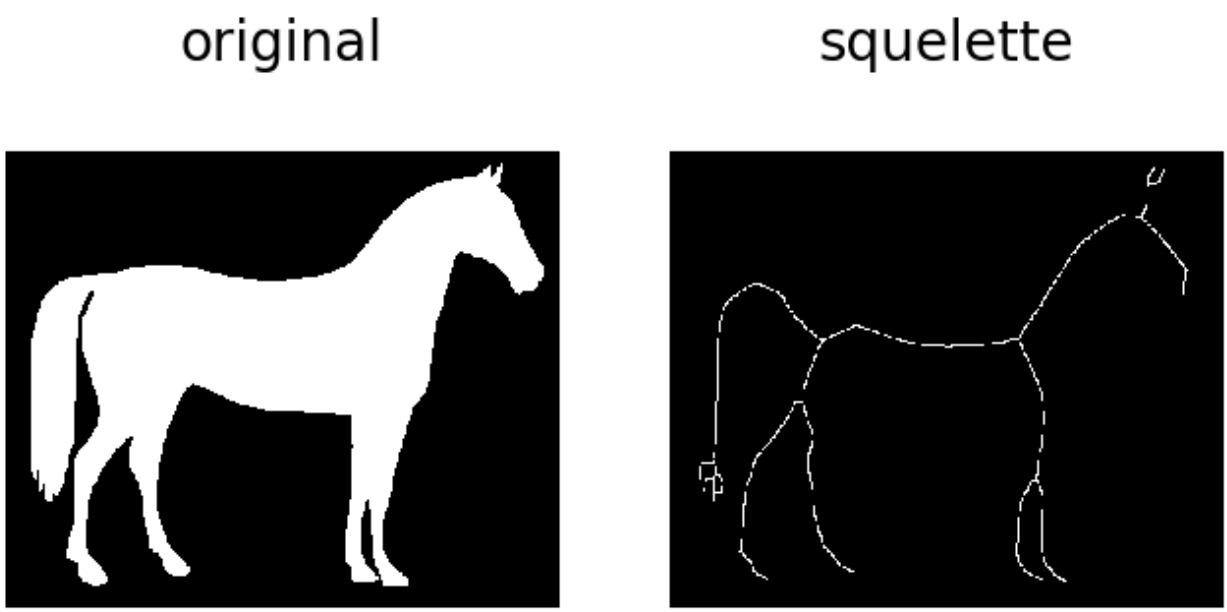
Fait par : MATTIOLI Pierre
DUFOIN Maxime

Encadrant : DUPE François-Xavier

De la silhouette au squelette à la silhouette

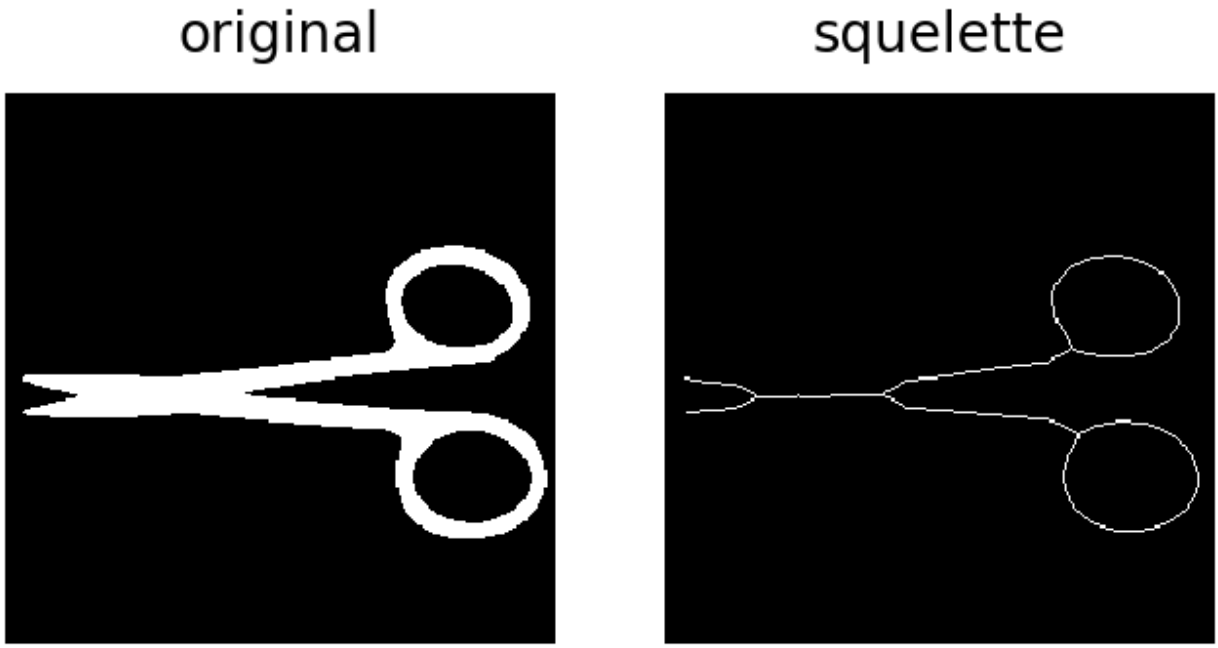


Sommaire

1-Introduction

La squelettisation est un processus qui prend une forme (silhouette) quelconque représentée par un ensemble de points, et amincit celle-ci jusqu’à avoir un simple point d’épaisseur. La nouvelle forme obtenu après cet amincissement est appelé « squelette de la forme ».

La squelettisation est une étape essentielle de la reconnaissance de forme. Elle a pour but de décrire chaque objet par un ensemble de lignes infiniment fines réduisant sensiblement le volume d'information à manipuler.



Voici un exemple d’une image et son squelette : nous voyons à gauche l’image original d’une paire de ciseaux (fichier « scissor05.pgm » du dossier img) et en exécutant le processus d’amincissement sur celle-ci, nous obtenons l’image à droite qui est son squelette.

Voici quelques propriétés du squelette d’une silhouette :

- **l’homotopie** :

  C’est la propriété du squelette la plus importante. Elle consiste à conserver la topologie de l’image.
  En d’autres termes, le squelette doit conserver les relations de connexité : préserver le nombre de composantes connexes de l’objet
  (en l’occurence préserver les trous, les segments et les extremités)

- **la préservation de la géométrie :**

  Le squelette doit rendre compte de la forme globale de la silhouette initial et de sa géométrie.

- **la reconstructibilité :**

  Il est possible de retrouver la forme initiale depuis son squelette.

La squelettisation peut être utile dans des domaines comme la reconnaissance de formes, la modélisation de solides pour la conception et la manipulation de formes, l'organisation de nuages de points, la recherche de chemins, les animations, etc.

2-Description détaillée du sujet

Objectif : Implémenter un programme capable de passer d’une silhouette à son squelette puis retrouver sa silhouette à partir de ce squelette

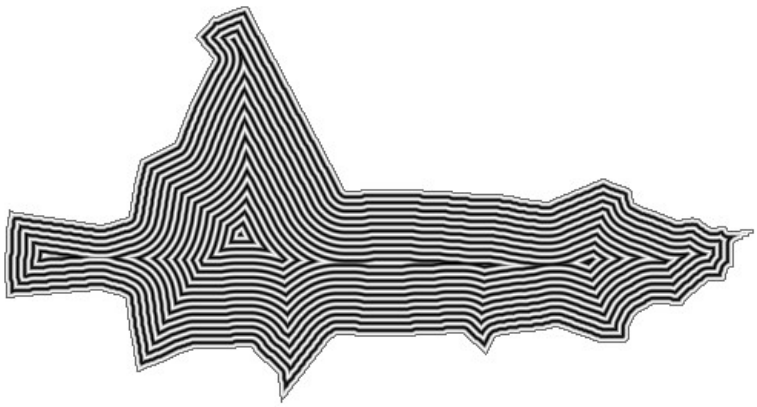
Le but de ce projet est d'implémenter un algorithme d'amincissement de silhouette et de pouvoir reconstituer la forme initiale selon le squelette obtenu.

Pour cela, nous utiliserons le langage de programmation Python ainsi que ses bibliothèques de traitement d’image (scikit-image), de tableaux multi-dimensionnelles (numpy) et de visualisation de données sous forme de graphiques (matplotlib).

Cet algorithme d’amincissement consiste à retirer au fur et à mesure les points du contour de la forme, tout en préservant ses caractéristiques topologiques.

Pour cela, nous allons utiliser le concept du « feu de forêt ».

Ce concept consiste à « brûler » une forme au fur et à mesure à chaque contour, ce qui va grignoter l’image de façon à a peu près uniforme. Chaque point de départ de feu va finir par converger en un simple point « central » de la forme. Ce sont ces points-ci qui nous intéressent.



Chaque barre correspond à un niveau de profondeur

Nous allons donc simuler un feu de forêt et attribuer à chaque pixel de l'image une profondeur relative à sa distance au bord de l'image le plus proche.

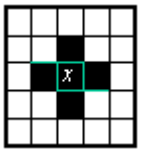
Nous allons ensuite supprimer couche par couche les contours de l'image jusqu'à obtenir une forme squelettique.

Pour savoir quels points doivent être supprimer, il faut comprendre les deux types de point suivant :

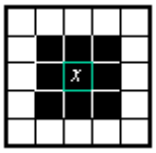
- **point simple :**
  Un point simple est un point dont la suppression n'affecte pas la topologie de l'image restante. C'est-à-dire que chaque point voisin à celui-ci restent connexes entre eux si ce point disparaît.

- **point terminal :**
  Un point terminal est un point qui, soit ne possède qu'un seul voisin, soit créerait une « rupture » de connexion entre ses points adjacents. C'est-à-dire qu'un ou plusieurs points adjacents se retrouverait isolés des autres sans ce point .

La notion de connexité dérive de la notion de voisinage. Il existe principalement deux ordres de connexité : 4 et 8. Ce nombre détermine si l'on prend les voisins diagonales au point initial. Un ordre de connexité 8 prend également compte des diagonales dans sa determination de sa « simplicité ».



4-voisinage



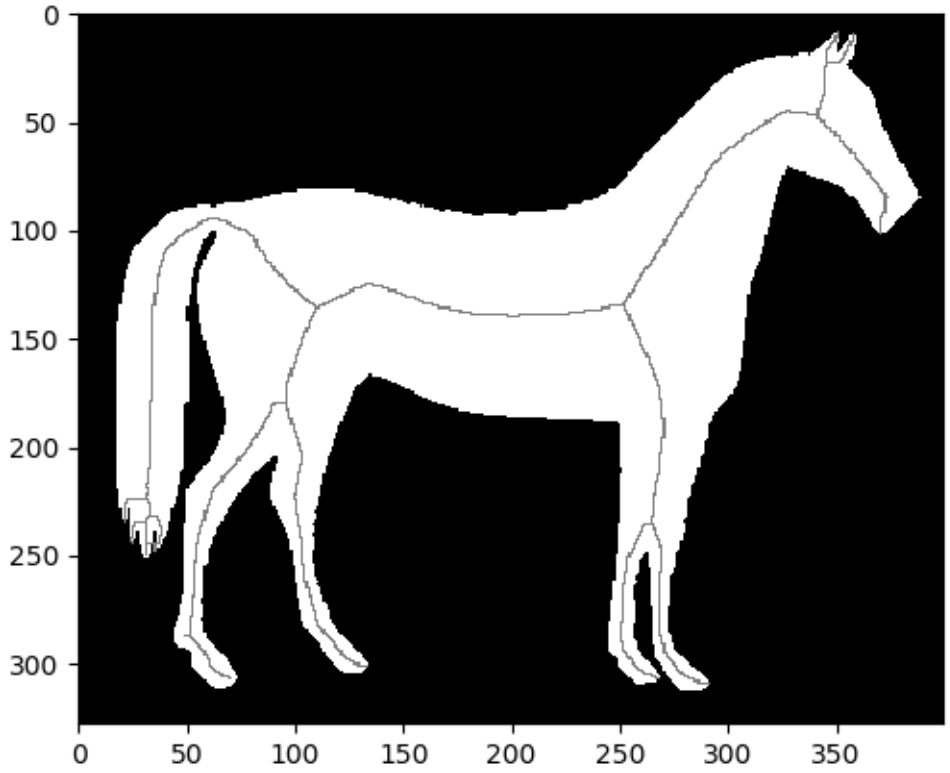
8-voisinage

Chaque squelette peut être différent pour une même image selon la relation de voisinage choisi.

Les points simples et points terminaux ne sont pas identiques pour une relation 4-voisinage que pour une relation 8-voisinage.

Cela ne veux pas dire que l'un est toujours supérieur a l'autre. Un squelette 8d peut etre plus minime mais peut aussi faire une reconstitution plus baclée qu'un squelette 4d comme dans notre forme du cheval.

Une fois notre relation de voisinage choisie, nous l'effectuons couche par couche sur notre forme de base pour obtenir le squelette. Nous pouvons reconstituer la forme à partir de celui-ci en effectuant un procédé inverse au « brûlage » : Une reconstitution uniforme des points en fonctions de la profondeur restant du squelette.



Reconstitution chevalnb.png 4-voisinage
(plus précise qu'une reconstitution 8 voisinage.)

3-Description du travail réalisé

1) Conversion d’images binaires en matrices/tableaux multi-dimensionnelles

Durant tout notre programme, nous allons utiliser une transposition matricielle de l’image pour la traiter.

Dans un premier temps, nous récupérons une image 2D en paramètre.

Notre premier algorithme teste le codage des points de l’image (binaire, coloré, avec transparence) et transforme celle-ci en une matrice représentative de l’image sous forme binaire pure.
Sous sa forme basique cet algorithme determine une forme blanche sur fond noir. Il est possible d’inverser cette représentation pour traiter de formes sur fond blanc, voir sur fond transparent.

Le tout est transformé en une matrice géante aux dimensions de l’image.

Cet algorithme necessite une seule passe de l’image pour analyser chaque pixel.

Pseudo algorithme :

Entrée : image
Debut
    Tester un pixel de l’image pour déterminer son type (binaire, coloré, avec transparence)
    Pour chaque pixel de l’image :
        ajouter dans une matrice m, 0 s’il n’y a aucune couleur,
        1 sinon
    si -inv, inverser les 0 et 1 et de la matrice.
    retourner m
Fin

2) Calcul de profondeur de l’image

Une fois notre matrice de la forme obtenue, nous passons au plus gros du projet.

En premier lieu, nous allons calculer la profondeur de la forme. En nous basant sur le principe du « feu de forêt » vu auparavant.

Nous allons donc parcourir l’image et déterminer la profondeur de chaque pixel relativement à ses « frontières » c’est-à-dire la distance la plus courte vis-à-vis des extrémités de l’image.

Plusieurs méthodes sont possible pour cela. Dans notre cas, notre algorithme calcule la profondeur de chaque pixel en partant des quatres extrémités (coins) de l’image. L’algorithme parcours chaque pixel de la forme et detecte la profondeur cardinale de ses voisins. Ce programme calcule la profondeur selon les 4 directions dans 4 listes différentes.

Une fois toute l’image parcourue, nous récupérons la profondeur minimum de chaque point, et nous obtenons alors une approximation correcte de la profondeur de la forme représentée avec une matrice contenant un entier représentatif de la profondeur.

Cet algorithme requiert une passe de l’image avec 4 séries de tests puis une une autre passe pour affecter la profondeur minimale.

Pseudo Code (determinisation de profondeur):

```
Entrée : Matrice de l’image.
Debut :
Pour chaque pixel de l’image :
        Pour chaque orientation de l’image (Haut-gauche, bas-gauche,haut-droite,bas droite) :
                Si le pixel appartient a l’image :
                        Si ses pixels précédent selon l’orientation n’appartiennent pas
                        a l’image :
                                profondeur du pixel=1
                        sinon :
                                profondeur du pixel=plus petite profondeur précédente+1
        Pour chaque pixel de l’image :
                Affecter la plus petite profondeur entre les quatre a ce pixel.
Fin
```

3) Fonction de recherche des points simples et terminaux

La prochaine étape consiste à utiliser notre matrice des profondeurs et d'y appliquer le processus d'amincissement.

Cependant, nous ne pouvons pas parcourir l'image entière à chaque niveau de profondeur afin de savoir quels pixels traiter, sans rendre exponentiel le temps de calcul.

Pour simplifier le processus, nous avons implémenté un système de mémoire interne sous forme de file d'attente. Nous implémentons une liste qui contiendra chaque points de la forme.

A l'aide d'un tableau de compteur, nous effectuons une seule passe de l'image, et insérons chaque paire coordonnées du pixel en fonction de sa profondeur : à chaque nouveau pixel, nous incrémentons la case du tableau égale a la profondeur dudit pixel. Puis nous insérons ces coordonnées dans la liste à la position correspondant à la somme de tout les compteurs de profondeurs inférieures précédents a celle ci.

Le résultat final est une liste géante contenant chaque pixel de la forme classés par ordre de pronfondeur. Nous utilisons alors cette liste dans notre algorithme précédent.

Pseudo code (file d'attente de priorité de profondeur) :

```
Entrée :Matrice de l'image avec priorité, un tableau de compteur (de taille de la
        plus grande profondeur, remplis de 0), une liste (vide.)
Debut
        Pour chaque pixel de l'image
                si le pixel possède une profondeur:
                        incrémenter le compteur du tableau a la case de cette profondeur
                        insérer les coordonées du pixel dans la liste a la position n=somme
                        des compteurs de profondeurs précédente.
        Retourner la matrice.
Fin
```

Maintenant nous allons, pour chaque niveau de profondeur, procéder a notre algorithme de determinisation pour savoir un pixel donné est « simple » ou « terminal ».

Cet algorithme consiste à :

- Déterminer quel point cardinal du pixel est une « frontière », c'est-à-dire quel pixel Nord/Sud/Est/Ouest n'appartient pas à la forme donnée.

- Calculer si ce point est ou non un point terminal vis à vis de cette frontière.
  Le procédé (et les résultats) étant différents selon que l'on fasse une detection 4d ou 8d, deux algorithmes différents existent pour accomoder cette tâche.
  Les deux algorithmes déterminent si un point est terminal ou pas dans une complexité entre 1 et 4 test pour la 4d, et 1 et 5 test pour la 5d.

- Cette tâche s'effectue sequentiellement pour chaque direction cardinale de la facon suivante : Parcours de chaque pixel frontalier a cette cardinalité et determinisation de son status, puis suppression potentielle de chaque pixel marqué avant de passer à une cardinalité différente.

Il est nécessaire de faire le troncage des pixels par cardinalités plutot que par profondeur pour avoir un resultat correct.

Une fois le procédé effectué pour les quatres cardinalités, nous augmentons notre profondeur et répétons le processus.

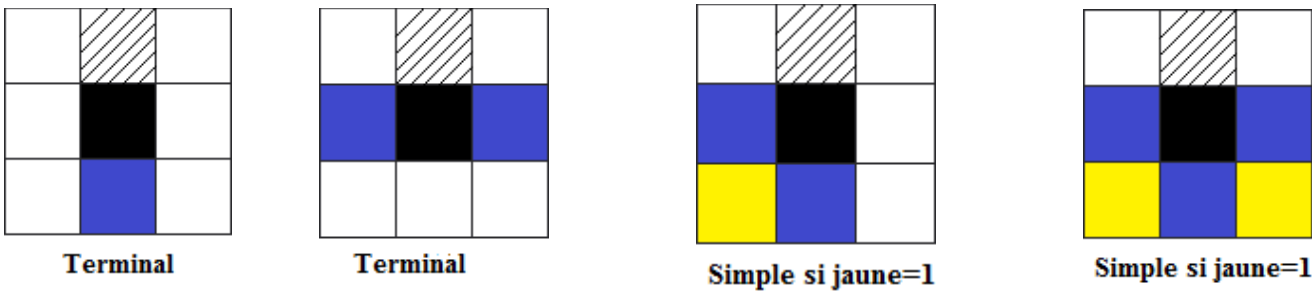
```
Entrée :
    liste des pixels de la forme par ordre de profondeur
Debut
    Pour chaque niveau de profondeur :
        Pour chaque pixel de ce niveau :
            determiner les frontières du pixel (cotés n'appartenant pas a la forme)
            Pour chaque point cardinal (N,S,E,O) :
                Pour chaque points frontière de cette cardinalité :
                    si le pixel est simplifiable(voir plus bas) :
                        le marquer pour suppression
                supprimer les pixels marqués.
Fin
```

4) Algorithme de simplification

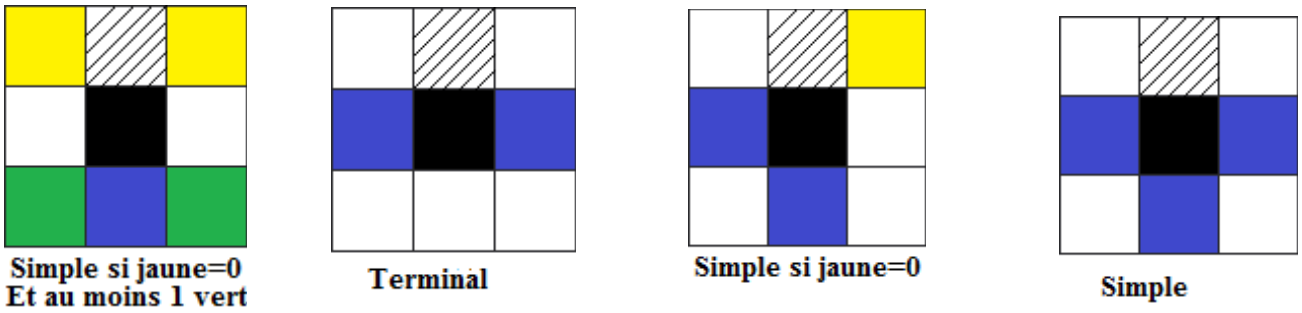
Cet algorithme determine le statut d'un point en un minimum de test. Selon la dimension donnée (4d ou 8d) les tests sont différents.

L'algorithme détermine dabord le nombre de points adjacents (non diagonales ou frontière) puis selon le nombre de points trouvé procède a un ou deux test supplémentaire pour determiner la simplicité du point ou non. Aucun point adjacent est forcément terminal.

Tests de simplifications 4d :



Tests de simplification 8d :



Bien évidemment chaques test est adapté a la cardinalité de la frontière et la positions des points adjacents présents.

L’ordre des cardinalités pour la simplification peut etre determiné par l’utilisateur (défaut N,E,S,O) mais les squelettes résultant sont presques similaires.

Une fois le calcul effectué sur chaque élément de la liste, les élements de la matrice restante correspondent au squelette de l’image.

Notre programme affiche ensuite 3 images possibles : Une carte de la profondeur simple, le squelette (par défaut), ou bien le squelette en conservant la profondeur des pixels restant. Dans les deux derniers cas il sauvegarde dans un fichier la matrice du squelette.

5) Reconstitution

Avec cette matrice de squelette de profondeur obtenu, nous pouvons effectuer une reconstitution approximative de l’image de base. Pour cela, nous restituons d’abord pour chaque pixel du squelette sa profondeur en distance dans les 4 directions. Ce processus restaure en grande partie l’image de base.

Cependant, pour quelques extrémités, l’image peut être tronquée selon l’efficacité de l’algorithme de squelettisation : les formes bombées en extrémités notamment.

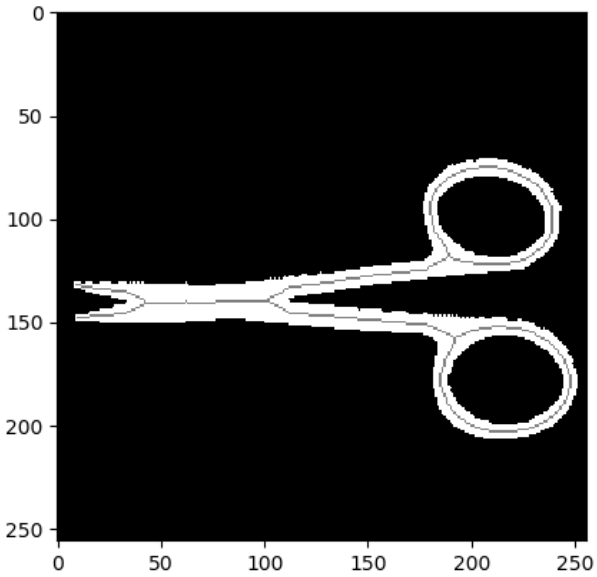
Pour contrer cela, nous pouvons alors compléter l’image en ajoutant les points dans un cercle au rayon équivalent à la profondeur du pixel du squelette. En utilisant la distance euclidienne, nous obtenons un cercle de rayon approximatif pour compléter la forme d’une façon un peu plus correct.

En l’éxécutant uniquement sur les points aux extrémités du squelette, nous obtenons des formes plus proche du programme précédent sans « gonfler » l’image. Ce procédé peut être configurable ou facultatif en paramètre selon l’image.

Pseudo code (reconstitution)
Entrée : Matrice du squelette.
Debut
Pour chaque point du squelette :
    Remplir chaque points dans les quatre directions a une distance correspondant à la profondeur du point originel (y compris en diagonale)
    (optionellement) si le point du squelette est une extrémité :
        remplir chaque point autour de ce pixel à une distance euclidienne égale a sa profondeur.
Retourne une matrice de l'image reconstituée.
Fin

Le procédé nécéssite une seule passe de la matrice de l'image pour trouver les pixels du squelette.

Le programme de reconstitution peut afficher la forme reconstituée simple ou également afficher le squelette sur lequel il à travaillé en fond.



Reconversion scissor05.pgm 8-voisinage

Manuel d’exécution de notre algorithme de squelettisation et de reconstitution

Squelettisation

Le programme principal « projet.py » nécessite uniquement en paramètre obligatoire le nom de l’image.

Par la suite, diverses options sont disponibles pour avoir des resultats différents.

- Ajouter -**p** pour afficher simplement la profondeur de l’image ou -**ps** pour afficher le squelette avec sa profondeur.

- Ajouter -**4d** pour faire une simplification avec un voisinage 4d (par défaut **8d**)

-Ajouter -**inv** pour inverser la matrice de l’image. Par défaut le programme prend des images blanc sur fond noir (comme les .pgm) cependant avec l’option -inv le programme peut traiter des images noires ou en couleurs sur un fond blanc ou transparent.

Enfin, vous pouvez determiner l’ordre de cardinalité de l’algorithme de squelettisation (c’est a dire les directions par lequelle l’algorithme simplifie l’image) en ajoutant les 4 initiales en un mot de quatre lettres.
Exemple : « **OSNE** » effectuera une simplification par l’ouest, sud, nord, est.
(défaut : Nord Est Sud Ouest)

Reconstitution

Après avoir exécuté au moins une fois le programme principal (hors option -p) lancez le programme reconversion.py
Utilisez soit :

-Le paramètre -f affiche la forme reconstituée simple.
-Le paramètre -s affiche la forme plus le squelette utilisé en gris.

Enfin, vous pouvez ajouter un nombre entier si vous voulez appliquer un affinement circulaire sur les points terminaux du squelette. (un nombre >0 réduit le rayon, peu nécessaire sur nos exemples)

4-Conclusion

En utilisant des outils relativement basiques de python, nous avons tout de même pu faire un algorithme de squelettisation et reconstitution correct.

Comme dis en introduction, il est difficile de reconstituer une image parfaitement identique à l'image d'origine.

Pour une image de base (200 000 pixels), sur un ordinateur moyen, notre algorithme met un temps de quelques secondes.

Nous avons cherchés diverses manières de plus simplifier le processus mais cela semble difficile.

Nous avons également travaillé sur d'autres fonctionnalités (comme la possibilité de flouter une image (extension .jpg) ou une amélioration de la reconstitution notamment les formes rectangulaires) mais n'avons eu le temps de les perfectionner.

Enfin, nous aurions aimé pouvoir approfondir nos algorithmes en intégrant la troisième dimension des images.

5-Bibliographie/Annexes

Bibliographie

Bibliothèque Scikit-image de Python :
http://scikit-image.org/

Base de données d'images :
https://iapr-tc15.greyc.fr/links.html#Benchmarking and data sets

Bibliothèque Numpy de Python :
http://www.numpy.org/

Bibliothèque Matplotlib de Python :
https://matplotlib.org/

Annexes

Fichiers PDF disposés dans le dossier « annexes/pdf » du rapport :

- TER_silhouette.pdf : algorithmes/pseudo-codes concernant la squelettisation

- Saito94.pdf : algorithmes/définitions concernant la distance euclidienne

Notre code Python est disponible dans le dossier « annexes/code » du rapport :

- projet.py : programme principal (permettant d'obtenir la squelettisation)

- reconversion.py : programme permettant la reconstitution de la forme initiale

- fonctions.py : contient les fonctions nécessaires au fonctionnement du programme principal « projet.py »

Des images sont aussi disposés dans le dossier « annexes/images » du rapport pour tester nos algorithmes (exemple : scissor05.pgm, key03.pgm, chevalnb.png, etc.)