CPE LYON

Année universitaire

2021 - 2022

4ETI — Traitement d'images

TP de contours actifs (Snakes)

Marion Foare

Eric Van Reeth

Objectif. Ce TP de 4 heures consiste à implémenter une segmentation par l'algorithme de *snakes* vu en cours sur des images simples. Á partir d'un contour initial, la segmentation sera actualisée itérativement à partir du schéma itératif vu en cours.

Déroulement. Ce TP s'effectue en binôme, en Python et en utilisant de préférence l'IDE VSCode. Vous trouverez avec le TP un dossier contenant les images sur lesquelles vous travaillerez, à savoir :

- l'image *im_goutte.png*, que tous les groupes devront segmenter dans un premier temps
- les autres images (numérotées), qui seront attribuées de façon individuelle à chaque groupe

Évaluation. Dans un délai précisé par l'encadrant, vous déposerez sur le dépôt ouvert un **compte-rendu au format pdf**, ainsi que le **code python** (fonctionnel) implémenté. Le tout sera compressé dans une archive (.zip) contenant les noms des membres du binômes.
Le compte-rendu contiendra une description rigoureuse des méthodes implémentées, une discussion sur le(s) critère(s) d'arrêt choisi(s) et les résultats obtenus.
Le code sera commenté et séparé en parties clairement identifiées permettant au correcteur de comprendre et de reproduire tous vos résultats (les codes rendus seront comparés, et les éventuels doublons seront sanctionnées pour les deux groupes).

Mise en place de l'environnement Python

Veillez à ce que l'**interpréteur Python** utilisé dans votre IDE soit le suivant :
`/opt/python/cpe/bin/python3`

Pour ce TP, vous aurez besoin des librairies suivantes :

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

Une aide des fonctions Python peut être obtenue via la commande `help(functionName)`.
Les commandes openCV nécessaires au TP sont fournies dans la dernière partie du sujet.

Algorithme à implémenter

Trame de code

Voici la trame du code à implémenter :

1. Lecture de l'image à segmenter (en niveaux de gris)
2. Initialisation du *snake* :
   - On prendra par exemple comme initialisation un cercle centré au centre de l'image, défini sur K points
   - En superposant le *snake* initial à l'image, assurez-vous qu'il englobe l'objet à segmenter
   - Le *snake* sera défini à partir de ses coordonnées cartésiennes, stockées dans des variables séparées (par exemple, x et y)
3. Initialisation de l'algorithme :
   - Choix des paramètres α, β et γ
   - Calcul de toutes les grandeurs qui n'évolueront pas lors du processus itératif (gradient(s) de l'image, opérateurs différentiels)
   - Conseil : Pour plus de stabilité et pour faciliter le réglage du paramètre γ, le gradient du terme d'énergie externe utilisé pourra être normalisé. On obtiendra alors $G_{externe}$:

     $$G_{externe}[x,y] = \frac{\nabla E_{externe}[x,y]}{\|\nabla E_{externe}[x,y]\|} = \frac{1}{\|\nabla E_{externe}[x,y]\|} \begin{bmatrix} \nabla_x E_{externe}[x,y] \\ \nabla_y E_{externe}[x,y] \end{bmatrix}$$

     Vous veillerez à éviter toute division par zéro lors de l'implémentation de ce calcul.
   - Définition du critère d'arrêt

4. Implémentation de la procédure itérative pour faire évoluer les coordonnées du *snake* jusqu'à convergence (voir formule du cours)
5. Analyse des résultats :
    - Affichage des informations relatives à la convergence (nombre d'itérations, critère d'arrêt, temps de calcul, évolution des différentes énergies, ...)
    - Affichage de l'évolution du *snake* au cours des itérations et de la segmentation finale obtenue à convergence
    - Conseil : L'affichage du *snake* pourra se faire avec la fonction d'OpenCV `cv2.drawContours` (voir Annexe).

Questions subsidiaires

1. Tester votre algorithme sur une image bruitée
2. Comparaison avec le schéma de convergence « explicite » discuté en cours

Aide pour le code

Commandes utiles

```
cv2.imread('imageName.ext', 0) # ouverture d'une image et
 ↪  conversion en niveaux de gris
np.meshgrid() # retourne des matrices de coordonnées
np.square() # élévation au carré de chaque élément d'un array
cv2.Sobel() # applique un filtre de Sobel à une image
cv2.GaussianBlur() # applique un flou gaussien sur une image
sp.diags() # pour la création de matrices parcimonieuses dont
 ↪  on définit les termes diagonaux
np.linalg.inv() # pour calculer l'inverse d'une matrice
```

Commandes pour l'affichage

Affichage standard en niveaux de gris

```
plt.figure() # ouvre une nouvelle figure
plt.imshow(I,'gray') # affichage de l'image I en niveau de
 ↪  gris
plt.show() # déclenche l'affichage
```

Commandes pour l'affichage en *subplot* avec une *colormap* spécifique

```
plt.figure() # ouvre une nouvelle figure
plt.subplot(131) # Image 1
plt.imshow(img1, 'binary') # colormap 'binary'
plt.title('First Image')
plt.subplot(132) # Image 2
plt.imshow(img2, 'gray') # colormap 'gray'
plt.title('Second Image')
plt.subplot(133) # Image 3
plt.imshow(img3, 'jet') # colormap 'jet'
plt.title('Third Image')
plt.show()
```

Sauvegarde d'une figure

La figure courante peut être sauvegardée au format .png avec la commande :
`plt.savefig("filename.png")`

Affichage des contours

La fonction `cv2.drawContours()` peut être utilisée pour superposer un ou plusieurs contours sur une image couleur. Cette fonction a 4 arguments obligatoires, et plusieurs autres optionnels, dont voici la description :

- image : l'image couleur sur laquelle sera superposé le contour
- contours : une liste contenant l'ensemble des contours à afficher. Chaque contour sera mis sous la forme d'un array de dimension : $(K, 1, 2)$ ou K est le nombre de points du *snake*.
- contourIdx : indice du contour contenu dans *contours* qui sera tracé. Une valeur négative entraînera le tracé de tous les contours.
- color : couleur du contour
- thickness : épaisseur du contour

Exemple d'utilisation :

```
cv2.drawContours(image=cv2.cvtColor(I, cv2.COLOR_GRAY2BGR),
 ↪  contours=c, contourIdx=-1, color=(0, 255, 0), thickness=2,
 ↪  lineType=cv2.LINE_AA)
```