficult task for the algorithm

This failure can be explained by several facts. The image is in black and white, thus the colors can't deliver useful information that would probably have given a good result. Besides, the image is relatively small, so this is even more difficult for the algorithm to find useful patches. We can however note that the propagation is correctly done, as a great region on the right of the original Stalin is replicated.

This kind of examples show us the need to implement search space constraints, that is to restrain the region where we search for patches to match. In the paper, that is done in an interactive way and greatly improves the results, it even accelerates the process as the program does not have to search on the full image.

4.2 Running times

Shutter is the image from Shutterstock showed above, and Stalin is our image of Stalin. Balloon (1) corresponds to the reconstruction task while Balloon (2) corresponds to the erasing of the air balloon. Balloon (3) is the same image but with all the right half masked.

Running time (s)					
Image	m	Size	% of the size of Shutter	Running Time	% time of Shutter
Shutter	2	$1240 * 820$	100	5451	100
Balloon (1)	2	$500 * 340$	16	800	15
Balloon (2)	2	$500 * 340$	16	816	15
Balloon (2)	4	$500 * 340$	16	517	9.4
Balloon (3)	2	$500 * 340$	16	1087	20
Stalin	2	$300 * 200$	6	410	7.5

We take our biggest image, Shutter, as a reference for comparison. Let's also remind that m is the chosen patch size. Obviously, the bigger an image, the bigger the execution time. But we can see that the evolution of the running time is linear with the resolution of the image, as the proportion of the size of Shutter is the same as the proportion of its

running time. When we double m in Balloon (2), the running time is almost two times lower. There's a difference in Balloon (3) compared to the two other ones: the number of occluded pixels is roughly ten times higher, and the running time is multiplied by $\frac{4}{3}$. So we can see that the proportion of masked pixels has an influence, but it stays relatively slight. We can assume that the complexity of this algorithm is $\mathcal{O}\left(\frac{M}{m}\right)$, M being the resolution. The majority of the time is thus being used by the NNF update algorithm, and the time complexity confirms the theory from Section 2.1.3.

Besides, we can note that even if my computer isn't the most powerful on the market, that other programs were running when I did these experiments and that I did not implement this algorithm on GPU, we are still far from real-time or interactive capabilities...

5 Conclusion and Perspective

The PatchMatch algorithm is a very powerful tool for inpainting, that is filling missing or degraded regions of an images with coherent patches from other parts of the image. This is done by searching for nearest-neighbor patches all over the image, for each pixel, and then propagating the best matches to neighboring points. This algorithm inpaints multiple scales of the original image, beginning by the smaller size and propagating the results upwards, allowing a fast and consistent missing regions reconstruction.

This report presents the PatchMatch algorithm in detail, and reassembles its different parts that are not explicit in the paper of [Barnes et al., 2009], but dispersed in [Wexler et al., 2007] and others, which made it difficult to implement in the first place. The algorithm itself takes the best of previous methods, such as the use of a global objective function that permits a consistency over the whole image. It also retains the method of pyramidal downscaling. But above all, it brings in its own improvements such as the propagation method, that take advantage of the assumed coherence of images, in order to obtain a much faster and more efficient method than what was previously existing.

I didn't implement all the additional features of the algorithm that are in the original paper, even if not explained in great detail, which should certainly be an interesting thing to do. It can be applied to image retargeting for example, which is the problem of displaying images without distortion on media of various sizes (cell phones, projection screens) using document standards, and provides important improvement to previous methods. The paper also deals with image reshuffling, which is the rearrangement of content within an image, based on interactive user input. As the results are difficult to predict and can widely vary between tries, interactivity is necessary to quickly choose the best result among alternatives, and the speed of PatchMatch is a real advantage in these circumstances. Adapting PatchMatch to space-time video completion was a logical next step where speed of execution is certainly beneficial, and has successfully been achieved by [Newson et al., 2013]. Besides, I did not fully use the power of this algorithm, that is that all steps can be heavily parallelized on a GPU, making it seven times faster in the paper, probably even more so now since it was published eight years ago; however, in my implementation, the speed and possibility of user interaction boasted seems overrated, even if it certainly is a point where PatchMatch really improves its predecessors.

Finally, is would like to thank Xavier Philippeau, whose topic on the forum *developpez.net* under the user name *pseudocode* helped me to put together all the pieces of PatchMatch, and I hope that this report could be used for an easier understanding in future implementations of this very powerful algorithm.

References

[Ashikhmin, 2001] Ashikhmin, M. (2001). Synthesizing natural textures. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D '01, pages 217–226, New York, NY, USA. ACM.

[Barnes et al., 2009] Barnes, C., Shechtman, E., Finkelstein, A., and Goldman, D. B. (2009). PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3).

[Efros and Leung, 1999] Efros, A. A. and Leung, T. K. (1999). Texture synthesis by non-parametric sampling. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1033–, Washington, DC, USA. IEEE Computer Society.

[McLachlan and Krishnan, 2008] McLachlan, G. and Krishnan, T. (2008). *The EM algorithm and extensions*. Wiley series in probability and statistics. Wiley, Hoboken, NJ, 2. ed edition.

[Newson et al., 2013] Newson, A., Fradet, M., Pérez, P., Almansa, A., and Gousseau, Y. (2013). Towards fast, generic video inpainting. working paper or preprint.

[Wexler et al., 2007] Wexler, Y., Shechtman, E., and Irani, M. (2007). Space-time completion of video. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):463–476.