Image Inpainting using Wexler and PatchMatch algorithm

Anass Denguir, Luca Noel, Trong Quy Phan, Jieren Wu
Vrije Universiteit Brussel, Brussels,
{andengui, Luca.Noel, tronphan, Jieren.Wu}@vub.ac.be

***Abstract*—This paper describes the implementation of an interactive inpainting of an occluded image via the Wexler method with the help of the PatchMatch algorithm, also known as the nearest neighbor algorithm. The Wexler method is a patch-based algorithm using a multi-resolution pyramidal reconstruction. This reconstruction is accelerated using the PatchMatch algorithm. The latter is a patch research method based on a randomized correspondence table improved by a propagation and randomized search mechanism. This algorithm is preceded by the selection of the inpainted zone via an implemented graphical user interface. Results of the PatchMatch algorithm and the Wexler reconstruction method are shown in this document. The results validate the implementation of both algorithms. On the one hand, the PatchMatch allows the reconstruction of a source image no matter the destination image used. On the other hand, the Wexler method is validated by the smooth reconstruction of a masked image.**

Keywords: Wexler, PatchMatch, Nearest-Neighbor-Field, patch-based, reconstruction, completion

I. Introduction

Image inpainting consists of reconstructing an occluded region of an image. This operation is widely used in image processing. The challenge of such a task is to complete the missing pixels in a way that seems natural for the human eye. To do that, different kind of algorithms exist. The followed approach in this paper is based on Wexler reconstruction algorithm. This algorithm translates the image inpainting task into a mathematical optimization problem.

II. Description of the algorithm

In the following, the Patch based Wexler approach to image inpainting will be presented. The purpose of this algorithm is to reconstruct a coherent image Ω which has been occluded by a hole H from the knowledge of the rest of the image (i.e. the database) $D = \Omega \setminus H$.

A. Mask detection

At the first place, a mask need to be created to select a part of any shape of an image and remove it. Once the mask is created, it can be applied to any images of the same size by directly multiplying them both, then the expected part will be removed and the following steps can proceed.

The implementation is done in Matlab. First an image of a certain resolution will be read to indicate the specification of the mask. Then thanks to Matlab function, $ginput$, the mask shape can be selected using the mouse to select infinite number of points on the image. After that another function $roipoly$ is used to form the shape of the mask. Because those points may not be able to create a closed shape, the accuracy comes from using the mouse is not as high as expected, in most cases, the last point couldn't overlap with the first point which causes the gap of the whole shape. To make the complete shape, approximation is necessary here.

After the previous steps, an image which just contains the selected part will be produced. The idea here is to extract the mask out of it. So here a reversing method is considered. Since the way of using the mask is to multiply it with the image, so naturally, for the mask, the value of the shape which is utilized is set to 1, and the rest of it is set to 0. In this case, after multiplying, the original image will keep the value of the part which is not masked, and set all masked value to 0. Considering this, the extracted image keeps the original value in the certain shape, and the rest part of the image is 0. So reversing the value here means setting all the 0 value to 1 and the value of the extracted part to 0. Then a binary mask is now created which has the desired shape.

B. Wexler et al. Algorithm

Once the hole H identified, all the pixels belonging to this region have to be reconstructed. The RGB values of each pixel p of an image mainly depends on its neighborhood. Hence, in order to maximize the coherence of the image, the similarity of the windows containing the missing pixel has to be maximized with respect to the rest of the image. That is why the reconstruction algorithm first determines the set of windows $\{W_p^i\}_{i=1}^N$ of size N containing the pixel p. For each window W_p^i, a patch $V_q^i \in D$ is associated. The latter minimizes the quadratic cost function 1.

$$d(W_p, V_q) = \sum_{(x,y)} \|W_p(x,y) - V_q(x,y)\|^2 \tag{1}$$

The similitude defined by Wexler et al. in [1] between each pair of windows (W_p, V_q) is hence maximized according to equation 2. A good practice is to choose σ as the 75 percentile of the patch distances in the current search.

$$sim(W_p, V_q) = e^{-\frac{d(W_p, V_q)}{2\sigma^2}} \tag{2}$$

Afterwards, the color of the pixel p has to be filled with a color c. Let (x, y) being the relative position of the pixel p in a window W_p^i. The corresponding pixel q is situated at the same relative position in the matched window V_q^i. As expressed

in equation 3, each patch V_q^i contributes to the decision of the color c. It is computed using a weighted average on the RGB components c_i of pixels q belonging to each patch V_q^i. The decision is weighed by the degree of similitude of the patches V_q^i with respect to the corresponding window W_p^i. The similitude of each window is taken into account in the color decision by introducing the weight $\omega_P^i = sim(W_p^i, V_q^i)\alpha_P^i$ in equation 3. The factor α_P^i depicts the degree of confidence associated to each pixel participating to the reconstruction. Pixels belonging to the database D will have higher confidence than pixels that have been reconstructed in the hole H. That is why $\alpha_P^i = \gamma^{-dist}$ decreases exponentially by moving towards the center of the hole. The distance is evaluated by computing the distance transform of the occlusion H and γ can be set to 1.3 for hole that are approximately spherical.

$$c = \frac{\sum_i \omega_P^i c^i}{\sum_i \omega_P^i} \tag{3}$$

Wexler et al. refines the color decision in [1] by only computing the color of the pixels belonging to the main RGB clusters. These clusters contain only the pixels that are close to the main color modes. The purpose of this operation is to speed the convergence by removing oulier pixels from the decision set M. To do that, they propose to cluster the pixels using the Mean Shift algorithm. However, a less computationally expensive approach proposed by [2] is followed, which is detailed in the following.

***Multiscale Solution**:* The reconstruction algorithm can be optimized by downscaling the image in L pyramidal resolution levels. Each resolution level l is downscaled by a factor 2 with respect to the level $l-1$. The reconstruction starts at the coarser pyramid level L where the hole is filled following the "onion-peel" approach proposed by Newson et al. in [2]. It consists of filling the pixels of the hole H layer by layer. The levels k of each layer ∂H_k of the occlusion are determined by applying the cityblock distance transform on the mask computed in section II-A. In order to not bias the reconstruction with black pixels contained in the occlusion, all the pixels belonging to the outer layer ∂H_k are reconstructed using windows W_p^* containing only the most external hole layer as specified in equation 4.

$$W_p^* = \{W_p^i \mid H \cap W_p^i \subseteq \partial H_k\} \tag{4}$$

Once this initial reconstruction finished, the occlusion H_l corresponding to the pyramid level l is upscaled at the lower pyramid level $l-1$. The missing region H_{l-1} is then replaced by the upscaled version of the hole computed at higher level l, namely $upscale(H_l)$. For each level $l < L$, the reconstruction is repeated until the color of the filled pixel converges to a given RGB value. Convergence is determined when the variation of the color between two consecutive iterations is below a given threshold. If convergence does not happen, the algorithm waits for a maximal number of iterations to jump on a lower pyramidal level. The goal of such a method is to refine the reconstruction of the occlusion at each pyramidal level. Hence the missing pixels become more and more detailed. The multi-scale reconstruction process is illustrated in Figure 1.

At the finest pyramidal level, a final transformation is applied on the reconstructed hole H. This is necessary because the reconstructed pixels so far will seem blurred compared to the rest of the image. This is an immediate consequence of equation 3 which averages the values of many pixels candidate. To avoid avoid blurred results, Wexler et al. proposed in [1] to cluster the pixels candidate and to reconstruct the image using only the dominant cluster M. Another approach followed by Newson et al. in [2] consists of choosing the best pixel that minimizes equation 5 and apply this final reconstruction method only at the finest pyramidal level. This method has the advantage of being computationally less expensive than the first one while providing good results.

$$c = \underset{q \in S}{\arg\min}\ d(W_p, V_q) \tag{5}$$

C. *PatchMatch*

An algorithm aiming at accelerating the computation speed has to be implemented due to the expensive computation of brute force searches. The PatchMatch algorithm is used to create a table of correspondence between two images. This table is found by applying a randomized algorithm to find the nearest neighbor of each pixel of an image A when compared to an image B. The core element of this patch sampling method is the repeated search of all patches in an image B similar to a patch in another image A.

Given images A and B, the algorithm must find for every patch of A the nearest neighbor in B that minimize the euclidean distance L_p. The resulting mapping is called the *Nearest-Neighbor-Field* (NNF).

This algorithm allows a significant acceleration in computation. However, this search remains the main source of heavy computation. This algorithm allows the computation to speed up. The run-time is $O(mM\log M)$ where m is the number of pixel and M is the patch size.

***Approximate nearest-neighbor algorithm**:* The implemented system computes the patch correspondences. The *nearest-neighbor field* (NNF) is defined as the function $f{:}A \mapsto R^2$ of offset, defined for all coordinates of image A. A distance function D between two patches is also defined. The values of f are refereed as offset. The software is created while taking this detail into account. The algorithm has three components. First, the NNF must be initialized with random offsets. During the implementation of the NNF function, a matrix with the same dimensions as image A is initialized with random coordinates. After initialization the NNF undergoes iterative updates process. The good patch offsets are propagated to adjacent pixels. The last component of the algorithm is the random search in which a shrinking window is scanned to find the best corresponding patch so far.

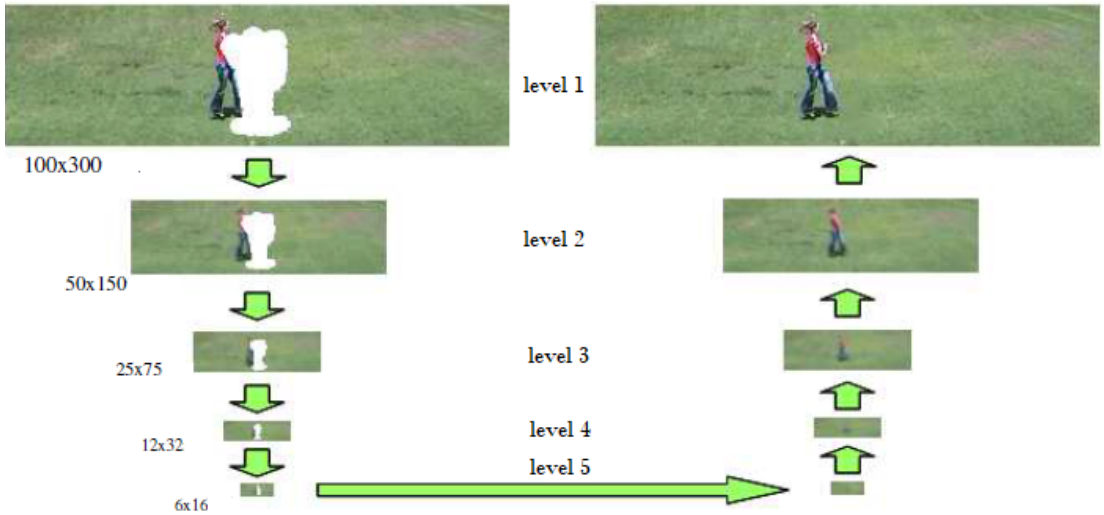

Fig. 1. Pyramidal resolution reconstruction

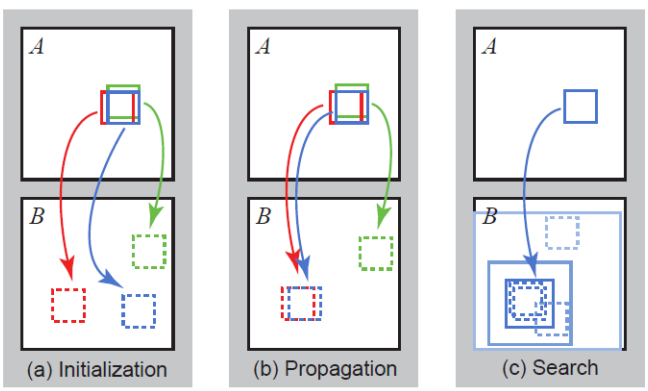

Fig. 2. PatchMatch scheme

Initialization: The NNF is initialized with random coordinates in this program. Independent uniformly distributed sampled are used to initiate the NNF. In the Wexler method, a resizing process is performed. The NNF is recalculated each time the algorithm goes to a finer level. If the NNF was upscaled and not recalculated, a reinitialization would have been preferable to avoid local minima, but in this case, a single initialization is done.

The initialization process is directly followed by the iterative research of similar patches. If P_i and S_i are respectively the propagation and the random search steps for the i th pixel then the algorithm proceeds as follows : $P_1, S_1, P_2, S_2, P_3, S_3, ..., P_n, S_n$where n is the number of patches.

Propagation: The value *f(x,y)* is improved by using *f(x-1,y)* and *f(x,y-1)*. We assume that the offsets for the three arguments considered are the same. Let's assume that a good mapping is done for *f(x,y-1)*, then the corresponding offset is used for the patch at (x,y). If *D*(v) represents the euclidean distance between patch (x,y) and (x,y)+*f(x,y)* (in L*a*b space), then the new offset assigned to (x,y) will be the arg min of *D((f(x,y))*, *D(f(x,y-1))*, *D((f(x-1,y))*.

If (x,y) has a correct mapping, then the patches below and on the right of the latter patch will correctly mapped as well. It is important to notice that for even iterations, the propagation process is done in reverse direction In other words, *f(x+1,y)* and *f(x,y+1)* are used for comparison with the patch (x,y)+*f(x,y)* as mentioned in [3].

Random search: After the propagation, v_0 = *f(x,y)* is used for the random search process. Several offset candidate are tested. The added offsets are limited by an exponentially decreasing window.

$$u_i = v_0 + w * \alpha^i * R_i \tag{6}$$

Where R_i is an 2D vector whose elements are uniformly distributed between -1 and 1. α is a fixed ratio between each search window sizes. The search window is reduce each time i increased until the search window is smaller than a pixel. In this case, $\alpha = 0.5$ and w is the maximum image dimension.

III. Experimental results

In this section, the results of the PatchMatch algorithm previously implemented in II-C will be illustrated. Afterwards, the results obtained for the Wexler reconstruction algorithm presented in section II-B will be shown using different patch research algorithms. The influence of parameters on the results will also be highlighted.

A. PatchMatch results

In Figure 3, the reconstruction of an image A (the Eiffel tower) is performed using only patches available in image B (representing a beach). The result is shown in the Reconstruction image. For example, one can see that the grass in image

Fig. 3. PatchMatch: reconstruct imageA with patch from imageB

(a) Original picture

(b) Brute-force (32 seconds)

(c) PatchMatch (7 seconds)

Fig. 4. Brute-force vs PatchMatch (79x70 pixels)

A is reconstructed with the leafs of the palm tree in image B. The good results presented here validate the PatchMatch implementation.

B. Wexler reconstruction results

The reconstruction algorithm results are illustrated here for two different patch research algorithms:

1) Brute force research
2) PatchMatch algorithm (implemented in section II-C)

Figure 4 compares the time performance of both research algorithms. The input image has a resolution of 79×70 pixels. One can see that the reconstruction speed is highly

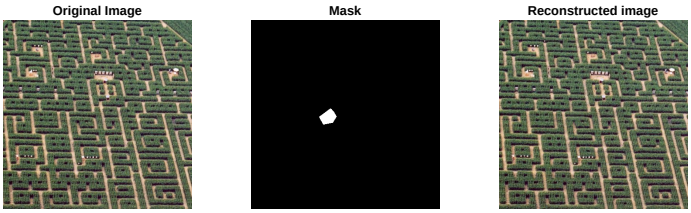

Fig. 5. Completion of a structured image

improved when using PatchMatch algorithm compared to simple Brute Force patch research. This is due to the fact that PatchMatch does not perform an exhaustive patch research. Moreover the outputs of both algorithms are very similar. Figure 5 illustrates the reconstruction of a structured image (a maze) using Wexler reconstruction method. One can see that the algorithm is very efficient as the original image is well reconstructed. This experiment validates the implementation of Wexler's algorithm. In Figure 7, the Wexler's algorithm is applied using windows of different size. A narrow window leads to bad reconstruction because the patch does not take into account enough neighboring pixels. Good results are obtained with a window size of 8 pixels.

(a) Original picture (b) Our result

Fig. 6. Image completion using PatchMatch (590x591 pixels)

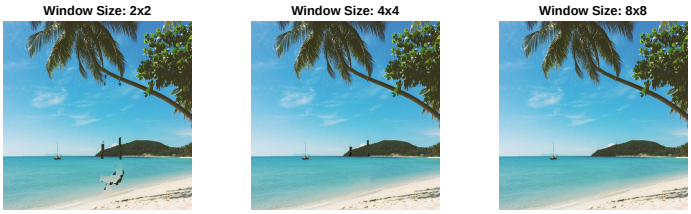

Fig. 7. Different window size

IV. Conclusions

An image inpainting algorithm was implemented using a patch based multi-scaled reconstruction method. The latter follows the approach proposed by Wexler et al. in [1] which aims at maximizing a global coherence function. To do so, the similitude between the patches containing the missing pixels and the patches contained in the rest of the image was maximized. A brute force research of those patches that maximizes the coherence is computationally heavy. In order to speed up the patch research, the PatchMatch algorithm was implemented. The results show that this method provides good reconstruction in a reasonable amount of time.

References

[1] Y. Wexler, E. Shechtman, and M. Irani, "Space-Time Completion of Video," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. 29, no. 3, p. 14, 2007.

[2] A. Newson, A. Almansa, Y. Gousseau, and P. Prez, "Non-Local Patch-Based Image Inpainting," *Image Processing On Line*, vol. 7, pp. 373–385, Dec. 2017. [Online]. Available: http://www.ipol.im/pub/art/2017/189/?utm_source=doi

[3] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "PatchMatch: A randomized correspondence algorithm for structural image editing," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 28, no. 3, Aug. 2009.

[4] Projet: Implmentation de patchmatch. [Online]. Available: http://vision.gel.ulaval.ca/~jflalonde/cours/4105/h17/tps/results/projet/111078106/index.html