Advenced Methods for Image Processing

Devoir 2, Assignment: PatchMatch

WOLSKI Axel

Chargé de TD : BUGEAU Aurelie

Master 2 Informatique Spécialité Image et Son 2017-2018

Octobre 2017

Contents

1 Explications

Pour ce deuxième rendu en traitement d'images avancés, il nous a été demandé de coder l'algorithme de **PatchMach** vue en en cours [1]. Celui-ci consiste à reconstituer une image d'origine (l'image A figure1 dans notre cas) à partir d'une autre image similaire (l'image B figure2) dans une image de destination (image C figure3) qui est l'association de pixels de l'image d'origine avec les pixels de la seconde image. Nous verrons plus en détail le fonctionnement de cette algorithme dans la partie suivante.

Figure 1: Image A

Figure 2: Image B

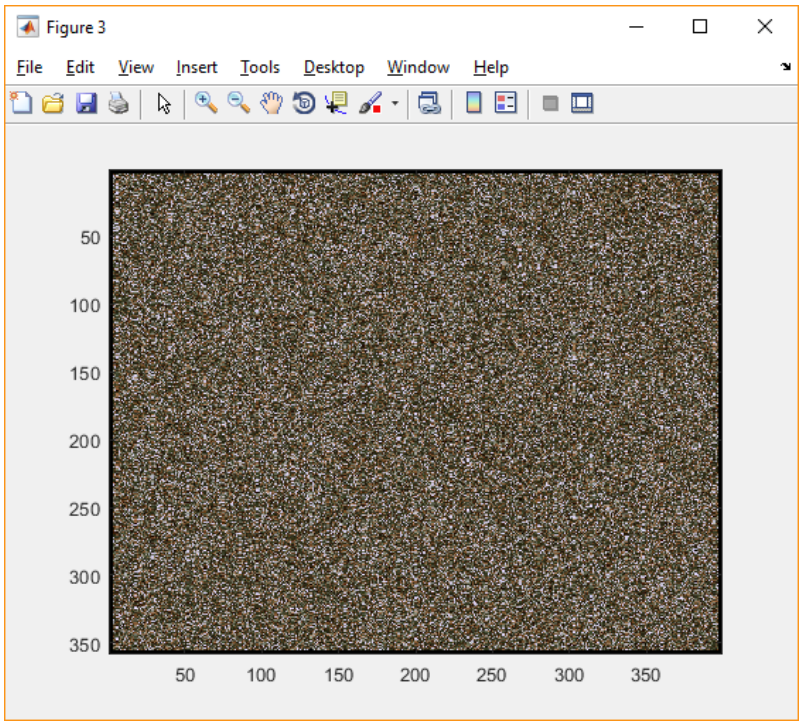

Figure 3: Image C

1.1 PatchMatch

1.1.1 Algorithme

Nous allons dérouler pas à pas l'algorithme de PatchMatch vue en cours [1] à partir du slide 22. Premièrement, nous avons besoin de deux images initiales, nous les appellerons image **A** (figure1) et image **B** (figure2). Ces deux images devront être similaires au niveau des couleurs. Dans notre cas, les deux images représentent le même personnage dans un même décors mais avec une posture différente.

La première étape est d'associer chaque pixels de l'image **A** à un pixel aléatoire dans l'image **B**. Cette association est appelée **NNF**, qui correspond à *Nearest Neighbor Field*. La **NNF** entre **A** et **B** sera visible dans l'image **C** (figure3), dans laquelle à chaque position `(x,y)` de **A** nous retrouverons un pixel provenant de **B**. Cette association est représentée dans la figure4. Dans mon cas je créé un tableau **NNF** dans lequel je stock à la position du pixel `(x,y)` de **A**, la coordonnée `(i,j)` du pixel correspondant dans **B**.

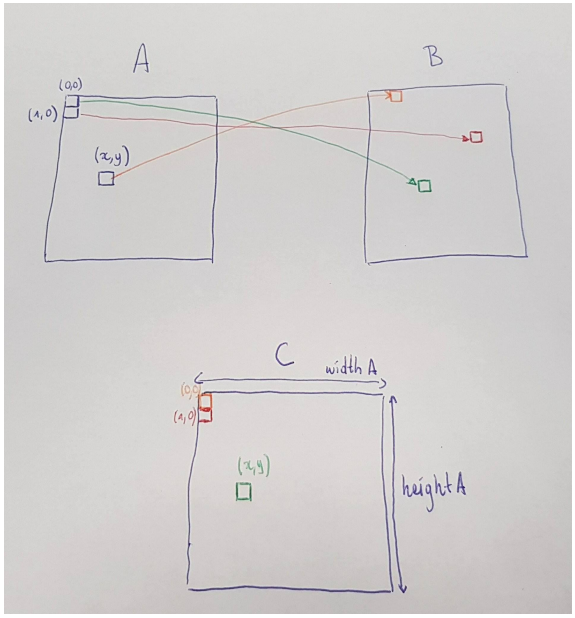

Figure 4: **NNF:** Association entre pixel de l'image A avec l'image B

La **NNF** étant initialisée, il faut maintenant la parcourir de gauche à droite et de haut en bas tel que sur la figure5 afin de calculer les **SSD** de la **NNF**. Soit calculer la **SSD** entre un patch de **A**, centré sur les coordonnées recherchées, et un patch de **B**, centré aux coordonnées du pixels associées à **A** dans la **NNF**.

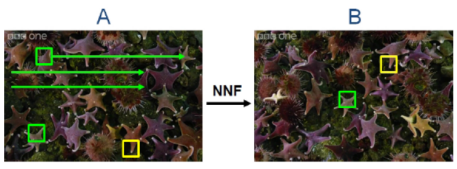

Figure 5: Parcours de la **NNF**

La prochaine étape de cet algorithme est la plus compliquée à comprendre (voir article de Barnes[2]). Elle consiste à comparer la **SSD** de la **NNF** aux coordonnées `(x,y)` de **A** avec la **SSD** de la **NNF** aux coordonnées `(x,y-1)` de **A** et avec la **SSD** de la **NNF** aux coordonnées `(x-1,y)` de **A**. Soit la comparaison entre un pixel donné, son voisin de gauche et son voisin du dessus. Cette comparaison est représentée dans la figure6 et elle est nécessaire afin de sélectionner la plus petite **SSD**.

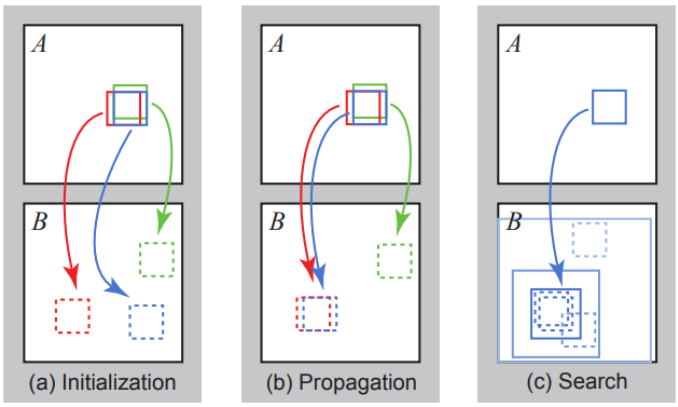

Figure 6: Recherche de la meilleure **SSD**

La recherche de cette plus petite **SSD** est nécessaire afin de placer aux coordonnées de **A**, où l'on se trouve, le patch correspondant dans l'image **C**. Il faudra alors mettre à jours la **NNF** qui contiendra à présent les coordonnées du centre de ce patch.
La seule difficulté dans cette étape est que, lorsque l'on compare la **SSD** des patchs voisins dans l'image **A** il faut également prendre en compte les voisins dans l'image **B** tel que représenté dans la figure7.

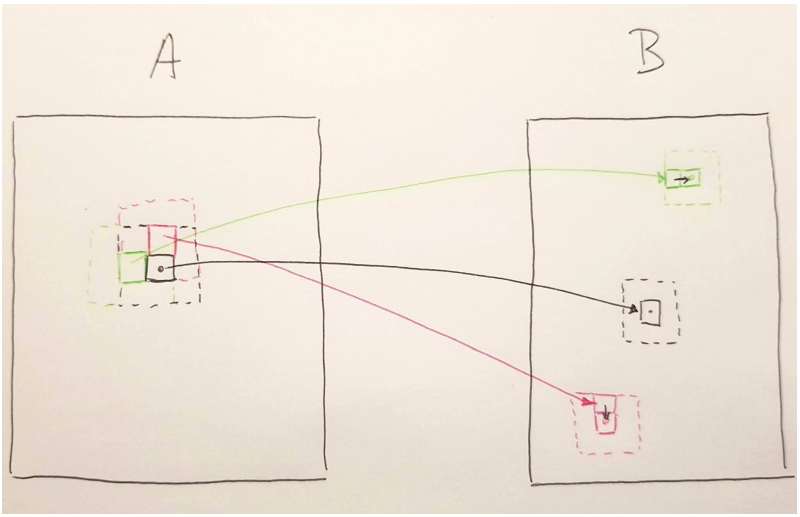

Figure 7: Sélection des patchs dans **B**

Nous parcourons donc toute l'image de cette manière et cela nous permet de recréer petit à petit l'image **A**. La figure8 représente l'image **C** en cours de génération à l'aide de cet algorithme.

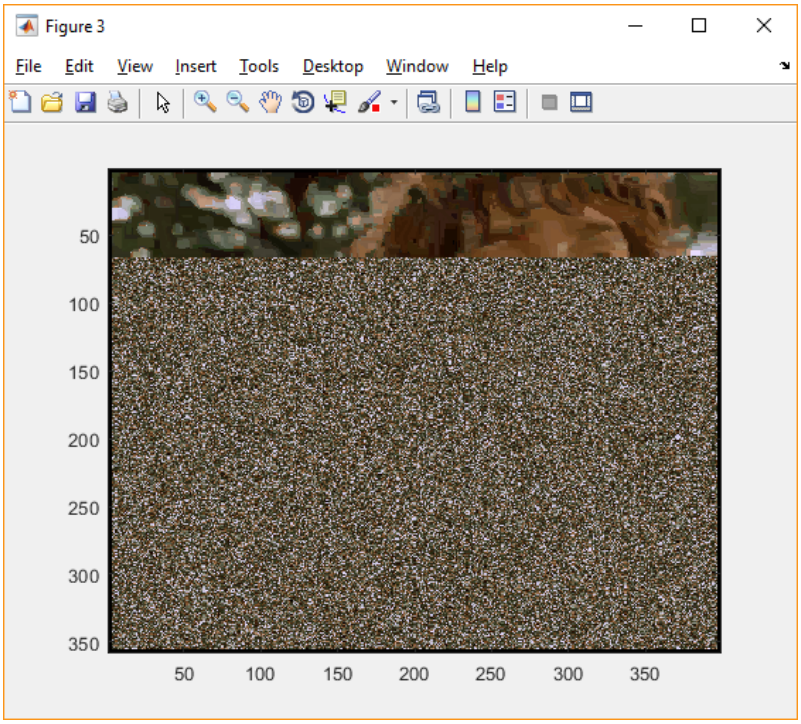

Figure 8: Génération de l'image **C** en cours de traitement

Lorsque ce parcours est terminé, il ne nous reste plus qu'à reprendre la même méthode, mais en parcourant l'image de bas en haut et de droite à gauche. Il faudra donc choisir le voisin à droite et celui de bas lors de la recherche de la plus petite **SSD**.
Afin d'affiner le résultat obtenu à l'aide de `PatchMatch` il est possible de réaliser plus de passages. Les résultats des figures suivante ont été obtenu avec 1, puis 2, puis 10 et 20 passages.

Figure 9: Image C à l'aide d'un passage

Figure 10: Image C à l'aide de deux passages

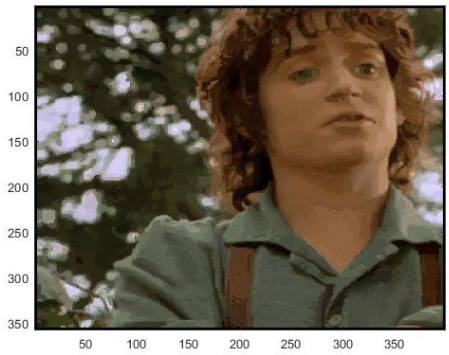

Figure 11: Image C à l'aide de dix passages

Figure 12: Image C à l'aide de vingt passages

1.1.2 Résultats

On peut constater qu'avec l'algorithme de `PatchMatch`, plus nous effectuons de passage, meilleur est le résultat. On constate bien qu'avec un seul passage sur la figure9, le résultat est très bruité. Alors qu'avec deux passages, l'image obtenue figure10 est bien meilleure, et les figure11 et figure12 le sont encore plus. Si le rendu est meilleur à chaque passage, c'est simplement que chaque pixels voisins dans **C** (donc dans la **NNF**) sont rangés, alors que lors du premier passage **C** est créé de manière aléatoire.

1.2 Lien entre PatchMatch et Texture Synthesis

Lors du cours et à l'aide des publications de Barnes[2] et de Prieler[3], nous avons vue qu'il est possible d'utiliser `PatchMatch` afin d'obtenir des résultats similaires à `Texture Synthesis`. La recherche des plus proches voisins randomisées étant plus efficace que la méthode utilisée dans `Texture Synthesis`. L'idée d'utiliser `PatchMatch` afin de synthétiser une texture parait logique et plus efficace. Le problème est que `PatchMatch` n'optimise pas toute l'image, il se contente seulement d'optimiser l'image au niveau des patchs voisins. Lorsque l'on utilisera ce dernier pour de la synthèse de texture, on obtiendra des artefacts sur l'image et donc une qualité visuelle moins bonne qu'avec `Texture Synthesis`.

2 Conclusion

L'algorithme `PatchMatch` permet de nombreuses choses. Il peut être utilisé pour supprimer des objets d'une images, générer ou agrandir d'autres objets, il peut également être utilisé à la place de `Texture Synthesis`. C'est un algorithme rapide et efficace dans l'utilisation que l'on en fait. Si on le compare aux algorithmes réalisés dans le TP précèdent, on obtient nos résultats bien plus rapidement.
Mon implémentation de l'algorithme `PatchMatch` est divisé en quatre fichiers:

- `SSD.m` la fonction qui calcul la **SSD**.
- `myPatch.m` la fonction qui créé un patch.
- `assoAB.m` la fonction qui associe un pixel de **A** avec un pixel dans **B**.
- `PatchMatch.m` le fichier qui effectue l'algorithme.

Cette version peut être améliorée car elle ne gère pas les bords et elle fait appel à des boucle `for`. J'ai pendant un moment voulu pré-calculer les **SSD** comme je le fait pour la **NNF** afin d'éviter de les recalculer à chaque passages, mais cela m'a paru plus compliqué, en revanche cela pourrait faire office d'une optimisation. Cependant j'ai évité au plus possible la duplication de code, qui à longtemps été présente dans mon algorithme.

3 Éléments bibliographiques

- Cours de traitement d'images avancés [1].
- PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing [2].
- Patchmatch for Texture Synthesis [3].

References

[1] Aurélie Bugeau. Advanced image processing, accelerating. https://moodle1.u-bordeaux.fr/pluginfile.php/343016/mod_resource/content/0/Acceleration.pdf. 2, 3, 8

[2] A. Finkelstein C. Barnes, E. Shechtman and D.B. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. http://gfx.cs.princeton.edu/pubs/Barnes_2009_PAR/patchmatch.pdf, 2009. 4, 6, 8

[3] Stefan Jeschke Daniel Prieler. Patchmatch for texture synthesis. https://www.cg.tuwien.ac.at/research/publications/2011/prieler_11_patchmatch/prieler_11_patchmatch-report.pdf, 2011. 6, 8