# Projet d'INF552

## Fusion d'images

Xavier Fontaine et Pierre Pakey

12/01/2015

# Introduction

Le but de ce projet d'Informatique est de réussir à recoller plusieurs images différentes sans que cela ne se voit. Les utilisations d'un tel recollage sont multiples. On peut dans un premier temps envisager de recoller plusieurs fois la même image sur elle-même afin d'obtenir une image plus grande. On peut ainsi, à partir d'un petit champ de blé en obtenir un plus grand. Une deuxième application de cette technique est de fusionner des images a priori très différentes pour en former une nouvelle. On peut par exemple créer des paysages artificiels à partir de deux photographies existantes.

Ce rapport est très largement inspiré de techniques présentées dans l'article GraphCuts Textures [1], qui utilise la technique de coupe minimale dans un graphe, que l'on peut aisément obtenir à l'aide de l'algorithme de Ford-Fulkerson.

On tâchera donc ici de présenter les algorithmes utilisés dans ce projet, les implémentations réalisées en C++, ainsi que les résultats obtenus.

# 1 Principe de l'algorithme

## 1.1 La technique de Graph Cuts

On souhaite dans ce projet réaliser la fusion de deux images. Pour ce faire, on commence par considérer une superposition partielle des deux images que l'on veut recoller. Pour chaque pixel de la zone de recouvrement, notre algorithme va devoir choisir s'il prend le pixel de l'image 1 ou celui de l'image 2. On est donc face à un **problème d'étiquetage binaire**, et cela fait naturellement penser à la technique de coupes dans un graphe.

Le principe fondamental de la technique de GraphCuts est de représenter la zone de recouvrement par un graphe orienté. Chaque pixel de la zone de recouvrement est en effet un noeud de notre graphe. Ce graphe peut donc être vu comme une grille de pixels. Deux pixels adjacents sont reliés par deux arcs orientés (chacun dans un sens) et valués. La fonction de coût sera détaillée plus loin mais est essentiellement une mesure de la différence entre les deux pixels considérés. On rajoute ensuite deux noeuds au graphe, classiquement notés s et t, et appelés *source* et *puits*. La source représente par exemple l'image 1 et le puits l'image 2. On relie la source aux pixels que l'on souhaite forcer à provenir de l'image 1 dans l'image finale, par des arcs de capacité infinie. On fait de même en reliant le puits aux pixels de l'image 2 que l'on souhaite forcer à appartenir à l'image finale, avec des arcs de capacité infinie.

Le but est alors d'obtenir une coupe minimale dans le graphe ainsi construit, "minimale" signifiant que le poids des arêtes coupées doit être le plus petit possible. Dans l'image finale, les pixels situés du côté de la source prendront donc la valeur des pixels de l'image 1 et ceux du côté du puits la valeur des pixels de l'image 2. On construit ainsi l'image finale exclusivement à partir des pixels des images 1 et 2. On comprend donc bien la nécessité de réaliser une coupe de faible coût afin d'avoir la transition la plus douce possible entre les deux images que l'on cherche à recoller.

## 1.2 Implémentation

Pour implémenter l'algorithme recherchant une coupe minimale dans un graphe, on utilise la bibliothèque `maxflow` déjà utilisée en TP. On doit cependant choisir soi-même une fonction de coût adaptée au problème. L'article précédemment cité propose la fonction suivante : si l'on considère deux images $I_1$ et $I_2$, le coût de l'arête entre les pixels a et b sera $c(a,b) = ||I_1(a) - I_2(a)|| + ||I_1(b) - I_2(b)||$. On a choisi pour le projet d'utiliser la norme euclidienne.

Comme suggéré dans l'article proposé, on a rapidement fait le choix de modifier l'expression de la fonction de coût utilisée, en introduisant dans la norme la valeur du gradient des deux images. En effet, le gradient donne une information intéressante sur les variations au sein d'une image et, afin d'obtenir une coupure peu visible entre les deux images, il peut être judicieux d'inciter l'algorithme à réaliser la coupe au niveau d'une forte variation d'une image d'origine. C'est pour cela que l'on a divisé la norme précédemment obtenue par une puissance du gradient. En divisant par une puissance (inférieure à 1) du gradient et non par le gradient lui-même, on s'éloigne un petit peu de l'article. Ce choix a semblé pertinent afin d'obtenir une coupure moins visible dans l'image finale.

# 2 Applications

On va présenter dans cette section trois applications que l'on a réalisées dans ce projet. Les deux premières sont assez semblables et concernent la fusion de deux images différentes, qui s'est en fait révélée être plus facile

1. *Graphcut Textures : Image and Video Synthesis Using Graph Cuts*, par Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk et Aaron Bobick

à réaliser que le recollement de la même image plusieurs fois. On traitera ce dernier problème comme troisième application.

## 2.1 Fusion horizontale d'images distinctes

On a commencé par s'intéresser à la fusion de deux images différentes, en utilisant un exemple suggéré dans l'article.

Le but est ici de fusionner les deux images suivantes, qui contiennent toutes les deux une étendue d'eau.

FIGURE 1 – Maison

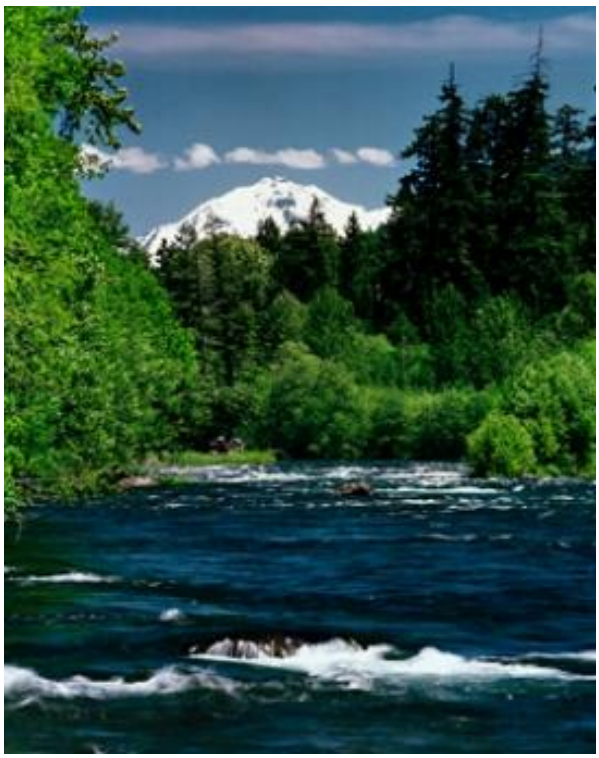

FIGURE 2 – Montagne

La disposition des deux images laisse supposer que la coupure sera à peu près horizontale. Ce cas particulier peut être codé directement dans la fonction. On impose ainsi que l'image finale contienne le bas de l'image de gauche, ainsi que le haut de l'image de droite.

Le résultat obtenu est assez satisfaisant. Comme la coupure reste assez franche entre les deux images, on peut penser à lisser l'image obtenue à l'aide d'un filtre qui pourrait s'apparenter à un filtre gaussien. Le résultat n'est pas fondamentalement différent, mais est un peu plus flou au niveau de la ligne de séparation. Cela est sans doute dû au fait que l'on n'applique le filtre que sur une étendue de 2 pixels de part et d'autre de la ligne de coupure entre les deux images.

FIGURE 3 – Résultat avec la ligne de coupure

FIGURE 4 – Résultat obtenu avec lissage

Le code de cette première application est donné dans le dossier `application1`.

## 2.2 Fusion de deux images dans le cas général

Dans la partie précédente nous avons vu l'exemple d'une fusion horizontale de deux images. On souhaite désormais s'intéresser à un cas plus général dans lequel les parties des images que l'on fixera dans l'image finale ne seront pas forcément en haut ou en bas des images d'origine, mais par exemple au milieu.

On a donc codé, dans le dossier `application2`, des fonctions supplémentaires permettant à l'utilisateur de dessiner un quadrilatère convexe dans les images d'origine, puis de choisir (à l'aide de la console) si la partie située à l'intérieur ou à l'extérieur du quadrilatère doit figurer dans l'image finale. Pour ce faire, l'utilisateur choisit sur chaque image les quatre sommets du quadrilatère, **dans le sens horaire**. On a ensuite codé une fonction qui, pour chaque pixel de l'image, indique s'il est dans le quadrilatère ou non. Il suffit pour ce faire de regarder les angles entre les côtés du quadrilatère et les segments joignant le pixel courant aux sommets du quadrilatère. L'étude du signe d'un déterminant suffit à obtenir le résultat.

La suite du programme est similaire à celle de la première application présentée dans ce rapport. La seule différence concerne les pixels de l'image finale qui sont forcés d'appartenir à l'image 1 ou à l'image 2. Ce ne sont plus les pixels du haut ou du bas de l'image mais ceux déterminés par l'utilisateur.

On a cherché à fusionner les deux images suivantes :

FIGURE 5 – Dauphin

FIGURE 6 – Lac de l'École polytechnique

La sélection de quadrilatères sur les deux images permet d'obtenir le résultat suivant :

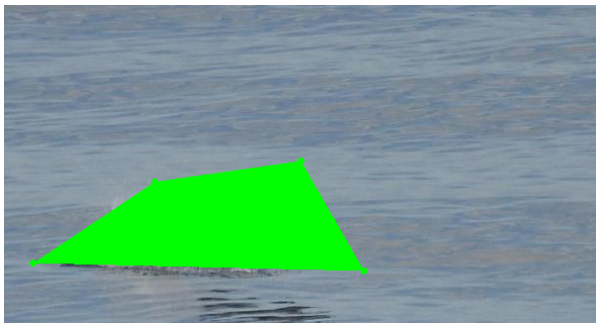

FIGURE 7 – Sélection du dauphin

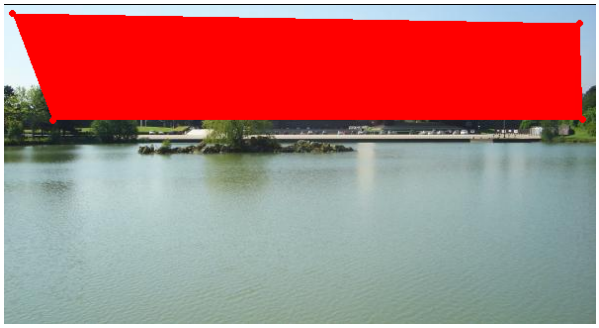

FIGURE 8 – Sélection des bâtiments et du ciel

FIGURE 9 – Résultat final

## 2.3 Fusions multiples de la même image

Le cas de la fusion multiple d'une même image est un peu plus complexe que la fusion simple de deux images différentes. En effet, il faut réaliser un nombre plus important de recollements et les coutures risquent d'être davantage visibles.

Le but dans cette partie est de réaliser une image plus grande à partir d'un même motif. On a choisi dans cette partie de recopier l'image suivante :

FIGURE 10 – Image de départ : pièces de monnaie

Dans le code fourni dans le dossier `application3`, on choisit de créer une image à peu près neuf fois plus grande, obtenue par superposition de neuf images d'origine.

Les processus de fusion par Graph Cuts expliqués précédemment sont à nouveau utilisés entre les images que l'on ajoute successivement à l'image finale. Il y a cependant une différence notable dans l'algorithme utilisé. En effet, puisque l'on réalise plusieurs fusions, il peut arriver qu'une image soit rajoutée sur une zone où il y a déjà eu une couture. On doit donc prendre en compte cette couture comme un nouveau noeud dans le graphe que l'on considère, comme cela est décrit dans l'article utilisé. Pour cela, il faut réussir à garder en mémoire les emplacements des coutures effectuées aux itérations précédentes, ainsi que la valeur des arêtes coupées. Il faut aussi mémoriser de quel échantillon faisait partie chaque pixel situé à l'extrêmité d'une arête coupée. C'est le rôle des matrices `seam` ainsi que `seam_pix` du code fourni.

On obtient alors le résultat suivant :

FIGURE 11 – Résultat final

Même si l'on remarque une probable répétition du même motif, les coutures sont très peu visibles, ce qui fait de ce résultat une image très satisfaisante.

# 3 Approfondissements possibles

Attardons-nous encore un peu sur la fusion multiple d'une même image. Une fois la "grande" image obtenue, le travail n'est pas complétement terminé. L'article suggère en effet de continuer à fusionner des bouts de l'image d'origine sur le résultat, afin de gommer certaines coutures trop visibles et de parfaire la solution.

Pour ce faire, les auteurs de l'article procèdent en deux étapes :

1. Ils commencent par déterminer dans l'image finale les zones qui ne sont pas satisfaisantes
2. Ils collent sur ces zones des bouts de l'image d'origine. Les auteurs présentent trois méthodes pour choisir ces bouts de l'image d'origine. On a choisi d'implémenter ici la première méthode, qui consiste à choisir aléatoirement une translation dans l'image d'origine. C'est la fonction `random_patch` du code fourni.

Pour le premier point, on a aussi opté pour une méthode aléatoire. C'est à dire que l'on choisit aléatoirement une zone de l'image finale sur laquelle on colle un nouvel échantillon.

Utiliser deux méthodes aléatoires ne semble cependant pas être une bonne solution, et le code fourni (à décommenter pour tester) n'améliore pas l'apparence de l'image finale, et la dégrade même. On voit ceci très bien sur l'exemple suivant :

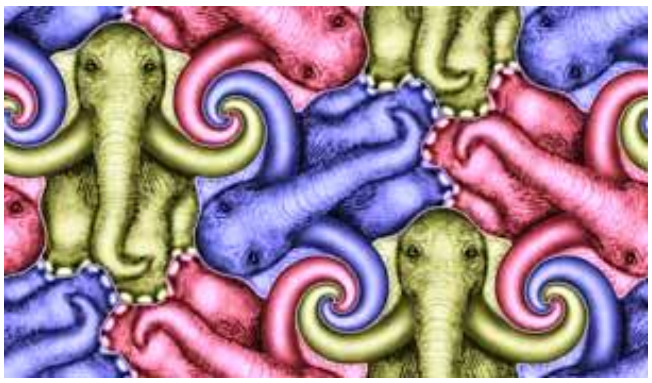

FIGURE 12 – Image de départ : un motif de pavage

FIGURE 13 – Résultat obtenu avec la méthode décrite en 2.3

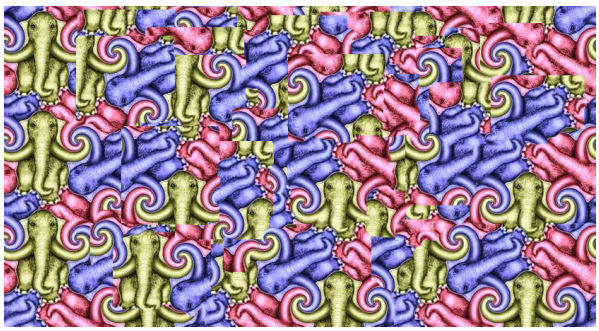

FIGURE 14 – Résultat obtenu en rajoutant aléatoirement des échantillons aléatoires

Un tel résultat peut s'expliquer vraisemblablement par l'abondance d'aléatoire : on rajoute des images aléatoirement à des endroits où la couture est déjà bien réalisée, cela risque de dégrader l'image.

Une amélioration pourrait donc être de déterminer les zones de l'image finale qui sont mauvaises et qui nécessitent la fusion d'une nouvelle image. Une telle détermination semble assez coûteuse dans la mesure où il faut chercher dans la matrice `output` un rectangle de coût trèd élevé. Or le nombre de rectangles de taille k dans une matrice croît de façon exponentielle avec k pour les valeurs de k qui nous intéressent. Un algorithme naïf est donc à proscrire. Le problème pourrait en revanche vraisemblablement se résoudre par programmation dynamique.

Une autre amélioration possible à ce projet serait d'affiner le dessin du quadrilatère présenté dans la section 2.2, afin de pouvoir dessiner n'importe quel polygône concave ou convexe et ainsi être plus précis dans la sélection des parties des images à conserver.

Une troisième amélioration pourrait être d'essayer de fusionner plus de deux images différentes entre-elles. Une utilité pourrait être de retoucher des photos de famille, où chaque membre de la famille n'est bien que sur une seule photo. Comme le problème n'est plus un étiquetage binaire, la technique de Graph Cuts ne devrait pas fonctionner simplement. Une solution pourrait cependant consister en une fusion séquentielle des images, c'est-à-dire que l'on appliquerait les algorithmes présentés précédemment à chaque image l'une après l'autre.

## Conclusion

On a vu ici comment appliquer le problème de recherche d'une coupe minimale dans un graphe à la fusion de plusieurs images. Ce projet confirme une nouvelle fois l'importance des coupes et flots dans un graphe. Afin d'obtenir de meilleurs résultats, il est nécessaire de faire appel à d'autres techniques de vision par ordinateur, comme par exemple le filtrage gaussien qui permet de gommer les irrégularités.

Finalement cette méthode de Graph Cuts doit aussi pouvoir s'appliquer à la fusion de deux vidéos. Cependant, la quantité de mémoire utilisée pour effectuer des calculs de Graph Cuts sur des vidéos est bien plus importante que sur des images à cause de la dimension temporelle, ce qui réduit les applications possibles avec la puissance de calcul disponible.