



ENSSAT
L A N N I O N

PROJET NOTÉ

Détection des contours d'une image : le filtre de Canny

[colinleverger \[at\] gmail \[dot\] com](mailto:colinleverger[at]gmail[dot]com)

Colin LEVERGER - ENSSAT Informatique, Multimédia et Réseaux
Promotion 2017

Destinataire : [Benoit VOZEL](#)

20 avril 2016

1 Introduction

Dans le cadre de la formation Informatique, Multimédia et Réseaux dispensé à l'ENSSAT de Lannion, nous avons étudié le traitement d'image en seconde année. Cette matière, qui s'inscrit dans la composante «multimédia» de notre cursus, nous a permis d'appréhender les concepts clés de la manipulation d'images par programmation. Nous avons notamment travaillé à la création de filtres gaussiens, à la création de matrices de gradient, etc.

Pour pratiquer et appréhender au mieux les concepts évoqués en CM, nous avons travaillé sur un TP noté. Il s'agissait d'écrire avec Scilab un programme permettant de trouver et d'isoler les contours des objets présents sur une image, de manière automatique. J'ai choisi d'utiliser un grand classique du traitement d'image, à savoir Lena, qui n'est autre qu'une pinup des années 70 qui se trouvait au bon endroit au bon moment (à savoir, dans les mains d'un chercheur dans le domaine de l'imagerie numérique lorsqu'il cherchait une image pour ses traitements).

Ce compte rendu présente la démarche lors de la réalisation et du codage de ce TP. La structure du sujet sera suivie, car nous allons expliquer les méthodes et fonctions dans l'ordre chronologique de codage. Les fonctions et méthodes codées seront en effet très souvent réutilisées pour la suite du TP.

Le code associé à ce compte rendu pourra être trouvé dans l'archive jointe à ce rapport. Très peu de code sera directement exposé dans le rapport, mais les structures complexes et les choix d'implémentation des algorithmes y seront clairement explicités.

2 Préliminaires

Afin de travailler avec Scilab, quelques étapes préliminaires ont été effectuées.

Tout d'abord, le choix de l'image était important pour pouvoir développer rapidement les premières fonctions. L'exécution des algorithmes prend un temps non négligeable, et il était important de prendre une petite image pour le début des opérations : pas plus de 64x64 pixels. Une image de 64x64 pixels représente par exemple une matrice de 64 par 64. Évidemment, plus l'image est grande (plus le nombre de pixels est important), plus la taille de la matrice à traiter est grande, et plus le traitement prend du temps.

La seconde chose à faire avant de se lancer dans le codage était l'installation du module Scilab nécessaire pour utiliser les fonctions basiques de chargement d'image. Il s'agit en effet de pouvoir transformer une image en entrée en matrice de valeurs, situées entre 0 et 255 (niveaux de gris). L'installation du module dans Scilab sur Mac OS est loin d'être triviale... L'utilisation du logiciel sous Windows est préférée.

3 Première étape : le rehaussement de Canny

La première étape de l'algorithme peut se décomposer en deux sous parties. Il s'agit tout d'abord de lisser l'image, afin d'en retirer les impuretés, et de calculer dans un second temps la norme du gradient et l'angle de la normale au gradient pour chaque pixel de l'image lissée.

3.1 Le filtrage de l'image

La présence d'impuretés sur une image est courante, et même très probable. Un exemple d'impuretés pourra être vu en figure 1 en page 2 [Phi]. Dans ce cas précis, il d'agit du bruit blanc gaussien, qui provoque une altération nettement visible de l'image.

On sent bien que la détection de contours demandée pourrait potentiellement être faussée par le bruit présent sur l'image : il est donc important de filtrer l'image, afin de supprimer le bruit et de pouvoir effectuer les traitements en limitant le biais.



FIGURE 1 – Le bruit blanc gaussien sur l'image Lena

Nous allons donc «lisser» l'image, en lui appliquant un filtre gaussien. Le filtre est lui même une matrice, et il faut l'appliquer en effectuant des produits de convolution entre l'image et le filtre. Typiquement, une boucle va parcourir toutes les valeurs de la matrice image de base et effectuer un traitement pour chacune de ces valeurs. Un schéma de l'application de ce filtre pourra être trouvé en figure 2 en page 2 [VOZ16].

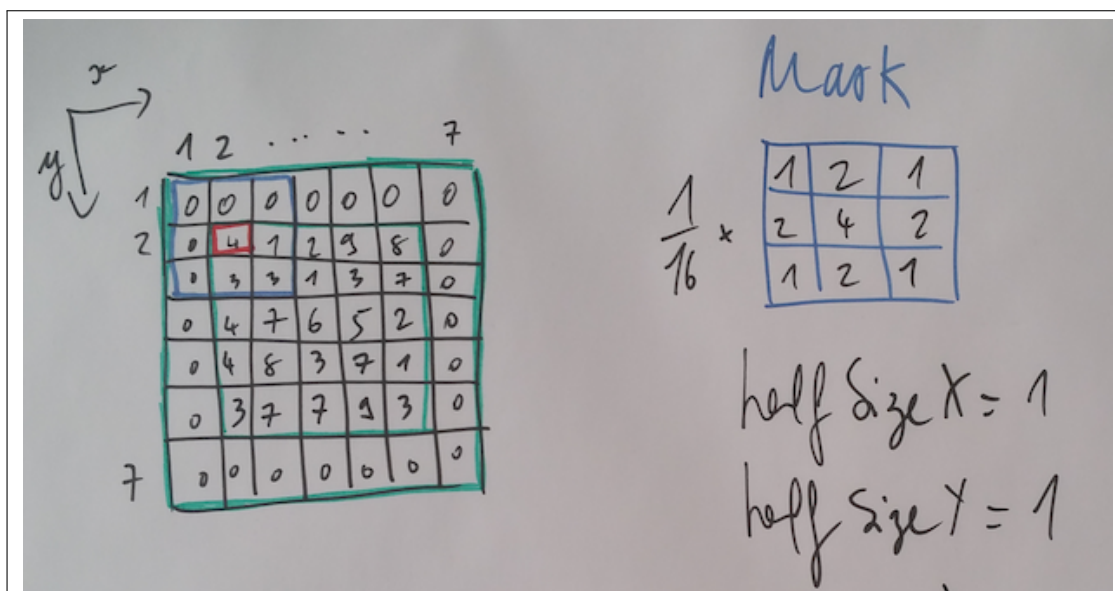


FIGURE 2 – Application du filtre par produit de convolution

Concernant l'application du filtre, le principe est le suivant :

1. Il faut d'abord superposer le filtre et l'image,

2. Multiplier par les facteurs présents dans la matrice de filtrage chaque pixel de l'image par superposition,
3. Sommer les résultats de chaque multiplication,
4. Normaliser le résultat,
5. Affecter cette somme normalisée au pixel considéré.

Une nouvelle matrice de taille égale à l'image sera initialisée et remplie au fil du traitement.

Le traitement des valeurs au bord de l'image (les pixels qui délimitent l'image) est une problématique qui se pose : en effet, positionner le masque et faire un produit de convolution sur des valeurs non existantes est impossible ! Pour pallier ce problème, plusieurs méthodes plus ou moins complexes existent, par exemple :

- Augmenter l'image en créant des bords artificiels à valeur égale à 0.
- Faire une réflexion et utiliser les bords opposés quand nécessaires.

La solution choisie lors de ce TP a été d'augmenter la taille de l'image et de remplir les valeurs inexistantes de 0. Il s'agit alors d'augmenter l'image d'une demi-taille de filtre de chaque côté (si le filtre fait une taille de 3x3, l'augmenter de 1 de chaque côté, s'il fait 6x6 de 2 de chaque côté, etc.) Notez que cette méthode peut ajouter un effet de bord, les valeurs ajoutées «0» étant en effet potentiellement fausses... Pour un exemple d'agrandissement de la matrice, voir figure 3 en page 3.

1.	2.	3.	4.	5.		
1.	2.	3.	4.	5.		
1.	2.	3.	4.	5.		
1.	2.	3.	4.	5.		
1.	2.	3.	4.	5.		
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	1.	2.	3.	4.	5.	0.
0.	1.	2.	3.	4.	5.	0.
0.	1.	2.	3.	4.	5.	0.
0.	1.	2.	3.	4.	5.	0.
0.	1.	2.	3.	4.	5.	0.
0.	0.	0.	0.	0.	0.	0.

FIGURE 3 – Agrandissement de l'image pour un filtre de taille 3x3. Première matrice : matrice de base, deuxième matrice : matrice temporaire faite de 0, 3e matrice : inclusion de la matrice de base dans la matrice temporaire.

Pour appliquer le filtre, il faudra donc commencer le traitement à l'indice ($X_{tailleDuDemiFiltre}$, $Y_{tailleDuDemiFiltre}$) de la matrice image.

Les étapes suivies si on suit la figure 2 seront donc (application pratique de l'algorithme) :

1. Placer le pixel rouge au début de notre image précédemment agrandie.

2. Faire les convolutions en faisant la somme de la multiplication par superposition les valeurs des deux matrices (ici, on va avoir $0 * 1 + 0 * 2 + \dots + 4 * 4 + 1 * 2 + \dots + 3 * 1$)
3. Normaliser le résultat en le multipliant par l'inverse de la somme de toutes ses valeurs (ici, $1 / [1 + 2 + 1 + \dots + 1] = 1 / 16$)
4. Affecter la nouvelle valeur au pixel rouge dans un nouveau tableau.
5. Déplacer le pixel rouge sur l'image¹ en suivant les axes x et y pour traiter chacun des pixels.

La taille filtre que nous appliquons sur notre image peut être plus ou moins grande. Évidemment, plus elle est grande plus le traitement est coûteux et plus il prend du temps. On note aussi que plus la taille du filtre gaussien est importante, plus l'image est floutée par le traitement. Deux filtres ont été utilisés pendant les manipulations (voir figure 4 en page 4). Le premier filtre va lisser l'image convenablement, mais moins efficacement que le second.

On note la présence de coefficients devant les filtres : ceux-ci représentent la normalisation des valeurs. On va en effet diviser chaque pixel par la somme totale des éléments présents dans le filtre.

1	2	1	x 1/16		
2	4	2			
1	2	1			
2	4	5	4	2	x 1/159
4	9	12	9	4	
5	12	15	12	5	
4	9	12	9	4	
2	4	5	4	2	

FIGURE 4 – Les filtres utilisés pour expérimenter

La fonction d'application du filtre sera utilisée plusieurs fois le long de ce TP : c'est pour ça qu'elle a été codée de la façon la plus générique possible.

Un exemple d'image lissée pourra être trouvée en 5 en page 5.

3.2 Calcul de normes

Afin d'effectuer des comparaisons entre pixels voisins, il faut désormais effectuer des calculs mathématiques, concernant la norme et la normale au gradient de chaque pixel de l'image.

Les formules à appliquer pour chaque pixel sont les suivantes :

$$e_s(i, j) = \sqrt{J_x^2(i, j) + J_y^2(i, j)}$$

$$e_o(i, j) = \arctan\left(\frac{-J_x}{J_y}\right)$$

Avec ici J_x qui représente les gradients en x et J_y les gradients en y. Pour calculer J_x et J_y , il s'agit simplement de réutiliser notre fonction d'application d'un masque codée précédemment, avec les matrices du gradient (voir figure 6 en page 5).

1. L'image considérée sera toujours l'image de départ.



FIGURE 5 – Lenna avant et après lissage gaussien

Une fois les masques appliqués et les matrices J_x et J_y prête à être exploitées, il suffit simplement d'appliquer les formules mathématiques pour obtenir les valeurs pour chaque pixel (en utilisant comme d'habitude une double boucle pour parcourir toutes les valeurs de la matrice image). La fonction qui effectue ce traitement ("gradientNorm" dans le code) retourne donc deux valeurs, e_s et e_o .

En x :	1	0	-1
En y :	1		
	0		
	-1		

FIGURE 6 – Masques à appliquer pour obtenir les gradients

Une subtilité pourra être soulignée : e_o à en fait été normalisée. Le but est de renvoyer un angle indiquant la direction approximée du gradient, afin de pouvoir trouver plus tard les voisins des pixels du contour (s'il existe). Pour approximer, nous avons suivi le schéma 7 en page 6. Typiquement, si une valeur est située dans la zone dessinée en rouge (une valeur telle que 12 degrés par exemple) on pourra l'approximer à 0 degrés pour simplifier.

Si une valeur est plus grande que $135 + 22.5 = 157.5$ degrés, nous avons choisi de lui enlever 180 degrés afin de la retenir dans cette fonction de normalisation (récursivité). Même principe, si un angle est inférieur à -22.5 , on lui ajoute 180 degrés et on recommence le traitement.

4 Deuxième étape : Suppression des non-maximums

La suppression des non maximums est une étape assez rapide. Il s'agit de supprimer de la norme du gradient toutes les valeurs "faibles", les valeurs qui ne sont pas les plus grandes en suivant l'angle de leur gradient.

Les étapes de cette suppression pour chaque pixel sont les suivantes :

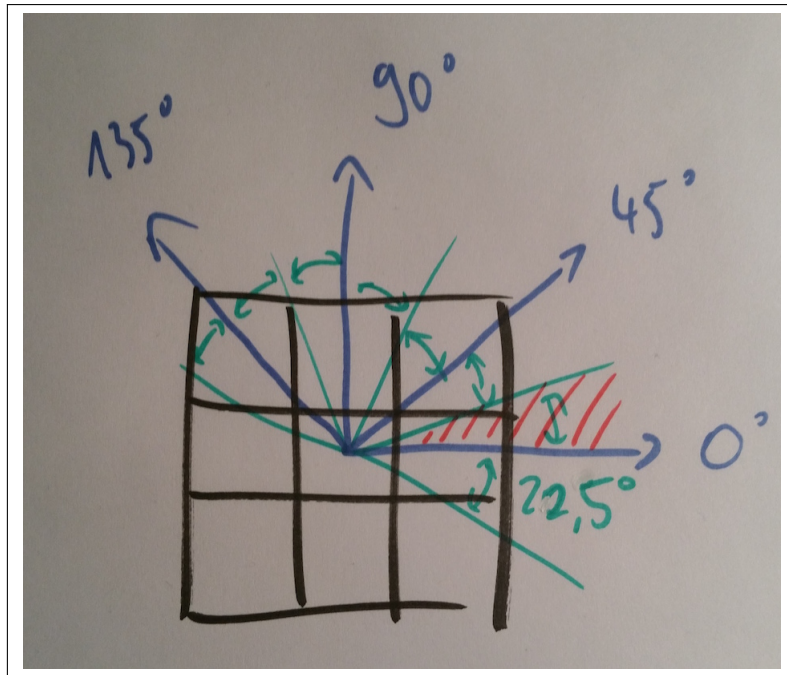


FIGURE 7 – Méthode de normalisation de l'angle du gradient

1. Trouver les voisins du pixel en suivant l'angle du gradient. Exemple : si l'angle est de 90 degrés, les pixels voisins seront le pixel juste au dessus et juste en dessous du pixel courant. Ces pixels auront les coordonnées $(x, y-1)$ et $(x, y+1)$.
2. Vérifier la valeur des voisins : si la valeur de l'un des voisins est plus grande que la valeur du pixel courant, supprimer le pixel courant (lui affecter la valeur 0), sinon le garder à sa valeur d'origine.

On note que la problématique du traitement des bords est aussi à considérer lors de cette étape. Ici, le choix à été fait de développer une fonction générique pour récupérer les valeurs d'un tableau à partir d'indices donnés, si elles existent. Si elles n'existent pas, la fonction retourne simplement une valeur égale à 0. La fonctions codées pour ce faire s'appellera "*getMatValueIfExists*".

Afin de rendre générique la récupération des voisins d'un pixels, une fonction à également été codée ; cette fonction sera par ailleurs réutilisée plusieurs fois le long du TP. Cette fonction a pour nom "*getNeighborhoodCoords*", et retournera les coordonnées des voisins théorique du pixel donné en parametre.

5 Troisième étape : seuillage par hystérésis

6 Dernière étape : trouver le seuil automatiquement

7 Conclusion

7.1 Tests & Benchmarks

L'algorithme a été testé avec une image 6000x6000 afin de le benchmarquer. L'hypothèse émise était que cet algorithme est linéaire ; si traiter une image de 60x60 prend 20 secondes, traiter une image de 6000x6000 prendrait simplement 100 fois plus de temps, soit 2000 secondes = 33,3 minutes. Malheureusement, le test n'a pas pu se terminer : erreur de capacité pour l'ordinateur...

Table des figures

1	Le bruit blanc gaussien sur l'image Lena	2
2	Application du filtre par produit de convolution	2
3	Agrandissement de l'image pour un filtre de taille 3x3. Première matrice : matrice de base, deuxième matrice : matrice temporaire faite de 0, 3e matrice : inclusion de la matrice de base dans la matrice temporaire.	3
4	Les filtres utilisés pour expérimenter	4
5	Lenna avant et après lissage gaussien	5
6	Masques à appliquer pour obtenir les gradients	5
7	Méthode de normalisation de l'angle du gradient	6

Références

- [Phi] PHILIPPEAU, Xavier : *Filtre Mean Shift*. <http://xphilipp.developpez.com/articles/meanshift/>
- [VOZ16] VOZEL, Benoit : *CM traitement image*, 2016