TP4
SQUELETTISATION

Préambule :
Pour diminuer la quantité d'information à traiter quand on a affaire au traitement automatique des documents, deux techniques sont utilisées, à savoir :

1. Le contour
2. La squelettisation

Les résultats de ces techniques peuvent être visualisés dans les figures suivantes :

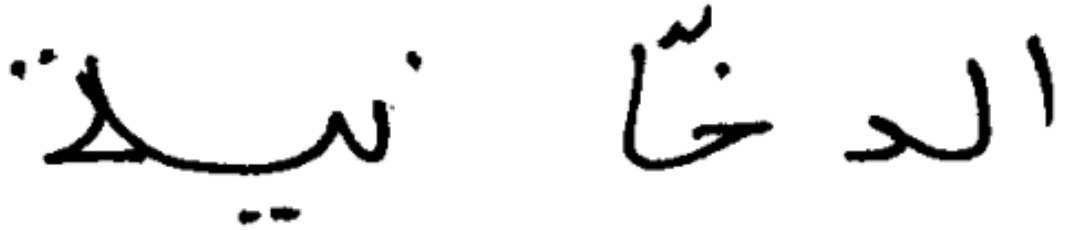

Figure 1. Image originale

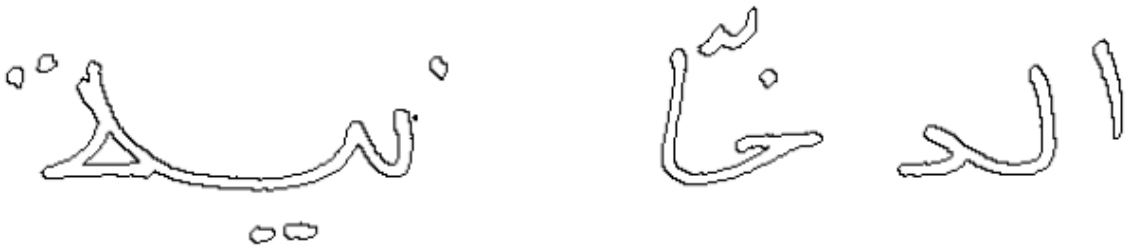

Figure 2. Image contour

Figure 3. Image squelette

Nous remarquons que les trois images transmettent la même information ou texte tout en ayant moins de charge dans la deuxième figure (contour) et beaucoup moins dans la troisième (squelette).

Le but de ce TP est d'implémenter deux algorithmes de squelettisation, très connus, à savoir :

1. L'algorithme de Zhang & Suen
2. L'algorithme de Hilditch

Description de l'algorithme de squelettisation, Zhang & Suen :

Deux étapes seront successivement appliquées à l'image.
Les huit voisins de P dans le sens des aiguilles d'une montre sont comme suit :

P8	P1	P2
P7	**P**	P3
P6	P5	P4

Définition 1
N(P) = le nombre de pixels non nuls voisins de P (= sum(P1, ..., P8))
Définition 2
Tr(P) = le nombre de transitions de 0 à 1, (0 → 1) dans la séquence P1, P2, P3, ..., P8, P1.

Étape 1 :
Tous les pixels sont testés et les pixels satisfaisant toutes les conditions suivantes (simultanément) sont notés à ce stade. Après itération sur l'image et collecte de tous les pixels satisfaisant toutes les conditions de l'étape 1, tous les pixels notés sont mis à 0.

Condition 0 : le pixel vaut 1 et a huit voisins
Condition 1 : 2 < = N(P) < = 6
Condition 2 : Tr(P) = 1
Condition 3 : P1 * P3 * P5 = 0
Condition 4 : P3 * P5 * P7 = 0
Étape 2 :
Tous les pixels sont à nouveau testés et les pixels satisfaisant à toutes les conditions suivantes sont notés à ce stade. Après itération sur l'image et collecte de tous les pixels satisfaisant toutes les conditions de l'étape 2, tous ces pixels notés sont mis à 0.

Condition 0 : le pixel vaut 1 et a huit voisins
Condition 1 : 2 < = N(P) < = 6

Condition 2 : Tr(P) = 1
Condition 3 : P1 * P3 * P7 = 0
Condition 4 : P1 * P5 * P7 = 0

Itération :

Si des pixels quelconques ont été définis dans ce cycle de l'étape 1 ou de l'étape 2, toutes les étapes sont alors répétées jusqu'à ce qu'aucun pixel d'image ne soit ainsi modifié.
Pour cet algorithme, il est à noter qu'à chaque étape la suppression des points notés doit être retardée jusqu'à ce que tous les pixels de l'image aient été visités.

Packages :

```
import cv2
# import numpy as np
import matplotlib.pyplot as plt
```

Lecture, binarisation :

```
# lecture en niveaux de gris
img = cv2.imread('images_tp4/images2.bmp',0)
# binarisation
im = cv2.threshold(img,0,1, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
```

Voisinage :

```
def voisins(im,x,y):
    voisinage = (im[x-1,y], im[x-1,y+1], im[x,y+1], im[x+1,y+1], im[x+1,y],
im[x+1,y-1],im[x,y-1], im[x-1,y-1])
    return voisinage
```

Somme voisinage :

```
def somme_voisins(v):
    return sum(v)
```

Transitions :

```
def transitions(v):
    tr=0
    for i in range(len(v)-1):
        if (v[i]==0 and v[i+1]==1):
            tr+=1
    if (v[-1]==0 and v[0]==1):
            tr+=1
    return tr
```

Squelette :

```
def squelette(img):
    im=img.copy()
    h,w = im.shape
    liste1=1
    liste2=1
    while(liste1 or liste2):
        liste1=[]
        for i in range(1,h-1):
            for j in range(1,w-1):
                if im[i,j]==1:
                    v=voisins(im,i,j)
                    s=somme_voisins(v)
                    tr=transitions(v)
                    if (2 <= s <= 6) and (tr==1) and (v[0]*v[2]*v[4]==0) and
(v[2]*v[4]*v[6]==0):
                        liste1.append([i,j])

        for l,s in liste1:
            im[l,s]=0
        liste2=[]
        for i in range(1,h-1):
            for j in range(1,w-1):
                if im[i,j]==1:
                    v=voisins(im,i,j)
                    s=somme_voisins(v)
                    tr=transitions(v)
                    if (2 <= s <= 6) and (tr==1) and (v[0]*v[2]*v[6]==0) and
(v[0]*v[4]*v[6]==0):
                        liste2.append([i,j])
        for k,m in liste2:
            im[k,m]=0
    return im
```

Appel :

```
thinned = squelette(im)
```

Affichage :

```
plt.figure()
plt.imshow(~im, cmap=plt.cm.gray)
plt.axis('off')
plt.title('original')
plt.figure()
plt.imshow(~thinned, cmap=plt.cm.gray)
plt.axis('off')
plt.title('squelette')
```

Résultats :

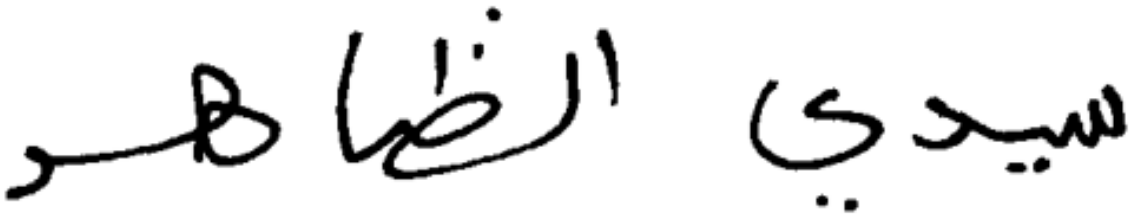

Figure 4. Image originale

Figure 5. Image squelette par Zhang & Suen

Description de l'algorithme de squelettisation, Hilditch

Deux étapes seront successivement appliquées à l'image.
Les huit voisins de P dans le sens des aiguilles d'une montre sont comme suit :

P8	P1	P2
P7	**P**	P3
P6	P5	P4

Définition 1
N(P) = le nombre de pixels non nuls voisins de P (= sum(P1, ..., P8))
Définition 2
Tr(P) = le nombre de transitions de 0 à 1, (0 → 1) dans la séquence P1, P2, P3, ..., P8, P1.

Itération :
Tous les pixels sont testés et les pixels satisfaisant toutes les conditions suivantes (simultanément) sont notés à ce stade. Après itération sur l'image et collecte de tous les pixels satisfaisant toutes les conditions, tous les pixels notés sont mis à 0.

Condition 0 : le pixel vaut 1 et a huit voisins
Condition 1 : $2 <= N(P) <= 6$
Condition 2 : $Tr(P) = 1$
Condition 3 : $P1 \times P3 \times P7 = 0$ ou $Tr(P1) \neq 1$
Condition 4 : $P1 \times P3 \times P5 = 0$ ou $Tr(P3) \neq 1$

Tous les pixels marqués comme effaçables sont ensuite effacés, et l'algorithme est réexécuté sur la nouvelle image, jusqu'à ce que plus aucun pixel ne soit effaçable.

Packages :

```
import cv2
# import numpy as np
import matplotlib.pyplot as plt
```

Lecture, binarisation :

```
# lecture en niveaux de gris
img = cv2.imread('images_tp4/images2.bmp',0)
# binarisation
im = cv2.threshold(img,0,1, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
```

Voisinage :

```
def voisins(im,x,y):
    voisinage = (im[x-1,y], im[x-1,y+1], im[x,y+1], im[x+1,y+1], im[x+1,y],
im[x+1,y-1],im[x,y-1], im[x-1,y-1])
    return voisinage
```

Somme voisinage :

```
def somme_voisins(v):
    return sum(v)
```

Transitions :

```
def transitions(v):
    tr=0
    for i in range(len(v)-1):
        if (v[i]==0 and v[i+1]==1):
            tr+=1
    if (v[-1]==0 and v[0]==1):
            tr+=1
    return tr
```

Squelette :

```
def squelette(img):
    im=img.copy()
    h,w = im.shape
    liste=1

    while(liste):
        liste=[]
        for i in range(1,h-1):
            for j in range(1,w-1):
                if im[i,j]==1:
                    v=voisins(im,i,j)
                    v1=voisins(im,i-1,j)
                    v2=voisins(im,i,j+1)
                    s=somme_voisins(v)
                    tr=transitions(v)
                    tr1=transitions(v1)
                    tr2=transitions(v2)
                    if (2 <= s <= 6) and (tr==1) and (v[0]*v[2]*v[6]==0 or tr1 !=
1) and (v[0]*v[2]*v[4]==0 or tr2 !=1):
                        liste.append([i,j])

        for l,s in liste:
            im[l,s]=0

    return im
```

Appel :

```
thinned = squelette(im)
```

Affichage :

```
plt.figure()
plt.imshow(~im, cmap=plt.cm.gray)
plt.axis('off')
plt.title('original')
plt.figure()
plt.imshow(~thinned, cmap=plt.cm.gray)
plt.axis('off')
plt.title('squelette')
```

Résultats :

Figure 6. Image squelette par Hiltitch