Université de Bordeaux

Traitement d'image Avancée

Assignment 2

PatchMatch

Author:
Jimmy Gouraud

Supervisor:
Aurélie Bugeau

October 20, 2017

Contents

1 Introduction

La méthode "PatchMatch" [1] proposé par Connelly Barnes, Eli Shechtman, Adam Finkelstein et Dan B Goldman, permet d'améliorer la vitesse de calcul des algorithmes ayant pour objectif de trouver les correspondances entre les patchs. En effet, au lieu de parcourir l'ensemble de l'image à chaque fois que l'on veut trouver le "meilleur" patch, la méthode PatchMatch va au maximum recherche 3 patchs proche du pixel en question. Et il suffit de réitérer l'opération afin d'affiner le résultat.

2 PatchMatch

2.1 Algorithme de PatchMatch

Lors de l'initialisation, on crée une image de destination de la taille de l'image source et on la remplir avec des pixels choisis aléatoirement dans l'image cible. Lors de cette phase, on va sauvegarder la positions des pixels que l'on a pris dans l'image cible afin de créer notre Nearest Neighboor Field (NNF).

On obtient ainsi à la fin de l'initialisation une image de destination remplie de bruit issue des pixels de B ainsi qu'un tableau 2D (NNF) de la taille de l'image source contenant les coordonnées des pixels trouvés.

Algorithme 1 : Pseudo Code - PatchMatch

input :

- nb_iter: nombre d'itérations
- patch_hs: demi-taille du patch
- imgA: image source (l'image que l'on veut recréer)
- imgB: image cible (l'image d'où l'on va prendre les pixels)

output :

- imgC: image résultat ressemblant à l'image source mais avec les pixels de l'image cible

```
1  Initialisation de l'image de destination (imgC) avec des pixels de
     l'image cible et sauvegarde des positions des pixels (NNF)
2  for iter ← 1 : nb_iter do
3      for pixelC in imgC do
4          if iter % 2 == 1 then
5              [x, y] = pixelC; // On parcourt l'image de haut en bas
6          end
7          else
8              [x, y] = size(imgC) - pixelC; // On parcourt l'image de bas
                 en haut
9          end
10         patchA = patch(imgA, x, y);
11         [x2, y2] = NNF(x, y);
12         patchB = patch(imgB, x2, y2);
13         [x2, y2] = NNF(x±1, y);
14         patchB1 = patch(imgB, x2−±1, y2);
15         [x2, y2] = NNF(x, y±1);
16         patchB2 = patch(imgB, x2, y2−±1);
17         best_patch = find_best_patch(patchA, patchB, patchB1,
             patchB2);
18         pixelC = best_patch.pixel;
19         NNF(x,y) = best_patch.pixel;
20     end
21 end
```

Après l'initialisation on parcourt entièrement notre image C, un certain nombre d'itérations : de haut en bas pour les itérations impaires et de bas en haut pour les itérations paire (Algorithme 1 - ligne 4-9).

Pour chaque pixel de l'image C, on va retrouver le pixel correspondant dans l'image B grâce à notre NNF. On va ensuite placer un patch autour du pixel de B, un patch autour du pixel correspondant dans l'image A et calculer la SSD entre les deux. On va faire la même chose pour les pixels situé à gauche et en haut du pixel en cours de traitement, sans oublié de se décaler dans l'image B vers la droite ou en bas pour centrer le patch sur le pixel en cours de traitement (voir Figure 1).

On récupère ensuite le pixel du meilleur patch (celui avec le SSD la plus faible) et on le colle dans l'image C en mettant à jour au passage notre NNF (Algorithme 1 - ligne 19) afin d'améliorer les futurs comparaisons de patch.

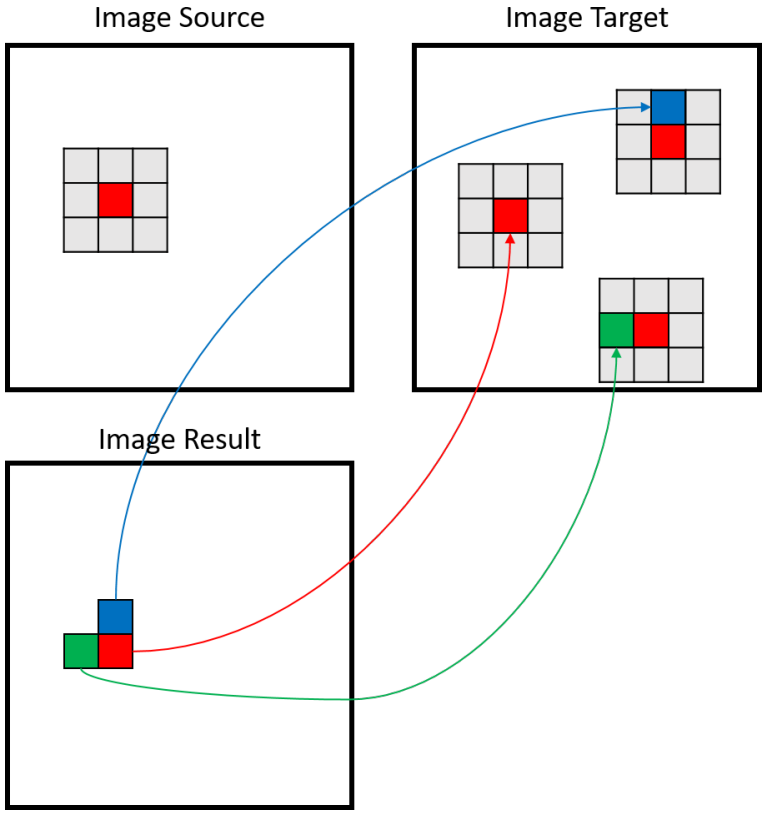

Figure 1: PatchMatch - Find best pixel

2.2 Résultats

La Figure 3 montre les résultats obtenus avec la méthode PatchMatch selon l'itération et la demi taille du patch. On remarque sans grande surprise que le nombre d'itération influence directement la qualité de l'image résultante. Cependant, après un certain nombre d'itération, l'image résultant ne changement pratiquement plus (à partir de 6 itérations, aucun changement notable n'est perceptible).

(a) demi-taille de 1 (b) demi-taille de 4

Figure 2: Itération 4

Plus la taille du patch augmente, moins bons sont les résultats. On entends par "bons résultats" une image ressemblant à l'image source sans bruit. Ainsi, avec une demi-taille de patch de 1 (c'est-à-dire une patch de taille 3x3 comme dans la figure 1), les résultats sont plus nettes qu'avec une demi-taille de patch à 4 (voir Figure 2).

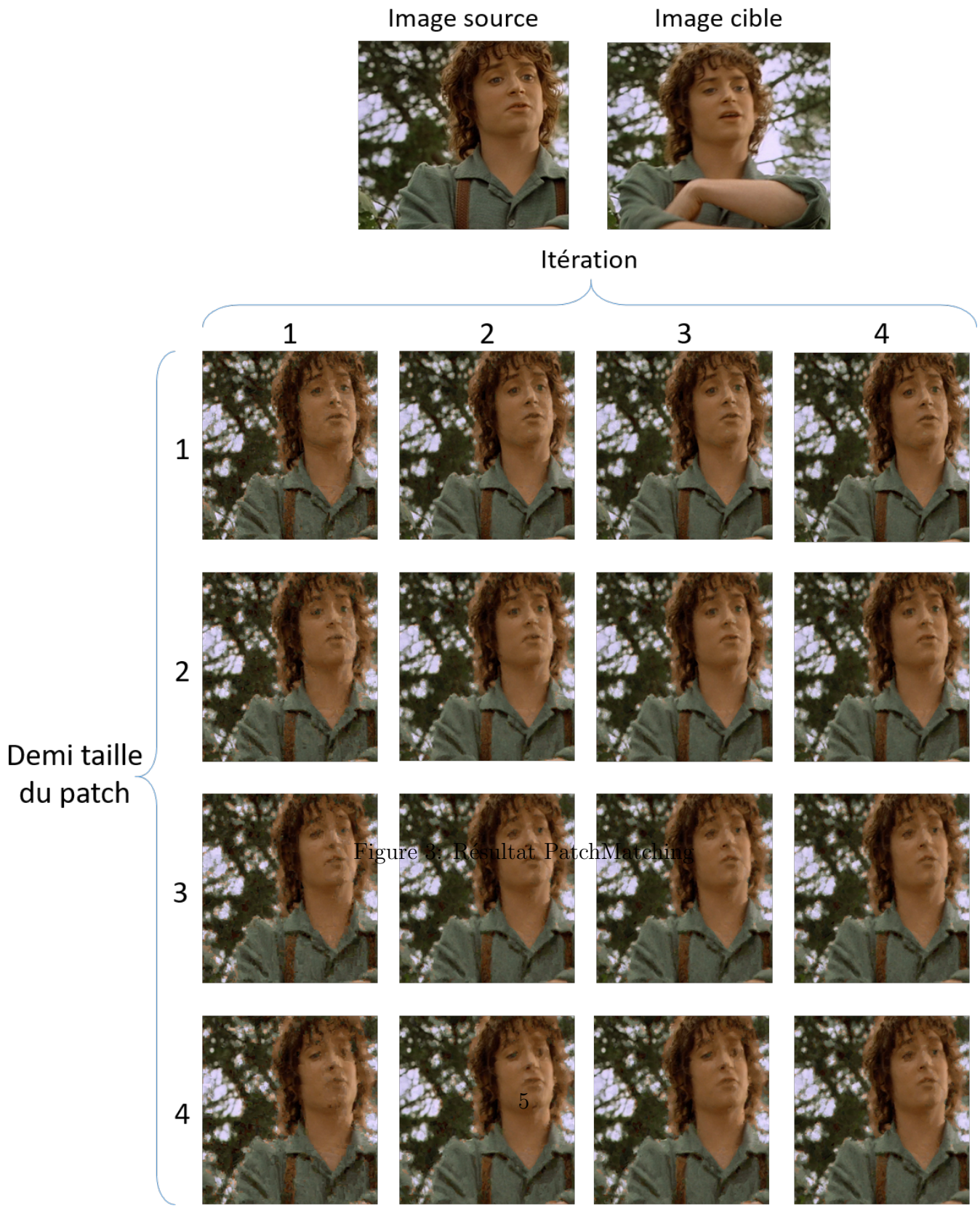

Figure 3: Résultat PatchMatching

2.3 Utilisation pour la synthèse de texture

La méthode PatchMatch permet d'accélérer la recherche de patch effectué dans la synthèse de texture. On pourrait par exemple à partir d'une texture, créer une nouvelle image de la taille souhaiter, coller une seed de notre texture au centre de cette nouvelle image et appliquer ensuite la méthode PatchMatch : remplir l'ensemble des pixels noirs avec des pixels aléatoire de notre texture et les traités avec la méthode PatchMatch. Cela permettrait d'améliorer grandement la vitesse de recherche de patch car au lieu de parcourir à chaque fois l'ensemble des pixels de notre texture, il suffirait de parcourir au maximum 3 pixels (pondéré par le nombre d'itérations effectués).

3 Conclusion

La méthode PatchMatch permet d'accélérer efficacement les algorithmes utilisant la recherche de patch. En effet, la méthode de PatchMatch n'a besoin au maximum que de 3 patchs pour trouver un patch satisfaisant et il suffit de réitérer l'opération afin d'affiner notre résultat.

References

[1] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), August 2009.