Rapport Analyse d'Image
TP1

Octobre 2016

Benneton Anna - p1007382
Marot Fabien - p1006013

Sommaire

1/ Introduction

Lors de ce TP nous avons réalisé une application qui a pour but de détecter les contours d'une image quelconque. Cette application est basée sur la librairie *OpenCV*, qui est spécialisée dans le traitement d'image.
Pour plus de confort d'utilisation, nous avons fait le choix d'utiliser une interface graphique (GUI) créée grâce à la librairie GTK. Cette version est pleinement paramétrable par l'utilisateur. Dans le cas où il est impossible d'utiliser GTK, nous avons également fait une version Console (CLI), mais dont les paramètres doivent être changés manuellement. *Voir le Readme pour la compilation et l'utilisation des différentes versions.*

Dans ce rapport, nous allons donc présenter nos choix de structures, nos choix algorithmiques pour implémenter les principales fonctions de traitement, nos résultats les plus pertinents, ainsi que les éléments qui nous paraissent améliorables. Vous trouverez également en annexe un manuel d'utilisation de notre application.

2/ Choix d'implémentation

a. Explication des classes

Notre projet utilises différentes classes, que ce soit pour le traitement d'image ou l'interface :

1. Classe Image

Elle hérite de la classe *cv::Mat* d'*OpenCV.* Cela permet d'utiliser facilement les fonctions de chargement de fichiers, de sauvegarde ou des fonctionnalités de traitement d'image basique (GrayScale, parcours d'image…).
C'est dans cette classe que sont la plupart des méthodes utiles à la détection de contours, ainsi que 3 membres qui permettent de sauvegarder des informations supplémentaires entre chaque traitement : Un tableau des gradients, un tableau d'amplitudes de ces gradients et un tableau de directions de ces gradients.

Ces principales fonctions sont :

- *lissage* (moyen, médian ou gaussien)
- *filtre_differentiel* (filtres de Prewitt, Sobel, Kirsch : selon différentes directions)

- *seuillage* (simple, local, global, hystérésis, hystérésis auto … par moyenne, médiane ou écart-type)
- *affinage* (suppression des épaisseurs de contour)
- *fermeture* (comble les trous dans les contours)
- *detection_contour* (appelle toutes les autres fonctions les unes à la suite des autres)
- *color_direction* (Color les contours en fonction de la direction des gradients)

S'ajoute à celles-ci des fonctions supplémentaires pour passer l'image en nuances de gris, la faire pivoter...

2. Classe Option

Cette classe stocke toutes les variables nous permettant de paramétrer au maximum notre application. Nous les avons regroupées dans une classe afin de pouvoir les passer facilement aux méthodes de la classe Image. En effet, l'objet *option* est le seul paramètre passé aux fonctions, ce qui rend le code plus facilement modifiable et modulable.

3. Classe Filtre

Nous utilisons cette classe pour stocker ou calculer les filtres de convolution que nous allons par la suite appliquer sur l'image. Ces filtres sont tous des matrices carrées *n*n* avec *n* impair.
De la même manière que la classe Image nous avons choisis que notre classe Filtre hérite de la classe cv::Mat pour profiter des possibilitées d'OpenCV.

4. Classe MainWindow

Cette classe est présente uniquement dans la version graphique (GUI) de notre application et nécessite la librairie GTK pour fonctionner. Cette classe gère toute la partie interface et la gestion des signaux lorsque l'utilisateur paramètre son traitement d'image.
De plus dans cette version de l'application il est possible de revenir en arrière/avant à chaque traitement appliqué à l'image ou bien revenir à notre image de base. Ceci est possible grâce à un tableau d'Image : chaque nouveau traitement est considéré comme une nouvelle Image et ajouté à cette liste.
Dans la version console de notre application, cette classe est remplacée par une boucle *While* dans le fichier *main.cpp* du dossier *CLI* et permet de naviguer entre les images traitées via le terminal.

b. Options paramétrables

Toutes les options présentées ici sont les paramètres modifiables par l'utilisateur dans l'interface :

- Options de lissage :

- *type de lissage* : moyen, médian ou gaussien
- *taille du filtre de convolution* : (taille ≥3 et impair 3*3, 5*5...)
- *sigma :* dans le cas d'un lissage gaussien nous pouvons choisir la valeur du sigma pour régler le degré de lissage.

- Options du filtrage directionnel :

- *type de filtre :* nous proposons quatre différents filtres , les trois présents dans le sujet (Prewitt, Sobel et Kirsch), ainsi qu'un filtre laplacien 5*5.
- *direction du filtre :* l'utilisateur peut choisir la direction souhaitée pour l'application du filtre : uni-directionnel (horizontal **ou** vertical), bi-directionnel (horizontal **et** vertical) ou multi-directionnel (gradient boussole).
- *type de norme* : Lors d'un filtrage bi-directionnel, le calcul de l'amplitude du gradient peut être fait soit avec une norme euclidienne, soit avec la somme des valeurs absolues, soit avec un maximum.

- Options du seuillage :

- *méthode de seuillage* : unique (seuil fixe), global (seuil calculé sur l'ensemble de l'image), local (seuil calculé localement sur une fenêtre) ou hystérésis (auto ou fixé) ;
- *méthode de calcul du seuil* : moyenne, médiane ou écart-type ;
- *valeur fixe*: changeable pour le seuillage unique ;
- *fenêtre de convolution* : changer la taille du noyau de convolution utilisé pour les seuillages local et hystérésis ;
- *seuil bas et haut* : Utiles pour l'hystérésis fixé. Dans le cas de d'hystérésis auto les seuils haut et bas sont fixés à partir de la moyenne et de l'écart-type.

- Option de fermeture :

- *taille de recherche* : défini le périmètre dans lequel une extrémité de contour va chercher à compléter le trait.
- *seuil de fermeture* : seuil au dessus duquel un pixel qui avait été éliminé par seuillage mais qui est détecté lors de la fermeture est considéré comme un contour à nouveau.

- Option générales :

- *la colorisation* : finaliser la détection de contours complète par une mise en couleur des contours selon la direction de leur gradient.
- *contours normalisés :* l'utilisateur peut choisir le fait que la couleur soit plus prononcée ou non (passage de tous les contours au dessus du seuil en blanc ou bien couleurs selon leur normes)

3/ Détection des contours

Pour la détection de contour nous nous sommes basé sur une approximation de la méthode de Canny. Nous effectuons donc au préalable un lissage gaussien, puis nous cherchons les contours, par défaut le filtre de Prewitt est utilisé en bidirectionnel. Ensuite les pixels sont filtrés pour éliminer les résultats non pertinents avec un seuillage par Hystérésis Auto et les contours restant sont affinés pour essayer de ne garder qu'un seul trait. A cela s'ajoute ensuite une fermeture pour tenter de combler les trous.

Explication des algorithmes :

Les lissages se font simplement avec un noyau de convolution de taille variable, que l'on applique à chaque pixel de notre image : noyau de moyenne, noyau gaussien ou noyau médian. Le filtre médian est calculé pour chaque pixel car il ne peut être fixé au préalable. De plus, pour la gestion des couleurs, notre filtre médian est marginal, c'est à dire que nous calculons la médiane pour chaque couleur séparément, sans se soucier de leur lien.

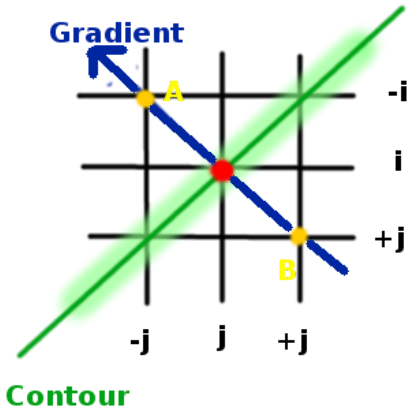
A partir de là, les traitements sont faits pour des images en nuances de gris et la plupart du temps nous ne testons les valeurs que d'un seul canal de couleur (les autres sont supposés identiques si l'image est bien en noir et blanc).
Au début du filtrage directionnel, l'image est passé en noir et blanc au cas où… si l'on ne passe pas par cette fonction d'abord ou si l'on applique pas manuellement un GrayScale sur l'image, les résultats ne seront pas probants !

Pour détecter les contours nous parcourons l'image et en chaque points on calcul le gradient en appliquant un noyau de convolution (ici un filtre de Prewitt).
Dans le cas bidirectionnel, 2 noyaux sont appliqués : un horizontal *Gx* et un vertical *Gy*. Ensuite nous calculs l'amplitude de ces gradients (par défaut avec une norme euclidienne $sqrt(Gx^2+Gy^2)$) et la direction est obtenu avec $arctan(Gy/Gx)$.
Dans le cas multi-directionnel, le filtre subi au préalable 8 rotations de 45° : les différents gradients sont calculés par convolution et la norme correspond au gradient maximum.

Nous avons différentes méthode de calculs pour le seuillage :
-*Unique* : Si une valeur est en dessous d'un seuil fixé, elle passe à 0, sinon c'est un contour.
-*Global* : Elle est identique au seuillage unique sauf que le seuil est calculé à partir soit de la moyenne des valeurs de l'image, soit de leur médiane, soit de leur écart-type.
-*Local* : Pour chaque pixel, on détermine le seuil en fonction de la moyenne/médiane/écart-type des pixels voisins dans une fenêtre donnée.
-*Hystérésis :* Si fixé : les seuils bas et haut sont fixés par l'utilisateur. Si Auto, nous calculons la moyenne et l'écart type de l'image : notre seuil bas correspond à la différence des 2, le seuil haut correspond à la somme des 2.
Ensuite pour chaque pixel si la norme est inférieure au seuil bas, on passe sa valeur à 0, si sa norme est supérieure au seuil haut alors ce pixel est considéré comme un contour.

Pour les normes entre les 2, nous regardons les voisins dans une fenêtre fixée par l'utilisateur : si il existe un pixel dont la norme est supérieure au seuil haut alors notre pixel actuel est un contour sinon on l'ignore et on le passe à 0.

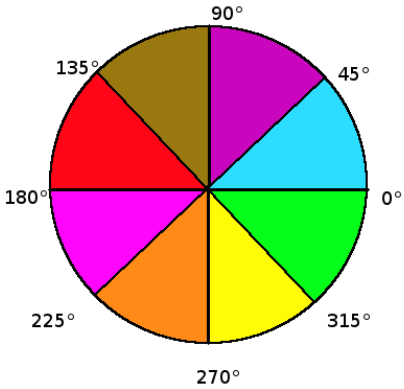
Pour affiner nos contours, nous parcourons l'image. Si on trouve un pixel étant un contour alors on regarde les pixels de part et d'autre de celui-ci, selon la direction de son gradient.
Si on trouve de nouveaux contours dans ces positions dont les valeurs sont inférieures à notre pixel actuel, alors on les supprime car on les considère comme étant un épaississement du contour, sinon si ces valeurs sont supérieures à notre pixels, on le passe à 0.



Affinage de contour : les points A et B sont enlevés

La fermeture se fait en 2 temps : d'abord, nous cherchons les extrémités de contours : pour chaque pixel détecté comme un contour, si celui ci ne possède qu'un seul voisin alors cela en fait une extrémité. Cette méthode se base sur le postulat que les contours ont été préalablement affiné et sont d'épaisseur 1.
Ensuite, pour chaque extrémitée, on regarde dans la direction perpendiculaire à son gradient (et opposée à son pixel voisin) : si l'on trouve un pixel à une distance inférieure à une taille fixée, dont la direction du gradient est identique au pixel actuel et qui a été supprimé par seuillage mais dont la norme est supérieure au seuil de fermeture : alors on ajoute tous les pixels entre celui-ci et le pixel actuel à la chaîne de contours. Ce nouveau contour devient notre nouvelle extrémité actuelle et on continue la recherche.

Pour finir, nous pouvons colorer nos contours en fonction de la direction et de la norme de leur gradient, en effet au préalable nous avons stocké une matrice contenant toute les normes de gradient en chaque point de l'image ainsi que les directions.
Pour chaque direction par palier de 45° nous attribuons une nouvelle couleur : cette couleur est atténuée / fonction de la norme si l'utilisateur le souhaite (cela donne des images moins marquées mais plus complètes).



Couleur du contour en fonction de la direction du gradient.

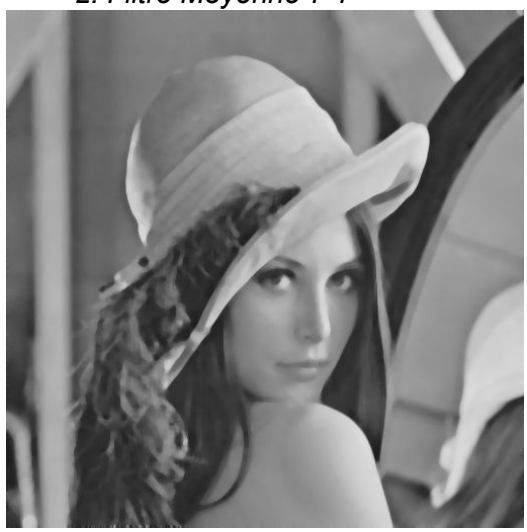
4/ Résultats

Lissage :

1. Image originale

*2. Filtre Moyenne 7*7*

*3. Filtre Gaussien 7*7 - = 5*



*4. Filtre Médian 7*7*

Filtrage bi-directionnel : (à partir de l'image lissée n° 3)

4. Prewitt (norme euclidienne)

5. Prewitt (norme absolue)

6. Sobel

7. Kirsch

Seuillage : (à partir de l'image filtrée n° 4)

8. Seuil Unique s=50

9. Seuil Gobal - Ecart-type

10. Hysteresis s=44 - S=60

11. Hysteresis Auto

Affinage : (à partir de l'image seuillée n° 11)

11. Non affinée

12. Affinage

5/ Conclusion

Pour ce TP nous avons donc implémenté toutes les fonctionnalités principales demandées, de manière totalement paramétrable par l'utilisateur, et nous permet d'avoir une base solide pour débuter le prochain TP durant lequel nous pourrons ajouter les fonctionnalités permettant la détection de formes simples.

Cependant, quelques remarques peuvent être faites sur la pertinence de nos résultats ou sur nos temps d'exécution :

- Lors de l'affinage, il arrive que des contours 'double' apparaissent. De plus notre élimination du bruit reste limitée nous pourrions éliminer des petits amas de pixels se trouvant loin d'autres contours par exemple.

- Notre fermeture n'est pas très fonctionnelle. En effet, la poursuite de contour se fait régulièrement dans de mauvaises directions.

- Notre programme n'est pas particulièrement rapide, surtout lors d'un traitement complet de détection de contours sur de grandes images. D'abord, nos calculs ne sont pas parallélisés, nous n'utilisons donc pas la puissance du multi-coeur que la plupart des ordinateurs possèdent. Ensuite, pour les besoins de l'interface, chaque traitement copie d'abord l'image en entrée pour renvoyer une 2e image modifiée. Cela génère beaucoup de parcours / copies en plus.

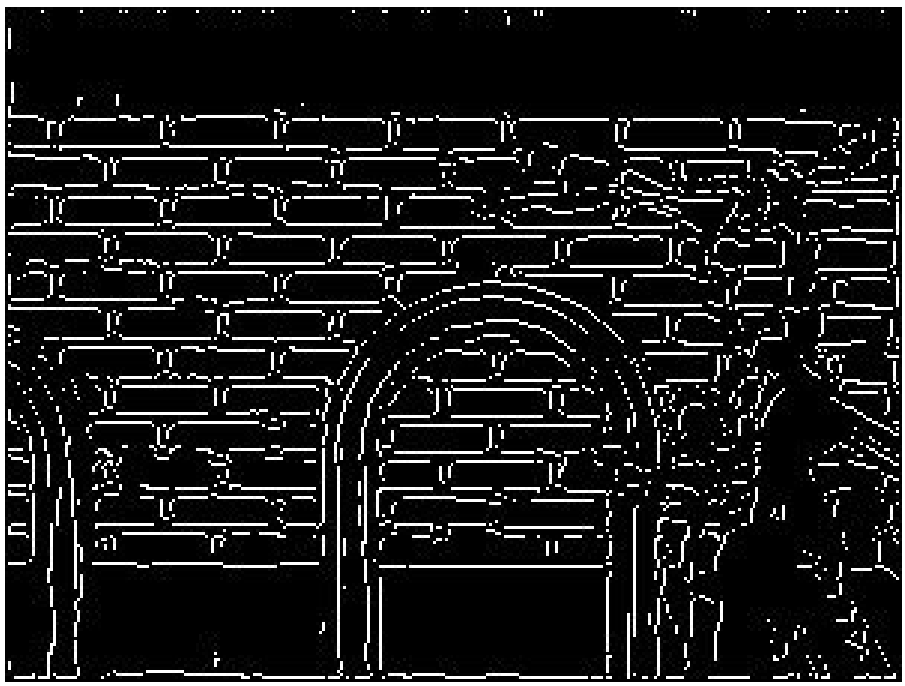
- La gestion des bords aurait également pu être améliorée : nous ne nous sommes pas du tout penché la dessus et à chaque convolution sur l'image, nous décalons le départ et la fin du parcours en fonctions de la taille du noyau ce qui fait que les bords ne sont pas traités.

Annexes

Interface Graphique

1. Ouvrir une image (jpg / png)
2. Enregistrer l'image de droite (-14-)
3. Undo : Annule la dernière action
4. Redo : Refait une action annulée
5. Reset : Supprime tous les traitements appliqués à l'image de base.
6. Apply : Remplace l'image de base à gauche -13- par l'image modifiée de droite -14-.
7. Zoom In/Out/Fit
8. GrayScale (3 types possibles) - Inverser les couleurs - Rotation de l'image.
9. Étape par étape de la détection de contour : les options pour chaque étapes se paramètre dans l'onglet du même nom dans le panel -12-.
10. Détection de contour qui applique automatiquement toutes les étapes précédentes avec les options définies dans le panel -12-.
11. Faire varier l'intensité des couleurs / Nuance de Gris des résultats.
12. Panel des Options. Voir la partie 2/ b. de ce rapport pour les détails.
13. Image en entrée (voir outil -1-)
14. Image de sortie : on peut l'enregistrer ou la définir comme image d'entrée.
15. Informations sur l'image
16. Console de Log : Historique de chaque traitement et de ses options appliqué à l'image.

*LIS. GAUSSIEN 7*7 = 5 ; PREWITT bi-directionnel - norme euclidienne ; HYSTERESIS AUTO 15*15*



*LIS. MEDIANE 5*5 ; PREWITT bi-directionnel - norme euclidienne ; seuillage Global par Moyenne*

*LIS. GAUSSIEN 7*7 = 5 ; PREWITT bi-directionnel - norme euclidienne ; HYSTERESIS AUTO 15*15*