



Advanced Methods for Image Processing

Devoir 3, Assignment: Minimal Cut between patches

WOLSKI Axel

Chargé de TD : BUGEAU Aurelie

Master 2 Informatique Spécialité Image et Son 2017-2018

Novembre 2017

Contents

1	Explications	2
1.1	Min Cut	3
1.1.1	Algorithme	3
1.1.2	Résultats	5
1.2	Graph Cut	6
1.3	Comparaison des résultats entre Min Cut et Graph Cut	7
1.4	Application pour Texture Synthesis	9
2	Conclusion	9
3	Éléments bibliographiques	10

1 Explications

Pour ce troisième rendu en traitement d'images avancés, il nous a été demandé de coder l'algorithme de **Min Cut** vue dans le cours **Accélération**[2] et l'algorithme **Graph Cut** vue dans le cours **Graph Cuts**[3]. Ils ont pour but de construire une image (image C figure 4) en fusionnant deux images (image A figure 1 et image B figure 2) similaires entre elles (dans notre cas deux textures identiques), à l'aide d'un découpage (figure 5) réalisé sur la partie où les deux images se chevauchent (représenté en figure 3). Nous verrons plus en détail le fonctionnement de ces algorithmes dans les parties suivantes.

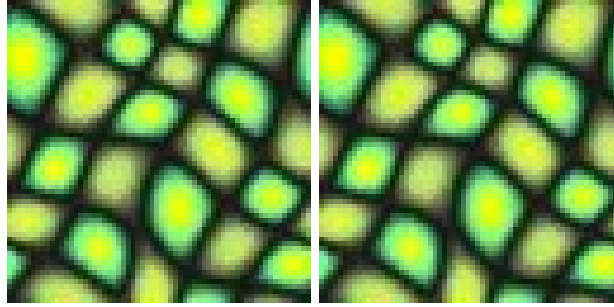


Figure 1: Image A

Figure 2: Image B

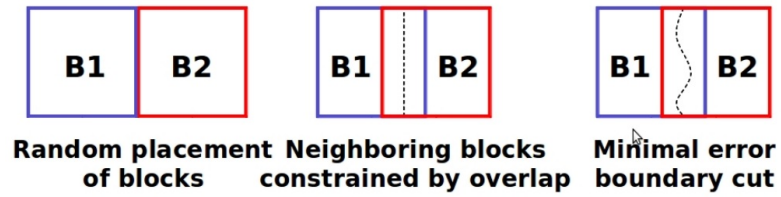


Figure 3: Fonctionnement du découpage

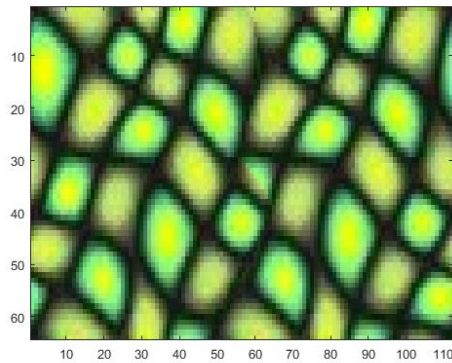


Figure 4: Image C

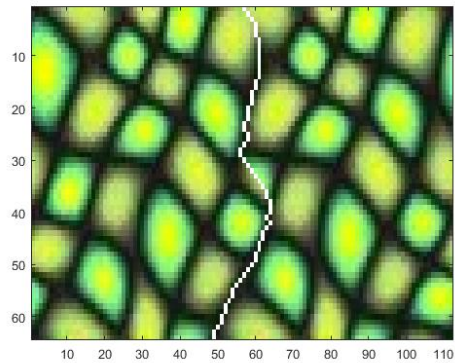


Figure 5: Image C avec frontière de découpage

1.1 Min Cut

1.1.1 Algorithme

Nous allons dérouler pas à pas l'algorithme de **Min Cut** vue en cours [2] à partir du slide 10.

Dans un premier temps nous devons initialiser toutes nos variables. C'est à dire récupérer notre image A et notre image B (figure 1 et figure 2), nous devons ensuite définir une valeur pour **overlap** qui correspond à la taille en largeur du nombres de pixels de A et B qui vont se chevaucher (représenté figure 3 image du centre). Nous pourrions également initialiser une image vide de la taille :

$\text{height}(A) \times (\text{width}(A) \times 2 - \text{overlap})$ qui correspondra à la taille de l'image résultat. Il va nous falloir également récupérer la partie de l'image de A (que nous appellerons **leftP**) et la partie de l'image de B (**rightP**) qui vont se chevaucher (comme représenter dans la figure 6).

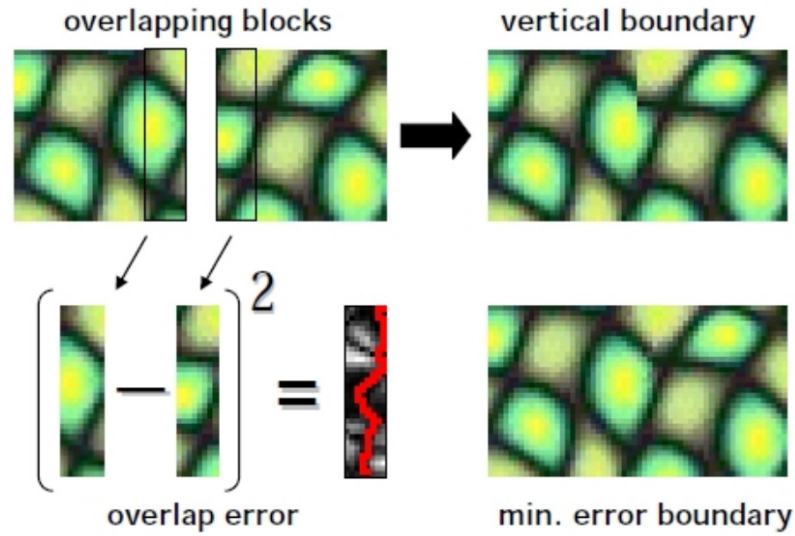



Figure 6: Calcul de overlap error

Nous devons calculer l'erreur entre leftP et rightP (valeurs en rouge figure 7) afin de déterminer le poids de chacun des pixels (valeurs en noires figure 7) et pouvoir construire une tableau M correspondant à la figure 7. Pour cela chaque pixels de la première ligne de M prend pour poids l'erreur entre leftP et rightP au pixel qui lui correspond. A partir de la deuxième ligne le poids du pixel correspondra à l'erreur calculée lui correspondant que l'on additionne au poids minimal d'un de ces 3 voisins se situant au dessus de lui. Dans la situation où l'on se trouverais sur un bord, nous ne devrions vérifier que les deux voisins du dessus (voir figure 7).

Algorithm Direction




1 1	4 4	3 3	5 5	2 2
3 + 1 = 4	2 + 1 = 3	5 + 3 = 8	2 + 2 = 4	3 + 2 = 5
5	2	4	2	1

Figure 7: Tableau des poids de chaque pixels de l'overlap

Dans mon implémentation je réalise également un tableau C de la même taille de M, dans le quel je stock la coordonnée x qui correspond au x du plus petit des trois voisins du dessus. Lorsque nous avons remplis entièrement M et C, nous pouvons passer à la dernière étape.

Algorithm Direction



1 1	4 4	3 3	5 5	2 2
3 4	2 3	5 8	2 4	3 5
5 8	2 5	4 7	2 6	1 5

Figure 8: Détermination du chemin de la frontière entre l'image A et l'image B

Il nous faut nous placer sur la dernière ligne (celle correspondant à $y = \text{height}(A)$), nous recherchons sur celle-ci la plus petite valeur de M sur cette même ligne. Lorsque celle-ci est trouvée nous pouvons alors remonter pas à pas en récupérant la coordonnée stockée dans $C(x,y)$ correspondant a $M(x,y)$. C'est à dire pour $M(y,x)$ nous pouvons, à l'aide de $C(y,x)=x_c$ récupérer $M(y-1,x_c)$ qui correspond au plus petit voisin de $M(y,x)$. On remonte comme cela jusqu'à $y=1$ afin d'obtenir une frontière entre l'image A et l'image B comme dans la figure 8. Dans notre image finale (figure 4) nous pouvons remplir toute la partie à gauche de la frontière calculée avec des pixels provenant de A, la partie à droite de la frontière avec des pixels provenant de B et à la frontière nous calculons la moyenne entre le pixel A et B correspondant.

1.1.2 Résultats

Nous allons maintenant comparer différents résultats pour un overlap égal à 5 (figure 18 et figure 10), 16 (figure 20 et figure 12) et 25 (figure 22 et figure 14) pour la texture 1 ainsi qu'un overlap = 16 pour la texture 2 5 (figure 24 et figure 16).

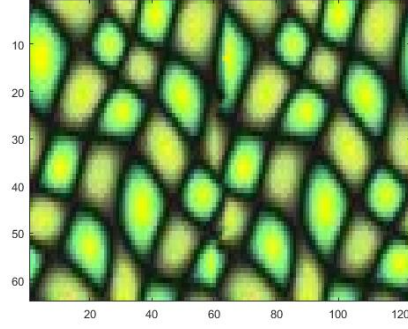


Figure 9: Image C avec overlap = 5

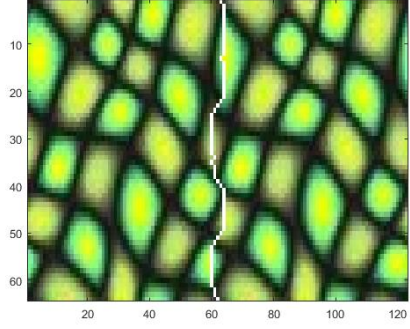


Figure 10: Image C avec frontière de découpage avec overlap = 5

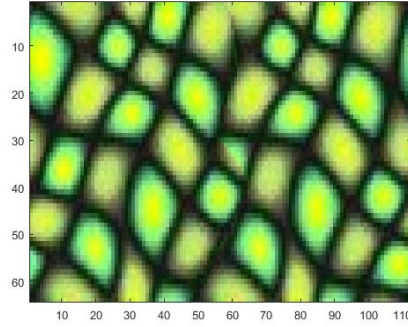


Figure 11: Image C avec overlap = 16

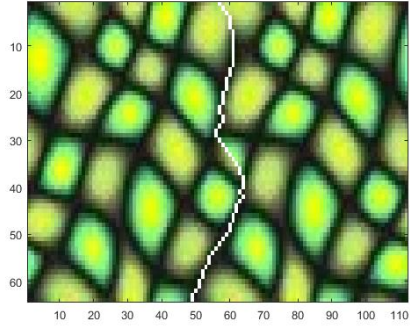


Figure 12: Image C avec frontière de découpage avec overlap = 16

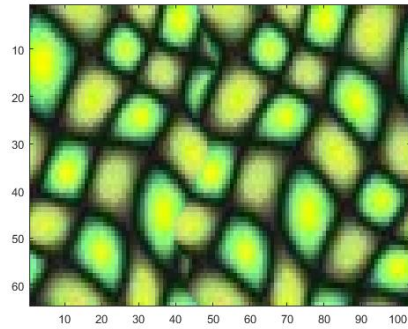


Figure 13: Image C avec overlap = 25

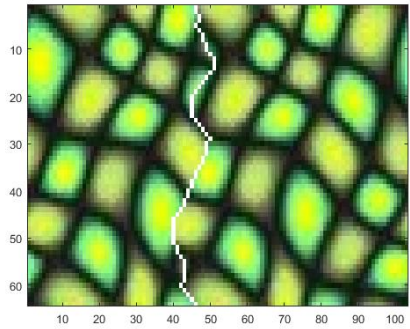


Figure 14: Image C avec frontière de découpage avec overlap = 25

On peut constater qu'en fonction de l'overlap que l'on choisi, le résultat est plus ou moins satisfaisant (nous pouvons voir la délimitation entre les deux images). Il est donc nécessaire de réaliser plusieurs tests et de choisir un overlap qui nous convient. Dans mon cas j'ai travailler avec un $\text{overlap} = 16$.

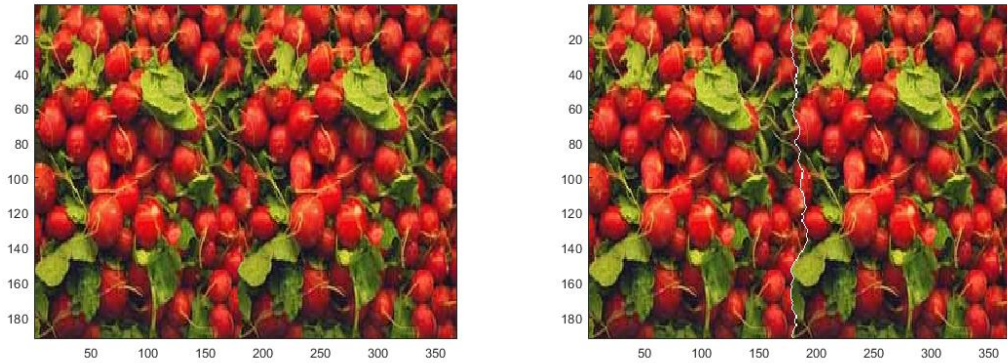


Figure 15: Image C, $\text{overlap} = 16$ avec text2 Figure 16: Image C avec frontière de découpage, $\text{overlap} = 16$ avec text2

Dans le cas de la texture 2, nous pouvons clairement distinguer la délimitation entre la partie gauche et droite (plus ou moins forte en fonction de la valeur de l'overlap). Cela provient du fait qu'à gauche de la texture source nous avons beaucoup de feuilles vertes, qui au contraire du coté droit de l'image sont beaucoup moins présentes. Par conséquent, peu importe le découpage : si d'un coté la couleur est rouge et que de l'autre elle est verte, nous verrons une délimitation à certains endroit.

Globalement nous pouvons conclure que le découpage ce fait de manière plutôt correct et permet de faire des fusions satisfaisantes avec une bonne taille d'overlap.

1.2 Graph Cut

Graph Cut est globalement très similaire à **Min Cut** comme nous avons pu le voir en cours [3] ainsi que dans l'article de Boykov al. [4].

Afin d'implémenter le code de **Graph Cut** il nous a simplement fallu récupérer le code présent dans l'exemple `GCMex_test.m` et appliquer les modifications nécessaires pour l'utiliser avec **Graph Cut**.

Paramétrage de GCmex :

- `class = zeros(N,1)`: `class` est un vecteur qui spécifie chacune des étiquettes, avec $N = \text{height}(\text{image A}) * \text{width}(\text{image A})$.
- `unary = (nombres d'image à fusionner, N)`: `unary` contiendra le coût de chaque pixel.
- `pairwise = sparse(N,N)`: `pairwise` contiendra le poids des voisins.
- `labelcost = [0,1 ; 1,0]`: matrice contenant le coût des noeuds.
- `maskTMP`: matrice qui stocke les pixels afin de pouvoir fusionner les deux images

Lors du parcours de chaque pixel de l'overlap (comme dans Min Cut), nous stockons dans `pairwise` le poids des voisins correspondant au pixel courant, à l'aide de la fonction `smoothness` (figure 17).

$$E_{\text{smoothness}}(\mathbf{X}) = \sum_{(p,q) \in \mathcal{E}} (|A(p) - B(p)| + |A(q) - B(q)|) \delta(X_p - X_q)$$

Figure 17: Calcul du poids des voisins pour Graph Cut

Lorsqu'on se situe sur l'extrémité de l'overlap en largeur (`row=1` et `row=width(A)`) nous appliquons un coût fort dans `unary`, car sur les extrémités nous devons conserver les images d'origines correspondantes. Soit pour `row=1` nous garderons les pixels de A et pour `row=width(A)` nous garderons les pixels de B. Cette méthode nous permet d'obtenir une liste de label grâce à laquelle nous pourrions déterminer quelle est la partie de l'image A (`maskA` dans mon implémentation) à garder ainsi que celle que nous garderons de B (`maskB`).

1.3 Comparaison des résultats entre Min Cut et Graph Cut

Nous allons maintenant comparer les résultats obtenue pour **Min Cut** et pour **Graph Cut** avec les même valeurs pour overlap.

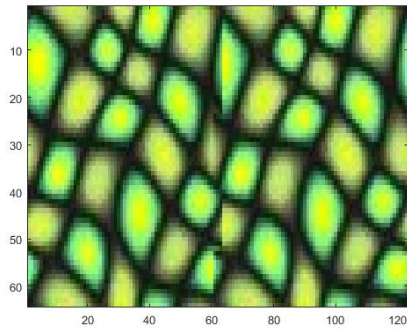
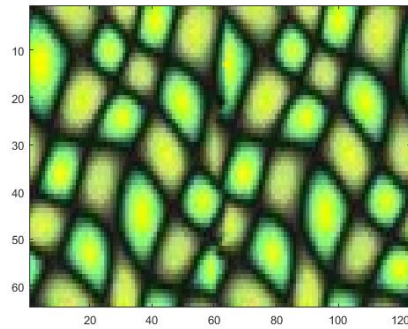


Figure 18: Min Cut avec overlap = 5

Figure 19: Graph Cut avec overlap = 5

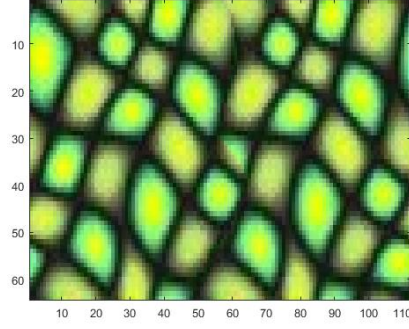


Figure 20: Min Cut avec overlap = 16

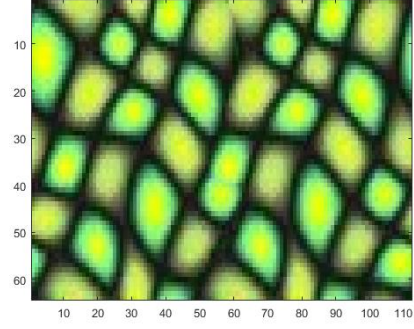


Figure 21: Graph Cut avec overlap = 16

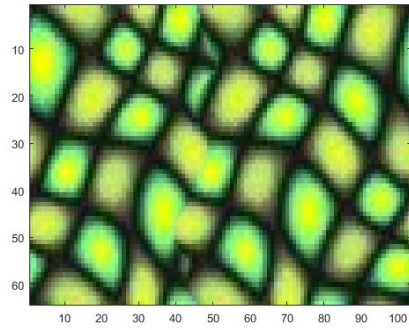


Figure 22: Min Cut avec overlap = 25

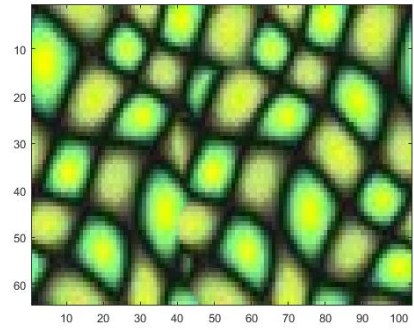


Figure 23: Graph Cut avec overlap = 25

Nous constatons donc que comme précédemment les résultats sont plus ou moins acceptables en fonction de l'overlap choisie. Dans le cas où l'on a l'overlap = 16 (le meilleurs cas) nous pouvons constater que le résultat avec **Graph Cut** est relativement meilleur que celui obtenue avec **Min Cut**.



Figure 24: Min Cut overlap = 16 avec text2



Figure 25: Graph Cut overlap = 16 avec text2

Dans le cas de la texture 2 avec une taille d'overlap = 16, le résultat de **Graph Cut** est également meilleur que celui de **Min Cut**, il y a beaucoup moins de problèmes liés aux feuilles.

1.4 Application pour Texture Synthesis

Dans mon implémentation j'ai directement travaillé sur une image entière que j'ai dupliqué. Ces deux algorithmes peuvent aussi bien être utilisés avec des patches. On peut donc utiliser ces deux algorithmes pour faire du **Texture Synthesis**, il suffit d'avoir deux images/patches que l'on superpose en continue gauche/droite et haut/bas. Cela nous permet de générer une plus grande image, et par conséquent de réaliser un résultat similaire à celui que l'on peut obtenir à l'aide de **Texture Synthesis**.

2 Conclusion

Les algorithmes **Min Cut** et **Min Cut** permettent de nombreuses choses. Ils peuvent être utilisés pour agrandir ou réduire une image, fusionner entre elles différentes parties d'images, ainsi que dans beaucoup de photomontages [1] et ils peuvent également être utilisés à la place de **Texture Synthesis**. Ce sont des algorithmes rapides et efficaces dans l'utilisation que l'on en fait. Cependant **Graph Cut** semble obtenir de meilleurs résultats que **Min Cut**.

Mes implémentations de ces algorithmes sont divisées en quatre fichiers:

- `overlapError.m` la fonction qui calcul l'erreur de overlap.
- `minCut.m` l'algorithme Min Cut.
- `smoothness.m` la fonction qui calcul de poids des voisins d'un pixel.
- `graphCut.m` l'algorithme Graph Cut.

J'ai essayé d'optimiser au mieux ces deux algorithmes, en évitant le plus possible des boucles `for` inutiles. Cependant je pense qu'en passant mes images en niveau de gris cela permettrait un gain de temps et optimiserait mieux mon code notamment pour l'algorithme de **Graph Cut** dans lequel à un moment je suis obligé d'appliquer mon masque séparément sur chaque canal, puis je re-bascule le tout en RGB.

3 Éléments bibliographiques

- Cours de traitement d'images avancés sur Acceleration [2].
- Cours de traitement d'images avancés sur Graph Cuts [3]
- Graph Cut: Interactive Graph Cuts for Optimal Boundary Region Segmentation of Objects in N-D Images [4].
- Différente application de Graph Cut : Interactive Digital Photomontage[1].

References

- [1] M. Agrawala S. Drucker A. Colburn A. Agarwala, M. Dontcheva. Interactive digital photomontage. <http://kneecap.cs.berkeley.edu/papers/photomontage/photomontage.pdf>, 2004. 9, 10
- [2] Aurélie Bugeau. Advanced image processing, accelerating. https://moodle1.u-bordeaux.fr/pluginfile.php/343016/mod_resource/content/0/Acceleration.pdf. 2, 3, 10
- [3] Aurélie Bugeau. Advanced image processing, graph cuts. https://moodle1.u-bordeaux.fr/pluginfile.php/304243/mod_resource/content/2/GraphCuts.pdf. 2, 6, 10
- [4] Marie-Pierre Jolly Yuri Y. Boykov. Interactive graph cuts for optimal boundary region segmentation of objects in n-d images. <http://www.csd.uwo.ca/~yuri/Papers/iccv01.pdf>, 2001. 6, 10