Taitement d'image
Segmentation d'image
Méthode de la segmentation par snakes

Axel Bruyere, Dorian Fabregue
4ETI-IMI

26 mars 2021

Objectifs :
- Implémenter un algorithme de segmentation par snakes sur des images simples
- Comprendre l'influence des paramètres d'énergies interne et externe
- Choix des paramètres optimaux pour les images *goutte* et *verre* (im10)

1 Contexte

Nous nous intéressons dans ce TP à la méthode de segmatation par snakes sur des éléments simples de différentes images.

Il s'agit d'une méthode de segmentation par contours actifs, basée sur des modèles déformables, par opposition à des méthodes qui visent à segmenter des régions à partir de discontinuités ou de similarités des niveaux d'intensité. Le terme actif désigne le fait que la segmentation se fait de façon dynamique, via un processus itératif, par opposition à une segmentation résultant d'un simple seuillage. Les contours actifs convergent en effet progressivement vers les contours des régions à segmenter, sous l'effet d'une "force" définie par les paramètres de l'algorithme.

La méthode "snakes" nécessite une modélisation paramétrique explicite du contour. Les paramètres du contour sont obtenus par la minimisation d'une fonctionnelle d'énergie composée de deux termes :
- une énergie interne liée à la forme intrinsèque du contour, qui vise à attirer le contour vers les bords des objets
- une énergie externe liée au contenu de l'image

2 Algorithme de *snakes*

Nous allons, dans un premier temps, implémenter cet algorithme sur l'image *im_ goutte.png* contenant un objet de forme elliptique ne présentant pas de discontinuité et sur lequel l'algorithme devrait fonctionner facilement.

Pour commencer, nous récupérons notre image et lui appliquons un flou gaussien grâce aux fonctions *imread* et *GaussianBlur* de la librairie *OpenCV* avec les lignes suivantes :

```
im = cv2.imread('imagesTP/im10.png', 0)
image = cv2.GaussianBlur(im,(5,5),cv2.BORDER_DEFAULT)
```

2.1 Courbe paramétrique

L'initialisation du *snakes* passe par la mise en place d'une courbe paramétrique faisaint le contour de l'image. Une courbe paramétrique du plan (x, y) se définit par les coordonnées suivantes :

$$(x, y) = (g(s), h(s))$$

Les équations $x = g(s)$ et $y = h(s)$ ne dépendent que d'un seul paramètre s, et sont appelées équations paramétriques de la courbe. Dans le cas d'un cercle centré à l'origine d'un plan (x, y) et d'équation $x^2 + y^2 = r^2$, on peut vérifier que :

$$x = r.cos(2\pi s) \text{ et } y = r.sin(2\pi s) \qquad \text{pour } s \in [0; 1]$$

L'utilisation d'un seul paramètre pour décrire la même courbe permet de réduire la complexité de la représentation, et ainsi de pouvoir l'implémenter bien plus facilement. Notre *snake* sera représenté par la courbe paramétrique c, de paramètre $s \in [0, 1]$:

$$c(s) = \begin{bmatrix} x(s) \\ y(s) \end{bmatrix} \tag{1}$$

On considérera des courbes fermées pour lesquelles $c(0) = c(1)$. On peut donc le coder de la manière suivante :

```
#Initialisation de notre snake
x = []
y = []
L_snake=1000
for i in range(L_snake):
    x.append(len(image[0])/2 + len(image)/2.5 * math.cos(i*2*np.pi/L_snake))
    y.append(len(image)/2 + len(image)/2.5 * math.sin(i*2*np.pi/L_snake))
```

On obtient donc l'image suivante :
Afin que le cercle soit bien configuré sur l'image, nous avons :

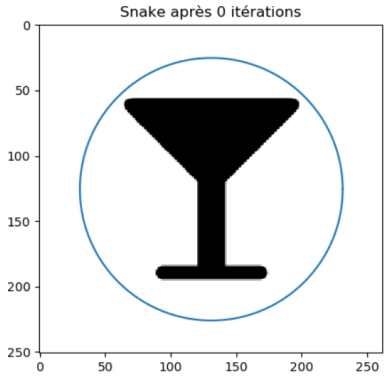

FIGURE 1: Snake initial

- placé son centre au centre de l'image tel que $centre = (\frac{largeur_{image}}{2}, \frac{hauteur_{image}}{2})$
- choisi un rayon $r = \frac{hauteur_{image}}{2.5}$ afin que celui ci soit totalement inclu dans l'image et soit suffisemment grand pour contenir l'objet à ségmenter
- pris un nombre de points $500 \leq L\_snake \leq 1000$ afin que la courbe c épouse au mieux les contours de notre objet, tout en veillant à garder un temps d'exécution du programme acceptable.

2.2 Energie du snake

On associe à ce *snake* une énergie $E(c)$ composée de 2 termes :

$$E(c) = E_{interne}(c) + E_{externe}(c) \tag{2}$$

Le terme d'énergie interne est lié au snake lui-même, alors que le terme d'énergie externe est lié à l'image que l'on souhaite segmenter. On cherche ici la courbe c qui minimise E, c'est à dire à la fois les énergies interne et externe.

Nous savons que les ces énergies peuvent s'écrire de la manière suivante :

$$E_{interne}(c) = E_{elastique}(c(s)) \quad + E_{courbure}(c(s)) \tag{3}$$

$$= \frac{1}{2}\int_0^1 \alpha||\frac{\partial c(s)}{\partial s}||^2 ds + \frac{1}{2}\int_0^1 \alpha||\frac{\partial^2 c(s)}{\partial s^2}||^2 ds \tag{4}$$

avec α et β deux réels positifs
Notons que cette énergie interne est complètement indépendante du contenu de l'image. Elle permet seulement de contraindre la forme du snake.

$$E_{externe}(c) = \int_0^1 E_{image}(c(s))ds = -\int_0^1 \gamma||\nabla I(c(s))||^2 ds \tag{5}$$

avec γ réel positif .

Les termes de pondération (α, β, γ) nous permettent d'influer respectivement sur $E_{elastique}$, $E_{courbure}$ et $E_{externe}$. Plus ces valeurs seront grandes, plus nous minimiseront l'importance de l'énergie en question. Nous verrons plus en détail comment se visualisent ces énergies lorsque nous aurons implémenté l'algorithme de segmentation.

2.3 Implémentation de l'algorithme

Après avoir implémenté la courbe paramétrique, il s'agit alors d'implémenter l'algorithme de la segmentation par snakes. Pour cela, nous avons besoin de créer notre matrice A telle que :

$$A = [Id - D]^(-1) \tag{6}$$

Avec :

$$D = \alpha * D2 + \beta * D4 \tag{7}$$

D2 et D4 étant des matrices parcimonieuses correpondant respectivement aux opérateurs matriciels de dérivée seconde et quatrième. De plus, nous avons aussi défini les paramètres α, β et γ.

Voici le code de ces initialisations :

```
# Initialisation des paramètres
alpha = 1.5
beta = 0.5
gamma = 0.001

#Initialisation de la matrice D2
D2 = sp.diags([1,1, -2, 1,1],[-(L_snake-1),-1, 0, 1,L_snake-1],shape=
(L_snake,L_snake)).toarray()

#Initialisation de la matrice D4
D4 = sp.diags([-4,1,1,-4,6,-4,1,1,-4],[-(L_snake-1),-(L_snake-2),-2,-1,0,1,2,
L_snake-2,L_snake-1],shape = (L_snake,L_snake)).toarray()

#Calcul de la matrice A
D = alpha * D2 - beta * D4
A = np.linalg.inv(np.eye(L_snake)-D)
```

Ensuite, nous initialisons différents gradients -qui vont nous être utiles lors de l'itération sur x et y- avec le code suivant :

```
#initialisation des différents gradients
grad_norm_x,grad_norm_y = np.gradient(grad_thresh)
grad_int_x = np.zeros(len(x))
grad_int_y = np.zeros(len(y))
```

Au préalable, nous avions passé l'image de notre gradient en binaire à l'aide de la méthode *threshold()* de la bibliothèque *OpenCV*.

Une fois toutes ces variables initialisées, nous pouvons démarrer l'itération sur x et y afin de rétracter notre snake autour de l'image. Cette itération s'appuie sur l'expression suivante :

$$x(t) = A.[x(t-1) + \gamma.\nabla_x[\|\nabla I(x(t-1), y(t-1)\|^2]] \tag{8}$$

Nous procédons alors à l'aide de deux boucles *for* imbriquées. L'une ayant pour rôle de permettre le calcul du nouveau gradient, l'autre visant à mettre à jour les valeurs de x et y. Dans le même temps, nous traçons le nouveau snake et effaçons l'ancien. Tout ceci est assuré par les lignes suivantes :

```
for i in range (30000):
    for p in range (L_snake):
#Mise à jour du gradient intermédiaire
        grad_int_x[p]=grad_norm_x[int(y[p])][int(x[p])]
        grad_int_y[p]=grad_norm_y[int(y[p])][int(x[p])]
#Mise à jour de x et y
    x = np.dot(A, x + gamma * grad_int_x)
    y = np.dot(A, y + gamma * grad_int_y)

#On efface les anciens x,y et on trace les nouveaux
    if i%2000 == 0:
        plt.clf()
        plt.imshow(image,'gray')
        plt.plot(x,y)
        plt.title("Snake après %d itérations" %i)
        plt.pause(0.01)
```

Ce code nous permet une certaine généricité. En effet, les paramètres α, β et γ sont aisément modifiables afin de procéder à différents tests d'optimisation de notre programme en rapport avec l'image à traiter.

2.4 Influence des paramètres

Comme nous l'avon s évoqué plus tôt, l'algorithme de segmentation par snakes propose de jouer sur trois paramètres : α, β et γ.

2.4.1 Influence de α

Ce paramètre permet de minimiser l'influence de l'énergie élastique de notre snake. Plus il est grand, plus cette énergie tend à se réduire. En d'autres termes, augmenter α signifie faire converger notre snake plus rapidement. Pour améliorer la vitesse d'exécution de notre programme, il est alors nécéssaire de choisir un α suffisamment grand. En revanche, s'il est trop grand, notre snake risque de "manquer" le bord de l'image, et ainsi continuer à converger après y être rentré. Sur la figure 2 ci-dessous, $\alpha = 1$. La convergence est relativement rapide et s'arrête localement lorsque la snake touche un bord.

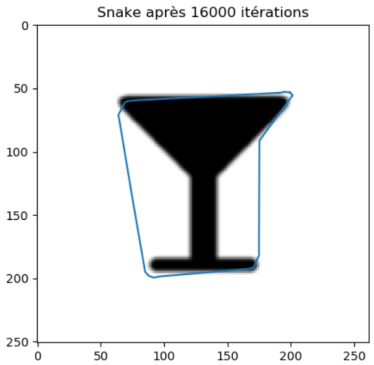

FIGURE 2: Snake avec un α relativement faible

Sur la figure 3 ci-dessous, nous avons initialisé α à 3. Cela implique un convergence bien trop rapide, et ainsi mauvaise, puisque notre snake "manque" les bords de l'image.

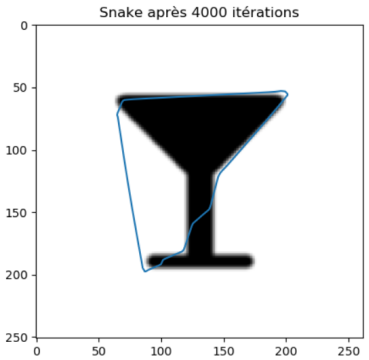

FIGURE 3: Snake avec un α trop grand

2.4.2 Influence de β

Le paramètre β a pour but de minimiser l'influence de l'énergie de courbure. En d'autres termes, lorsque β est grand, se voit diminuée, et le snake épousera moins le contour que l'on recherche. En effet, celui-ci aura une plus grande appétence à privilégier une "ligne droite" afin de minimiser l'énergie de courbure. Choisir un β trop grand implique donc un détourage plus grossier. A l'inverse, choisir un β trop faible impliquera une énergie de courbure pouvant être plus grande, au risque de voir alors le snake prenant des virages inutiles, notamment pendant l'itération, ce qui augmente le temps d'exécution.
Malheureusement, notre code ne nous permet pas de considérer en pratique l'influence de β.

2.4.3 Influence de γ

Le paramètre γ permet de minimiser l'influence de l'énegie externe du snake, qui elle est intrinsèquement liée à l'image que l'on souhaite segmenter. En effet, celle-ci permet de "repousser" le snake lorsqu'il arrive au niveau d'un contour. Ce paramètre, s'il est mal choisi, peut notamment amener le snake à "manquer" un contour, en ne contrebalançant pas suffisamment sa convergence (voir la figure 3, le phénomène se traduisant de la même manière). A l'inverse, lorsqu'il est choisi trop grand, il repousse le snake de façon à ce qu'il ne se stabilise jamais autour de l'image, comme sur la figure 4 ci-dessous. Pire encore, s'il est vraiment trop grand, il peut repousser le snake au point de le faire sortir de l'image, amenant de cette façon une erreur sur lors de l'itération suivante.

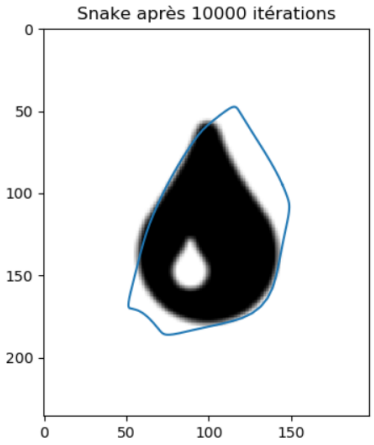

FIGURE 4: Snake avec un γ trop grand

2.4.4 Influence du rayon et du nombre de points du snake initial

A l'instar des paramètres α β et γ, le rayon du snake initial ainsi que le nombre de points qui le composent ont également un impact sur le bon déroulement de la segmentation. En effet, il est important de ne pas choisir un nombre de points trop faible car ceci rend le snake très imprécis, l'amenant à "manquer" les contours. S'il est choisit trop grand, alors le temps d'exécution est fortement impacté. Pour notre part, nous avons opté pour un nombre de points $500 \leq L\_snake \leq 1000$ dans le but de trouver un bon compromis.
Le rayon du snake initial a également son importance. En effet, il paraît évident qu'il doit être suffisamment grand pour entourer l'image à segmenter entière. Par ailleurs, un snake trop grand rallonge inutilement le temps d'exécution, puisque les premières étapes de la convergences se feront "dans le vide", le temps que le snake s'approche de l'image. A l'excès, si le snake n'est pas entièrement contenu dans l'image, nous obtiendrons une erreur dès la première itération.

3 Limites du programme

Le programme se segmentation par snakes est assez efficace, cependant il présente des limites. La première et la plus évidente est le temps d'exécution. Le bon choix des différents paramètres est alors crucial pour le réduire au maximum. De plus, il est très performant sur des formes bien définies, mais il l'est moins sur des formes présentant beaucoup d'aspérités. Par ailleurs, il demande une certaine qualité d'image, puisque des contours flous de trouvent bien plus difficiles à trouver.
Pour notre part, nous l'avons trouvé difficile à implémenter. Il en résulte un fonctionnement qui n'est pas satisfaisant, mais nous n'avons pas sur trouver l'origine du problème.

4 Annexe : code complet

```
import numpy as np
from matplotlib import pyplot as plt
from scipy import sparse as sp
import math
import cv2


#Récupération de l'image en nuances de gris
image = cv2.imread('imagesTP/im10.png', 0)
#image = cv2.GaussianBlur(im,(3,3),cv2.BORDER_DEFAULT)

#Calcul du carré de la norme du gradient de l'image et passage en image binaire

gradx,grady = np.gradient(image)
grad_norm = np.square(gradx)+np.square(grady)
ret,grad_thresh = cv2.threshold(grad_norm,10,245,cv2.THRESH_BINARY)

#Initialisation des paramètres
alpha = 1.5
beta = 0.5
gamma = 0.01
L_snake = 1000

#Initialisation de notre snake
x = []
y = []

for i in range(L_snake):

    x.append(len(image[0])/2 + len(image)/2.5 * math.cos(i*2*np.pi/L_snake))
    y.append(len(image)/2 + len(image)/2.5 * math.sin(i*2*np.pi/L_snake))

#Initialisation de la matrice D2
D2 = sp.diags([1,1, -2, 1,1], [-(L_snake-1),-1, 0, 1,L_snake-1], shape=(L_snake,L_snake)).toarr

#Initialisation de la matrice D4
D4 = sp.diags([-4,1,1,-4,6,-4,1,1,-4],[-(L_snake-1),-(L_snake-2),-2,-1,0,1,2,L_snake-2,L_snake-

#Calcul de la matrice A
D = alpha * D2 - beta * D4
A = np.linalg.inv(np.eye(L_snake)-D)


#initialisation des différents gradients
grad_norm_x,grad_norm_y = np.gradient(grad_thresh)
grad_int_x = np.zeros(len(x))
grad_int_y = np.zeros(len(y))
```

```
plt.ion()

for i in range (30000):
    for p in range (L_snake):
#Mise à jour du gradient intermédiaire
        grad_int_x[p]=grad_norm_x[int(y[p])][int(x[p])]
        grad_int_y[p]=grad_norm_y[int(y[p])][int(x[p])]
#Mise à jour de x et y
    x = np.dot(A, x + gamma * grad_int_x)
    y = np.dot(A, y + gamma * grad_int_y)

#On efface les anciens x,y et on trace les nouveaux
    if i%2000 == 0:
        plt.clf()
        plt.imshow(image,'gray')
        plt.plot(x,y)
        plt.title("Snake après %d itérations" %i)
        plt.pause(0.01)

plt.show()
```