Cours “Segmentation Morphologique”
MASTER OIV, Université Jean Monnet, Saint-Etienne
Corrigé de l’examen écrit (février 2014)

Serge BEUCHER
CMM, Mines ParisTech

Question 1
Cochez par un X les propriétés de chaque transformation.

Solution
Voici les réponses correctes:

	Croissance	Extensivité	Anti-extensivité	Idempotence	Homotopie	
Dilatation générale	X					(1)
Erosion	X					
Dilatation géodésique	X	X				(2)
Erosion géodésique	X		X			
Ouvertures (générales)	X		X	X		(3)
Fermetures	X	X		X		
Filtres	X			X		(4)
Erosion ultime			X	X		(5)
Amincissement (de façon générale)			X			(6)
Epaississement (de façon générale)		X				
Opérateur thinL			X	X	X	(7)
Opérateur thinD			X	X	X	
Opérateur thickD		X		X	X	
Ligne de partage des eaux [1]			X	X		(8)

(1) La dilatation, de façon générale, n’est pas extensive (cette supposée propriété de la dilatation est un piège classique). Elle n’est extensive que lorsque l’origine de l’élément structurant appartient à l’élément structurant. De la même façon, l’érosion n’est pas anti-extensive sauf si l’origine de l’élémént structurant appartient à cet élément structurant.
(2) La dilatation géodésique est en revanche extensive. En effet les éléments structurants étant des boules géodésiques, ils contiennent toujours leur origine. La dilatation géodésique n’est pas idempotente. Seule son itération infinie (reconstruction géodésique) l’est. Même remarque pour l’érosion géodésique qui est anti-extensive.

[1] On considère la ligne de partage des eaux valuée

(3) Ces trois propriétés des ouvertures et des fermetures constituent leur définition même.
(4) Croissance et idempotence sont les deux propriétés qui définissent un filtre morphologique.
(5) L'érosion ultime, comme tous les opérateurs résiduels, n'est pas croissante. Cependant, elle est anti-extensive et idempotente. L'érodé ultime d'un ensemble ou d'une fonction est toujours contenu dans l'ensemble ou inférieur ou égal à la fonction. Effectuer l'érosion ultime d'un érodé ultime ne change évidemment rien.
(6) La seule propriété d'un amincissement général est d'être anti-extensif (extensif pour l'épaississement). Ces deux transformations ne sont en aucun cas croissantes!
(7) thinL comme thinD sont obtenus par itération d'amincissements utilisant des éléments structurants préservant l'homotopie. Ils sont donc anti-extensifs, idempotents (à cause de l'itération infinie) et homotopiques. L'opérateur thickD est lui extensif, idempotent et homotopique.
(8) On demandait de considérer la ligne de partage des eaux valuée. L'opération consistant à réaliser la ligne de partage des eaux valuée d'une fonction numérique (supposée positive ou nulle - une image de gris) est anti-extensive (par définition) et idempotente (à condition de toujours utiliser le même algorithme pour la réaliser). En revanche, elle n'est pas homotopique. En effet les dômes (maxima) de la fonction initiale peuvent être supprimés lors de l'opération. On dit parfois que la LPE est semi-homotopique.

Question 2
Filtrage de particules selon le nombre de trous (chapitre 8 page 34). Utiliser l'image *gruyere* et l'image *noise*.

a) Définissez un algorithme permettant de séparer les particules avec trous des particules sans trou(s) (simplement connexes).

b) Parmi les particules avec trou(s) , séparer celles avec un seul trou.

c) Parmi les particules avec plus d'un trou, extraire celles qui ont deux trous seulement.

d) Pouvez-vous définir un algorithme pour extraire toute particule avec au plus n trous?
Toute validation de vos algorithmes (code, images, résultats, etc.) est la bienvenue.

e) La transformation $\Psi_n(X)$ qui ne conserve que les particules ayant plus de n trous est-elle une ouverture algébrique?

Eléments de solution
Ce problème de formulation assez évidente (filtrer des composantes connexes en fonction de leur nombre de trous) peut se révéler assez complexe à résoudre si on cherche des solutions à la fois rapides, élégantes et universelles. Cependant, rien dans l'énoncé n'exigeait qu'une telle solution soit proposée. C'est pourquoi la solution basée sur une analyse individuelle des particules proposée par certains d'entre vous est certes triviale et lente mais... elle marche!
La mise en oeuvre de cette solution est la suivante:
- L'image initiale est labellisée. L'opérateur ***label*** renvoie le nombre N de composantes connexes présentes dans l'image. **Attention!** L'opérateur label de Mamba n'utilise pas l'ensemble des valeurs entières consécutives comprises entre 1 et N. Les valeurs multiples de 256 ne sont pas utilisées. C'est pourquoi, plutôt que le nombre de particules, il est préférable d'utiliser la valeur de label maximum utilisée par l'opérateur. Cette valeur est fournie par ***computeRange***. De plus, une valeur de i égale à 0 modulo 256 signifie qu'elle n'est pas utilisée et qu'on peut passer à la valeur suivante.
- La i-ème particule est extraite par un seuillage à la valeur i de l'image labellisée.

- La mesure du nombre de connexité ν de la composante connexe fournit son nombre de trous n_T par la formule $n_T = 1 - v$.
- Toute particule dont le nombre de trous est égal à une valeur donnée est ajoutée à l'image résultat.

L'implémentation de cette procédure dans Mamba peut être réalisée comme suit:

```
def holesSieving(imIn, imOut, n):
    """
    Puts in imOut all the connected components of imIn which contains exactly
    n holes.
    """

    imWrk0 = imageMb(imIn, 32)
    imWrk1 = imageMb(imIn)
    imOut.reset()
    # Initial image labelling
    nParticles = label(imIn, imWrk0)
    vLabel = computeRange(imWrk0)[1]
    for i in range(vLabel):
        # each particle is extracted and its connectivity number
        # is measured
        if ((i + 1) % 256) <> 0:
            threshold(imWrk0, imWrk1, i+1, i+1)
            nc = computeConnectivityNumber(imWrk1)
            # if the number of holes is equal to n, the particle is
            # added to the output image
            if (n==(1 - nc)):
                logic(imWrk1, imOut, imOut, "sup")
```

La recherche de solutions plus rapides et plus élégantes conduit à analyser ce problème étape par étape et à commencer par trouver des approches permettant de séparer les particules trouées de celles simplement connexes.

(a) Extraction des particules simplement connexes

Deux pistes principales ont été explorées dans les réponses fournies: la première basée sur l'utilisation de transformées homotopiques, la seconde utilisant la reconstruction géodésique (ces deux approches sont utilisées dans la solution fournie sur le site web de Mamba, comme les plus curieux d'entre vous n'ont pas manqué de le découvrir!).

Une première solution consiste à utiliser l'opérateur thinD. Cette itération d'amincissements jusqu'à idempotence réduit toute composante simplement connexe à un seul point. Il suffit alors, après avoir effectué cette transformation, de supprimer les points isolés et de reconstruire l'ensemble initial à l'aide de marqueurs restants. Ces points isolés peuvent être extraits à l'aide d'une transformée en tout-ou-rien avec le double élément structurant suivant (en trame hexagonale):

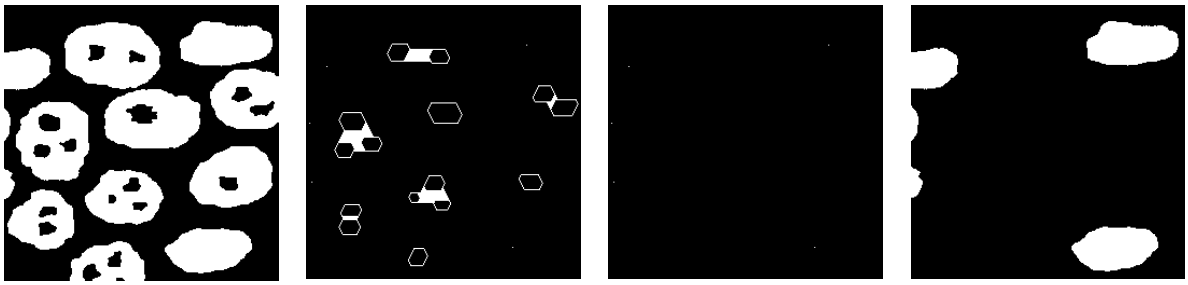
0 0

0 1 0

0 0

Ils peuvent également être supprimés par une érosion linéaire élémentaire (dans n'importe quelle direction). Il est important d'utiliser une érosion linéaire et non hexagonale, cette dernière étant en effet susceptible de supprimer également des marqueurs de composantes non simplement connexes. Les différentes étapes du processus sont données et illustrées ci-dessous.

```
# L'image initiale est contenue dans imbin1.
# Amincissement de imbin1 avec D, résultat dans imbin2.
thinD(imbin1, imbin2)
# Extraction des points isolés dans imbin3.
hitOrMiss(imbin2, imbin3, 126, 1)
# Reconstruction géodésique de l'image initiale avec les points isolés.
build(imbin1, imbin3)
# imbin3 contient les particules simplement connexes.
```



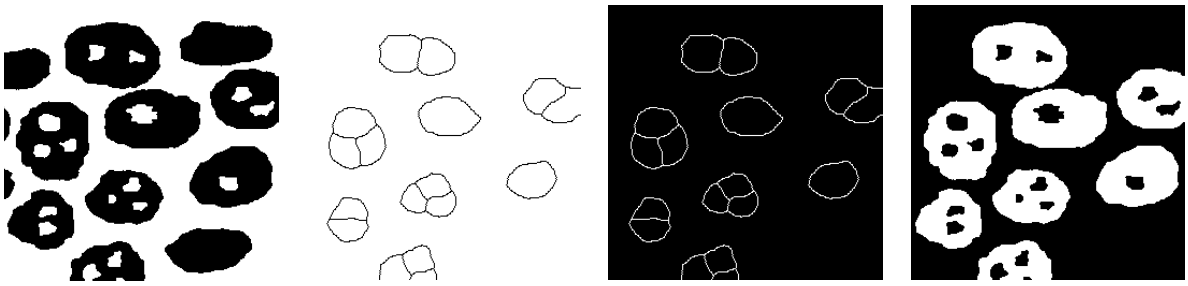
De gauche à droite: image initiale, amincissement par D, points isolés, reconstruction de l'image initiale par les points isolés (particules simplement connexes).

On peut aussi réaliser le squelette par zones d'influence du complémentaire de l'ensemble initial. Les composantes connexes sans trous ne contiennent alors aucune partie du squelette par zones d'influence. Elles ne seront donc pas marquées et donc reconstruites par ce squelette.

```
# L'image initiale est contenue dans imbin1. Elle est inversée et stockée dans imbin2.
negate(imbin1, imbin2)
# Le squelette par zones d'influence est effectué et placé dans imbin3.
fastSKIZ(imbin2, imbin3)
negate(imbin3, imbin3)
# La reconstruction permet d'extraire les particules avec trou(s) et de les soustraire
# de l'image initiale.
build(imbin1, imbin3)
diff(imbin1, imbin3, imbin3)
```

Cette solution est plus rapide que la précédente car elle utilise l'opérateur fastSKIZ basé sur la Ligne de Partage des Eaux, ce qui permet d'obtenir les zones d'influence très rapidement et

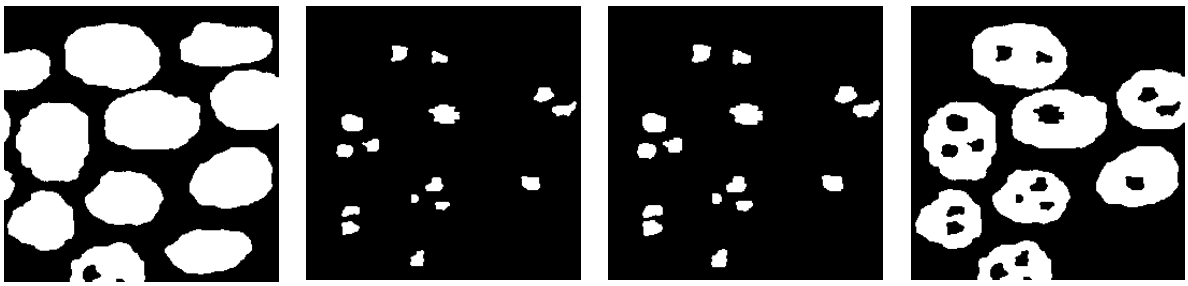
surtout en un temps indépendant de la taille des composantes connexes.



De gauche à droite: image initiale inversée, zones d'influence des trous, SKIZ, particules avec trou(s).

L'autre approche proposée consiste à reconstruire les composantes connexes marquées par des trous. Après avoir bouché et extrait les trous, ceux-ci, après une dilatation élémentaire, marquent les particules qui les contiennent. Ces dernières peuvent alors être reconstruites.

```
# L'image initiale est contenue dans imbin1.
# Les trous sont bouchés et extraits. Ils sont utilisés (après une dilatation élémentaire)
# comme marqueurs pour reconstruire les particules avec au moins un trou.
closeHoles(imbin1, imbin2)
diff(imbin2, imbin1, imbin2)
dilate(imbin2, imbin2)
build(imbin1, imbin2)
# imbin2 contient maintenant les particules avec trou(s).
```



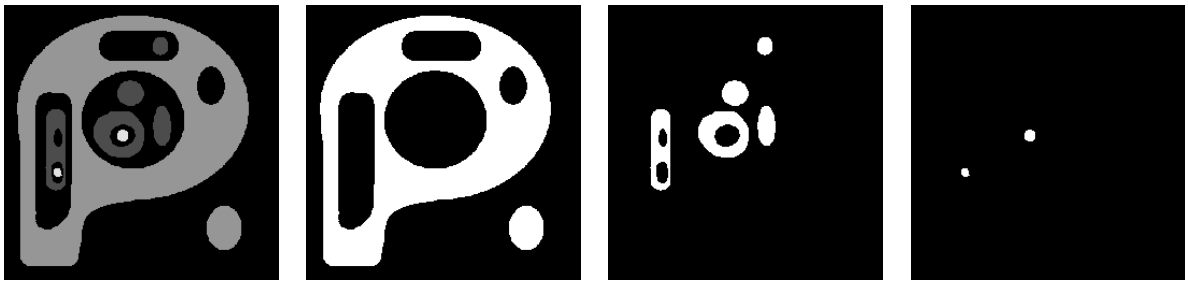
De gauche à droite: bouchage des trous, trous isolés, trous dilatés, résultat de la reconstruction géodésique (particules avec trous).

On peut penser que cette dernière approche fournit un résultat identique aux approches précédentes. En fait, il n'en est rien! En effet, on peut être confronté à des structures assez complexes où des trous peuvent contenir des composantes connexes qui peuvent elles-mêmes contenir des trous, constituant une sorte de "mise en abyme" de la structure élémentaire constituée par l'association particule/trou(s). L'image *gruyere* ne présente pas cette structure complexe. En revanche, l'image ci-dessous en est une bonne illustration. Une telle structure peut être représentée par un arbre, appelée arbre d'homotopie. Le tronc de l'arbre correspond au fond de l'image, les premières branches correspondent aux composantes connexes adjacentes à ce fond. Les branches suivantes correspondent aux trous adjacents aux composantes connexes précédentes, etc. On peut également représenter cet arbre sous forme d'une image dans laquelle on attribue à chaque composante connexe un label correspondant à sa position dans l'arborescence. La procédure suivante permet de construire cette image.

Image présentant une imbrication des composantes connexes et des trous beaucoup plus complexe que l'image gruyere (à gauche). A droite, représentation sous forme d'une image labellisée de l'arbre d'homotopie..

```
def homotopyTreeBuild(imIn, imOut):
    """
    Builds the homotopy tree of the initial binary set imIn. The result is stored
    in image imOut. Each grey level corresponds to a hierarchy level of the homotopy
    tree. Extracting each level i of embedding of particles and holes of the initial
    set consists simply in a [i, i] thresholding of imOut.
    """

    imWrk0 = imageMb(imIn)
    imWrk1 = imageMb(imIn)
    imWrk2 = imageMb(imIn, 8)
    copy(imIn, imWrk0)
    v = computeVolume(imIn)
    imOut.reset()
    i = 0
    # Loop performed until no connected component remains.
    while v!=0:
        i = i+1
        # Background extraction.
        closeHoles(imWrk0, imWrk1)
        negate(imWrk1, imWrk1)
        # Connected components adjacent to the background are rebuilt.
        dilate(imWrk1, imWrk1)
        build(imWrk0, imWrk1)
        # These particles are at level i in the homotopy tree.
        # They are removed from the current set, giving access to the next level (in imWrk0).
        diff(imWrk0, imWrk1, imWrk0)
        v = computeVolume(imWrk0)
        # The level-i particles are labelled with i and added to the label image.
        convertByMask(imWrk1, imWrk2, 0, i)
        add(imOut, imWrk2, imOut)
```

Chaque niveau de hiérarchie i de l'arbre d'homotopie peut facilement être obtenu par un simple seuillage de l'image label à la valeur i. Le dernier algorithme proposé doit alors être appliqué sur chaque image seuillée.



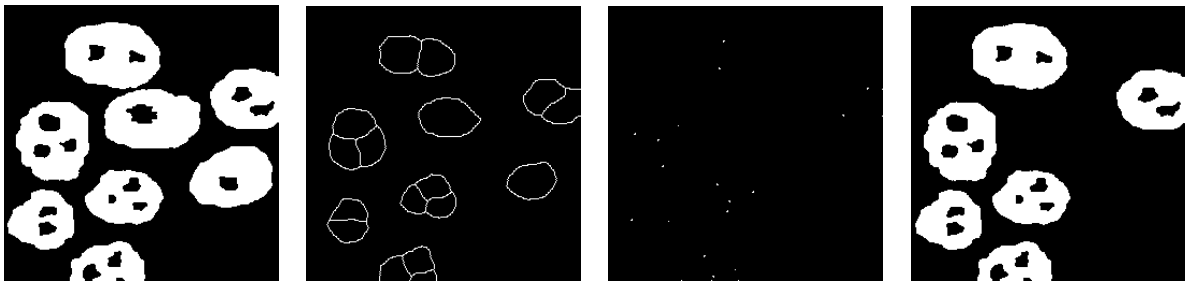
L'arbre d'homotopie et les différents niveaux de hiérarchie des structures particles/trous imbriquées. L'algorithme précédent doit être appliqué sur chacune d'entre elles pour obtenir un résultat correct.

A l'inverse, l'utilisation de l'approche basée sur le SKIZ sur l'image initiale fournit directement le résultat escompté sans nécessité de représenter et de décomposer cette image initale avec l'arbre d'homotopie.

(b) Extraction des particules avec un seul trou.
L'utilisation du SKIZ dans la question précédente fournit une solution immédiate à ce problème. On constate en effet que le SKIZ des trous ne présente aucun point multiple lorsqu'une particule ne contient qu'un seul trou. Il suffit donc d'extraire ces points multiples (l'opérateur ***multiplePoints*** de Mamba fait parfaitement l'affaire) et de les utiliser comme marqueurs pour reconstruire et soustraire les particules ayant plus d'un trou.

```
# Séparation des particules avec un seul trou de celles qui en contiennent davantage.
# imbin1 contient uniquement des composantes connexes avec trous(s).
negate(imbin1, imbin2)
fastSKIZ(imbin2, imbin2)
negate(imbin2, imbin2)
# Les particules avec plusieurs trous sont marquées par des points multiples du SKIZ.
multiplePoints(imbin2, imbin2)
# Reconstruction des particules avec plus d'un trou.
build(imbin1, imbin2)
# Suppression de l'image initiale. imbin2 contient seulement des particules avec un trou.
diff(imbin1, imbin2, imbin2)
```

Cet approche cependant fait apparaître un phénomène assez gênant. La particule située en bas de l'image est considérée comme ayant trois trous comme le montre l'illustration ci-dessous.



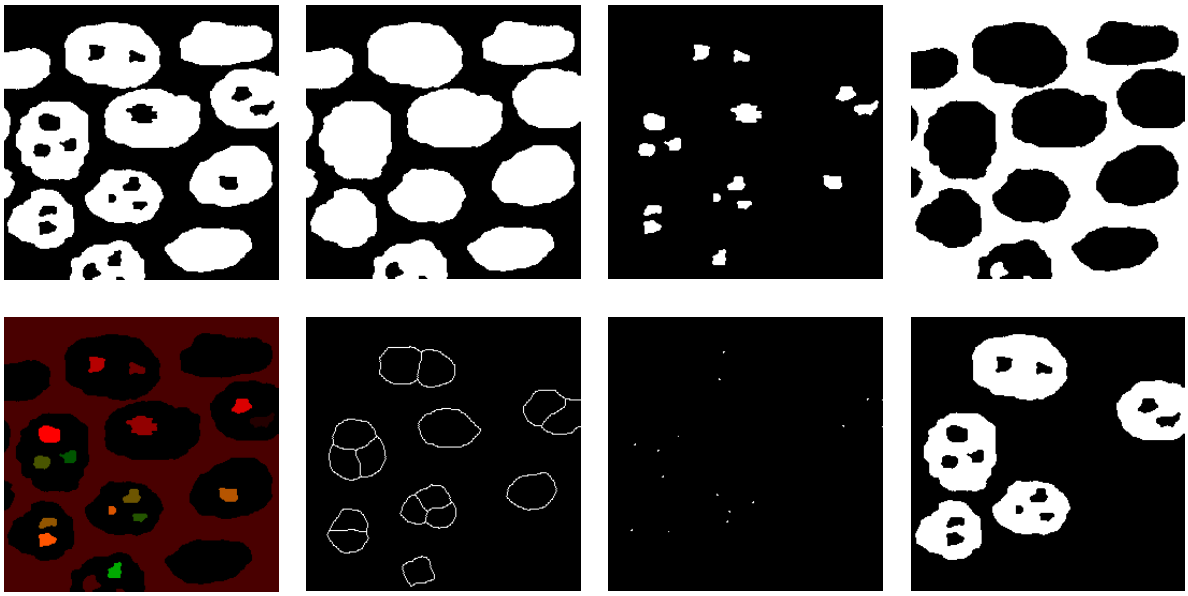
De gauche à droite: particules avec trou(s), SKIZ des trous, points multiples du SKIZ, reconstruction finale (particules avec au moins deux trous). Cette solution considère que la particule en bas de l'image a trois trous!

A quoi est dû ce résultat? Il provient de la façon dont est réalisé le SKIZ. En effet, cette transformation est définie comme la ligne de partage des eaux du complémentaire de l'ensemble initial. Avec cette définition, toute composante connexe du fond est considérée comme une source d'inondation. C'est donc en particulier le cas pour les deux "pseudo-trous" de la particule concernée. Comment peut-on corriger cela? Simplement en remplaçant la ligne de partage des eaux dans le SKIZ qui est une ligne de partage des eaux classique par une ligne de partage des eaux contrôlée par marqueurs où les sources d'inondation sont, d'une part les (vrais) trous, d'autre part le fond de l'image, fond auquel appartiennent les "pseudo-trous", ce marqueur agissant comme une source d'inondation unique. Voici la suite des opérations à effectuer:

```
# L'image initiale est dans imbin1. On bouche les trous. L'image imbin2
# contient les particules sans trous et imbin3 les trous.
closeHoles(imbin1, imbin2)
diff(imbin2, imbin1, imbin3)
# L'inversion de l'image sans trou permet de générer le marqueur du fond contenant
# également les pseudo-trous (image imbin4).
negate(imbin2, imbin4)
# Les (vrais) trous sont labellisés.
label(imbin3, im32_1)
# On ajoute 1 à toutes les valeurs en additionnant imbin3 à im32_1. Le label
# minimum des trous est donc égal à 2.
add(im32_1, imbin3, im32_1)
# On ajoute le fond dans l'image label (toutes les composantes connexes du fond
# ont un label égal à 1).
add(im32_1, imbin4, im32_1)
# L'image initiale est convertie en une image 8-bit à 2 valeurs (0 et 1).
convertByMask(imbin1, im2, 0, 1)
# Sa ligne de partage des eaux contrôlée par l'image label est réalisée.
# elle est transférée dans une image 8-bit puis binaire.
watershedSegment(im2, im32_1)
copyBytePlane(im32_1, 3, im3)
threshold(im3, imbin4, 1, 255)
# Les points multiples sont extraits et les particules marquées reconstruites.
# imbin3 contient les particules avec 2 trous ou plus.
multiplePoints(imbin4, imbin3)
build(imbin1, imbin3)
```

Toutes ces étapes sont illustrées ci-après .

On peut également utiliser le squelette par zones d'influence géodésique. Le SKIZ géodésique est réalisé avec une ligne de partage des eaux mais, contrairement au SKIZ, elle est contrôlée par marqueurs, ce qui va nous permettre, moyennant un bon choix de ces marqueurs, d'obtenir une solution très semblable à la soluton précédente.
Considérons l'ensemble initial. Bouchons tous les trous et effectuons le SKIZ géodésique des trous dans l'espace géodésique constitué par cet ensemble sans trou. Ce SKIZ geodesique sépare en plusieurs morceaux les particules qui possédent au minimum deux trous. Il suffit alors de reconstruire les particules marquées par ce SKIZ géodésique pour obtenir les particules avec plus d'un trou.

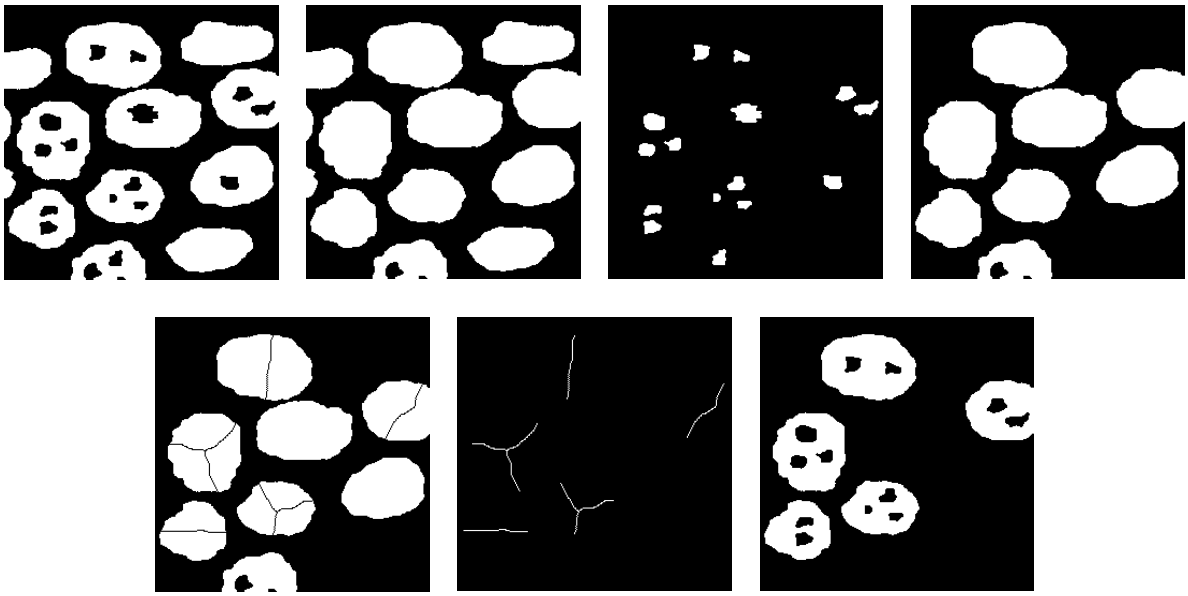


De gauche à droite et de haut en bas: image initiale, image sans trous, trous détectés, marqueur du fond (image sans trou inversée), image label (les trous sont labellisés ainsi que le fond dont toutes les composantes conexes reçoivent une valeur identique), ligne de partage des eaux de l'image initiale, extraction des points multiples et reconstruction des particules marquées. La particule du bas est maintenant justement considérée comme ne contenant qu'un trou.

Les différentes étapes de cette procédure utilisant le SKIZ géodésique sont les suivantes:

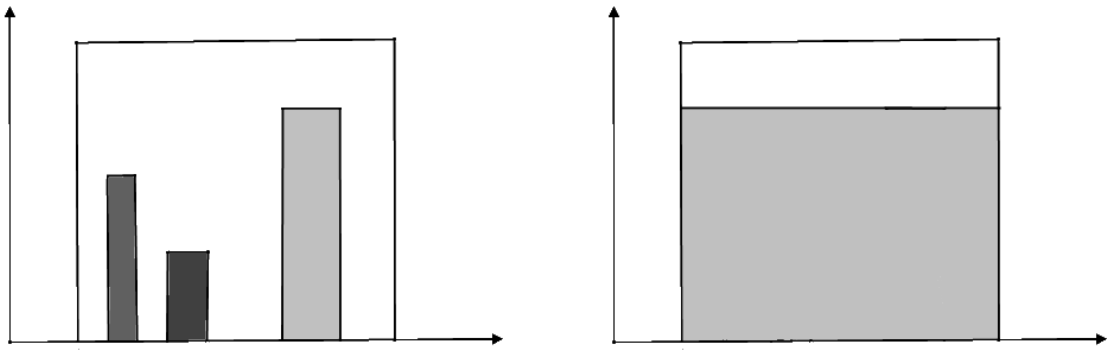
```
# L'image initiale est dans imbin1.
# Bouchage des trous dans imbin2.
closeHoles(imbin1, imbin2)
# Les trous sont récupérés dans imbin3.
diff(imbin2, imbin1, imbin3)
# Les trous sont dupliqués dans imbin4
copy(imbin3, imbin4)
# Les particules avec trous sont extraites dans imbin4 (après bouchage).
build(imbin2, imbin4)
# Le SKIZ géodésique des trous dans l'espace géodésique constitué des particules
# avec trous préalablement bouchées est effectué et stocké dans imbin3.
geodesicSKIZ(imbin3, imbin4, imbin3)
# Les frontières du SKIZ géodésique sont extraites dans imbin3.
diff(imbin4, imbin3, imbin3)
# Ces frontières marquent les particules avec au moins deux trous.
# Celles-ci sont reconstruites dans imbin3.
build(imbin1, imbin3)
```

Ces étapes sont illustrées ci-dessous.



De gauche à droite et de haut en bas: image initiale, image avec les trous bouchés, trous extraits, particules avec trous (trous bouchés), SKIZ géodésique des trous dans l'image précédente, frontières du SKIZ, reconstruction des particules avec au moins deux trous.

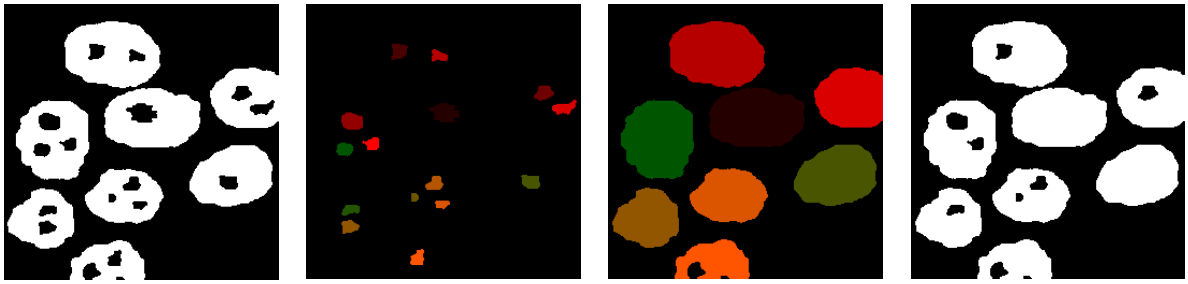
Utiliser la reconstruction géodésique pour réaliser cette séparation est moins évident. On peut néanmoins le faire en définissant un opérateur qui va boucher un et un seul trou dans toute composante connexe avec des trous. Cet opérateur ne fonctionne cependant qu'avec des particules ne présentant qu'un seul niveau dans l'arbre d'homotopie. Si ce n'est pas le cas, il faudra l'appliquer sur chaque niveau de l'arbre d'homotopie.
Cet opérateur est construit de la façon suivante:
- on commence par boucher tous les trous de l'ensemble de départ.
- ces trous sont extraits et labellisés. Chaque trou est donc affecté d'une valeur numérique entière unique.
- l'ensemble avec les trous bouchés de départ est converti en une fonction définie sur 32 bits avec 2 valeurs: 0 et $2^{32} - 1$.
- on reconstruit alors cette fonction avec l'image des trous labellisés. (voir figure explicative ci-dessous).



A gauche, une particule contient 3 trous avec trois labels differents. A droite, résultat de la reconstruction de l'image initiale (particule dont les trous ont été bouchés) par l'image label. Seul un trou a le même label. Les points ayant même valeur sur les deux images forment le masque de ce trou.

- chaque composante connexe de l'image reconstruite prendra donc la valeur de label d'un

seul trou qu'elle contient (c'est le label maximum de tous les trous contenus dans cette composante connexe).. Il suffit alors d'extraire ce trou grâce à l'opérateur ***generateSupMask*** qui génère un masque binaire correspondant à tous les pixels dont la valeur sur la première image est plus grande que ou égale à la valeur sur la seconde. L'union de ce masque avec l'image initiale permet de boucher un et un seul trou dans chaque composante connexe.



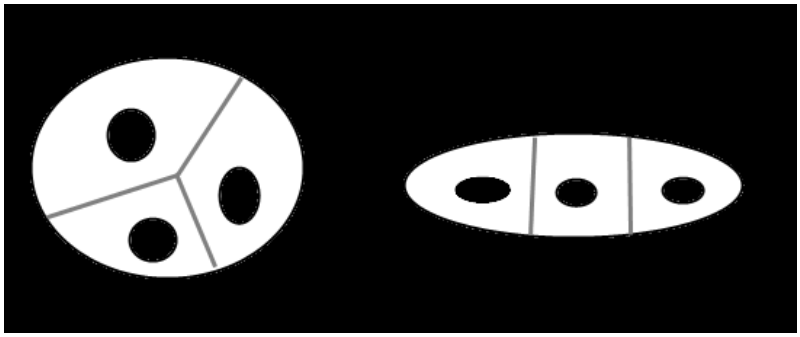
De gauche à droite: particules avec au moins un trou, image label des trous, reconstruction de l'image des particules bouchées avec l'image label. Cette reconstruction affecte à chaque particule le label d'un seul trou inclus dans la particule. L'extraction des points de valeur identique dans l'image label et dans la reconstruction fournit un masque des trous à supprimer. Le résultat du bouchage d'un unique trou par particule est fourni dans l'image de droite.

Cette procédure permettant de ne boucher qu'un seul trou par particule est décrite ci-dessous.

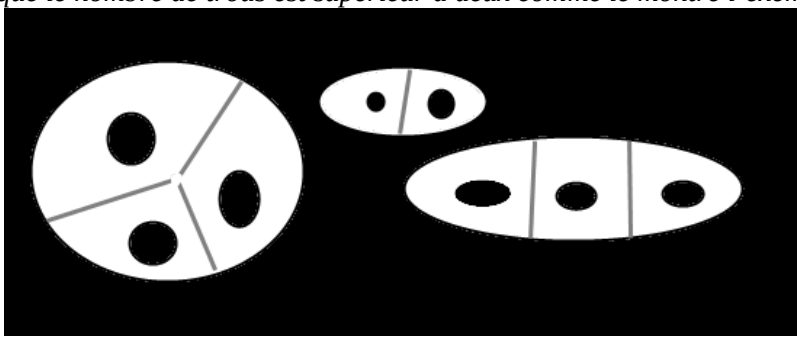
```
def closeOneHole(imIn, imOut):
    """
    This procedure allows to close one and only one hole in each connected component
    of the binary image imIn. When a connected component has no hole, it remains
unchanged in the final image stored in imOut.
    This algorithm requires a single level of homotopy in the initial image.

    imWrk0 = imageMb(imIn, 32)
    imWrk1 = imageMb(imIn, 32)
    imWrk2 = imageMb(imIn, 32)
    imWrk3 = imageMb(imIn)
    imWrk4 = imageMb(imIn)
    # The holes are filled in imWrk3 and extracted in imWrk4.
    closeHoles(imIn, imWrk3)
    diff(imWrk3, imIn, imWrk4)
    # The holes are labelled.
    nb = label(imWrk4, imWrk0)
    # The image with filled holes is converted in 32-bit.
    convertByMask(imWrk3, imWrk1, 0, computeMaxRange(imWrk1)[1])
    # Geodesic reconstruction of the label image.
    copy(imWrk0, imWrk2)
    build(imWrk1, imWrk2)
    # The holes with same label as the built image are extracted...
    generateSupMask(imWrk0, imWrk2, imWrk4, False)
    logic(imWrk4, imWrk3, imWrk3, "inf")
    # ... and filled.
    logic(imIn, imWrk3, imOut, "sup")
```

(c) Extraction des particules avec deux trous seulement

La séparation des particules à deux trous de celles qui en possèdent plus est encore plus complexe lorsqu'on utilise l'approche basée sur les opérateurs de squelettisation. On peut cependant définir une solution basée sur le squelette par zone d'influence géodésique que nous avons déjà utilisé à l'étape précédente. Considérons l'ensemble constitué de particules avec deux trous ou plus. Bouchons tous les trous et effectuons le SKIZ géodésique des trous dans l'espace géodésique constitué par cet ensemble sans trou. Le résultat obtenu sur l'image *gruyere* nous incite à utiliser les points multiples du SKIZ géodésique comme marqueurs des particules à plus de deux trous. Cependant, ce serait une erreur! En effet, comme le montre l'exemple suivant, le SKIZ geodésique des trous peut très bien ne pas faire apparaître de point multiple lorsqu'une composante connexe a trois trous. Mais, après élimination des éventuels points multiples, on constate que seules les particules initialement avec deux trous, possèdent un et un seul arc dans le SKIZ géodésique. Il suffit alors d'effectuer le SKIZ géodésique des arcs précédents. Lorsqu'un arc unique est présent dans une composante connexe, le SKIZ géodésique est vide. En revanche, ce n'est pas le cas lorsqu'il y a plus de deux trous. Ce dernier SKIZ géodésique peut être utilisé comme marqueur pour reconstruire ces particules avec un nombre de trous supérieur à deux. Notons également que cet algorithme ne fonctionne pas lorsque la structure des trous est complexe et que l'arbre d'homotopie contient plus d'une ramification.



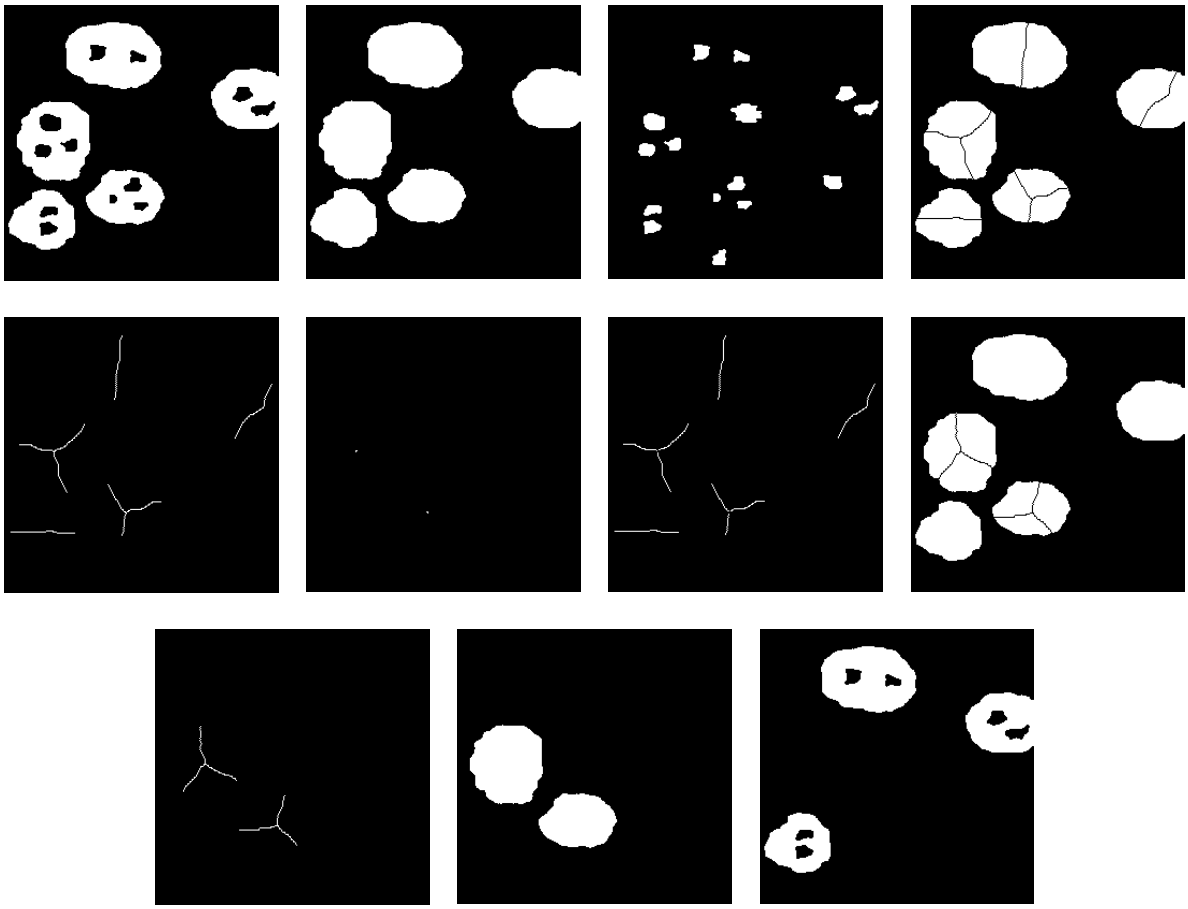
Le SKIZ géodésique des trous dans les particules ne fait pas nécessairement apparaître des points multiples lorsque le nombre de trous est supérieur à deux comme le montre l'exemple ci-dessus.



La suppression des éventuels points multiples permet de marquer les particules avec un certain nombre d'arcs simples. Seules les particules avec deux trous sont marquées par un arc unique.

On trouvera ci-dessous la suite des opérations à effectuer pour obtenir le résultat souhaité. Rappelons que cette procédure utilise l'image des particules avec au moins deux trous comme image initiale.

```
# L'image initiale est dans imbin1.
# Les trous sont bouchés dans imbin2.
closeHoles(imbin1, imbin2)
# Les trous sont stockés dans imbin3.
diff(imbin2, imbin1, imbin3)
# SKIZ géodésique des trous dans l'espace géodésique formé par les particules sans trous.
# Le SKIZ est placé dans imbin3.
geodesicSKIZ(imbin3, imbin2, imbin4)
diff(imbin2, imbin4, imbin3)
# Les éventuels points multiples du SKIZ géodésique sont extraits et supprimés.
# imbin4 contient uniquement les éléments d'arc simples.
multiplePoints(imbin3, imbin4)
diff(imbin3, imbin4, imbin4)
# On effectue le SKIZ géodésique de ces arcs dans imbin2. Résultat dans imbin3.
geodesicSKIZ(imbin4, imbin2, imbin3)
diff(imbin2, imbin3, imbin3)
# Les particules marquées par ce deuxième SKIZ sont reconstruites.
build(imbin2, imbin3)
# Les particules avec deux trous sont dans imbin3.
diff(imbin1, imbin3, imbin3)
```



De gauche à droite et de haut en bas: image initiale (particules avec au moins 2 trous), particules bouchées, trous, SKIZ géodésique des trous, frontières du SKIZ, points multiples, arcs simples du SKIZ, deuxième SKIZ géosésique de ces arcs simples, frontières du deuxiéme SKIZ, particules reconstruites (particules bouchées avec plus de 2 trous), résultat final (particules avec 2 trous).

L'approche basée sur les reconstructions est, a contrario, aisée puisqu'il suffit d'itérer deux

fois l'algorithme permettant de boucher un seul trou dans une particule. Les particules sans trou à l'issue de cette itération sont celles qui en possédaient deux initialement.

(d) Conception d'un algorithme pour extraire toute particule avec au plus n trous

Rappelons qu'un tel algorithme existe déjà. Il a été présenté au début de ce corrigé et consiste simplement à utiliser l'analyse individuelle et la mesure du nombre de connexité de chaque particule. Cet algorithme est cependant très lent. On peut donc rechercher des solutions plus rapides et plus efficaces.

Une solution simple consiste à itérer n fois l'algorithme permettant de boucher un seul trou dans une particule et à extraire les particules sans trou à la fin de cette boucle. La procédure suivante réalise ce filtrage (plus exactement, elle fait l'inverse en éliminant les particules qui ont au plus n trous).

```
def holesFiltering(imIn, imOut, n):
    """
    Removes from image imIn all the particles which contain up to n holes and
    puts the result in imOut. imOut contains particles with more than n holes.
    The homotopy tree of imIn is supposed to be simple (one level only).
    n must be positive or equal to zero..
    """

    imWrk1 = imageMb(imIn)
    imWrk2 = imageMb(imIn)
    copy(imIn, imWrk1)
    i = n
    while i>0:
        closeOneHole(imWrk1, imWrk1)
        i -= 1
    closeHoles(imWrk1, imWrk2)
    diff(imWrk2, imWrk1, imWrk2)
    dilate(imWrk2, imWrk2)
    build(imIn, imWrk2)
    copy(imWrk2, imOut)
```

La vitesse de cette procédure ne dépend pas du nombre de particules mais du nombre de trous présents dans chaque particule. De plus, cette procédure ne fonctionne que sur des ensembles sans imbrication des composantes connexes et des trous. Si ce n'est pas le cas, il faudra décomposer l'ensemble en ses différents niveaux d'homotopie et appliquer la procédure sur chacun d'entre eux.

Il existe cependant une procédure qui fonctionne relativement vite et pour tous les types d'ensembles même ceux présentant un arbre d'homotopie des plus complexes. Je vais présenter succinctement cette approche, en indiquant le principe de son fonctionnement dans les grandes lignes. Cet algorithme (et d'autres) fera l'objet d'une prochaine publication sur ma page Web (http://cmm.ensmp.fr/~beucher/publi.html). Les plus curieux pourront d'y référer.

Cet algorithme suit le même schéma que l'algorithme permettant de labelliser chaque composante connexe d'un ensemble par sa surface afin de réaliser rapidement un opérateur d'ouverture surfacique (opérateur décrit dans les exemples sur le site Web de Mamba). Il

consiste à labelliser rapidement chaque composante connexe d'un ensemble avec le nombre de points appartenant à un autre ensemble et qui sont inclus dans cette composante connexe. Considérons un ensemble X formé de plusieurs composantes connexes X_i et un autre ensemble Y (pas nécessairement inclus dans X). Certains points de Y peuvent être contenus dans X_i et ils forment alors un sous-ensemble Y_i. Comment labelliser chaque composante X_i avec le nombre de points contenus dans Y_i et cela le plus rapidement possible? Pour ce faire, nous allons utiliser plusieurs opérateurs rapides de la librairie Mamba: ***label***, ***getHistogram*** et ***lookup***:

- on commence par labelliser l'ensemble X. Chaque composante connexe X_i prend une valeur unique l_i.
- l'infimum entre l'image label et Y converti en une image numérique est effectué. Chaque point de Y_i se verra alors affecter une valeur l_i.
- on calcule alors l'histogramme h de cette nouvelle image label. La valeur de l'histogramme à chaque indice l_i sera donc égale aux nombre de points de Y_i, noté $n(Y_i)$.
- il suffit maintenant d'utiliser l'opérateur ***lookup***. Cet opérateur utilise une table de correspondance t pour remplacer chaque valeur de gris j d'une image par la valeur correspondante t(j) dans la table. Cet opérateur ainsi que la labellisation et le calcul de l'histogramme sont très rapides car ils nécessitent un seul balayage de l'image. On va donc construire la table de correspondance t suivante:

$$t(l_i) = h(l_i) = n(Y_i)$$

L'application de l'opérateur ***lookup*** utilisant cette table t sur l'image label de départ (image label de X) va donc affecter à chaque composante connexe de X un nouveau label égal au nombre de points de Y tombant dans cette composante connexe.

Pour labelliser chaque composante connexe d'un ensemble par sa surface, il suffit de faire Y=X.

En pratique, cet opérateur est un peu plus complexe car les opérateurs ***getHistogram*** et ***lookup*** ne fonctionnent que sur des images 8-bit alors que l'image label peut exhiber un nombre de valeurs supérieur à 255 si le nombre de composantes connexes de X est lui-même supérieur à 255. Pour résoudre ce problème, l'image label est traité par tranches successives de 255 valeurs, ce qui ralentit légérement le processus lorsque le nombre de composantes connexes est supérieur à 255 (il s'effectue néanmoins 255 fois plus vite que l'approche par analyse individuelle!).

Cette procédure de labellisation intitulée ***measureLabelling*** réalisée avec la librairie Mamba est fournie ci-après.

```
# The measure labelling procedure is defined.
def measureLabelling(imIn, imMeasure, imOut):
    """
    Labelling each particle of the binary image 'imIn' with the number of pixels
    in image 'imMeasure' contained in each particle. The result is put is the 32-bit
    image 'imOut'.
    """

    # Working images.
    imWrk1 = imageMb(imIn, 32)
    imWrk2 = imageMb(imIn)
    imWrk3 = imageMb(imIn, 8)
    imWrk4 = imageMb(imIn, 8)
    imWrk5 = imageMb(imIn, 8)
```

```
imWrk6 = imageMb(imIn, 32)

# Output image is emptied.
imOut.reset()
# Labelling the initial image.
nbParticles = label(imIn, imWrk1)
# Defining output LUTs.
outLuts = [[0 for i in range(256)] for i in range(4)]
# Convering the imMeasure image to 8-bit.
convert(imMeasure, imWrk4)
while nbParticles > 0:
    # particles with labels between 1 and 255 are extracted.
    threshold(imWrk1, imWrk2, 0, 255)
    convert(imWrk2, imWrk3)
    copyBytePlane(imWrk1, 0, imWrk5)
    logic(imWrk3, imWrk5, imWrk3, "inf")
    # The points contained in each particle are labelled.
    logic(imWrk3, imWrk4, imWrk5, "inf")
    # The histogram is computed.
    histo = getHistogram(imWrk5)
    # The same operation is performed for the 255 particles.
    for i in range(1, 256):
        # The number of points in each particle is obtained from the histogram.
        value = histo[i]
        j = 3
        # This value is splitted in powers of 256 and stored in the four
        # output LUTs.
        while j >= 0:
            n = 2 ** (8 * j)
            outLuts[j][i] = value / n
            value = value % n
            j -= 1
    # each LUT is used to label each byte plane of a temporary image with the
    # corresponding value.
    for i in range(4):
        lookup(imWrk3, imWrk5, outLuts[i])
        copyBytePlane(imWrk5, i, imWrk6)
    # The intermediary result is accumulated in the final image.
    logic(imOut, imWrk6, imOut, "sup")
    # 255 is subtracted from the initial labelled image in order to process
    # the next 255 particles. Subtracting 256 (instead of 255) comes from the fact
    # that the label operator does not use label values which are multiple of 256.
    floorSubConst(imWrk1, 256, imWrk1)
    nbParticles -= 255
```

A l'aide de cette procédure, nous allons labelliser chaque composante connexe d'un ensemble avec le nombre de trous que contient cette composante. On sait que le nombre de trous n_T d'une composante connexe est relié à son nombre de connexité ν par la relation suivante:

$$n_T = 1 - \nu$$

Afin d'éviter de labelliser les composantes connexes sans trou avec la valeur 0, la labellisation se fera avec la valeur $n_T + 1 = 2 - \nu$.
En trame hexagonale (je laisse au lecteur le soin de transposer cet algorithme en trame carrée s'il le souhaite), le nombre de connexité s'obtient en dénombrant le nombre de configurations

$\begin{matrix} 0 & 0 \\ 1 & \end{matrix}$ et $\begin{matrix} & 0 \\ 1 & 1 \end{matrix}$ présentes dans la composante connexe selon la formule:

$$\nu = n\begin{pmatrix} 0 & 0 \\ & 1 \end{pmatrix} - n\begin{pmatrix} & 0 & \\ 1 & & 1 \end{pmatrix} = n_1 - n_2$$

On peut mettre en évidence les points de chaque composante connexe présentant ces deux configurations à l'aide d'une transformée en tout-ou-rien. Les deux ensembles obtenus avec ces deux transformées en tout-ou-rien peuvent être utilisés pour labelliser l'ensemble X de départ avec les valeurs n_1 et n_2 correpondant à chaque composante connexe à l'aide de l'opérateur ***measureLabelling*** décrit plus haut. Il suffit alors de soustraire ces deux images label et d'ajouter la valeur 2 à chaque valeur non nulle pour obtenir une nouvelle image label où chaque composante connexe prendra une valeur égale au nombre de trous qu'elle contient augmenté de 1:

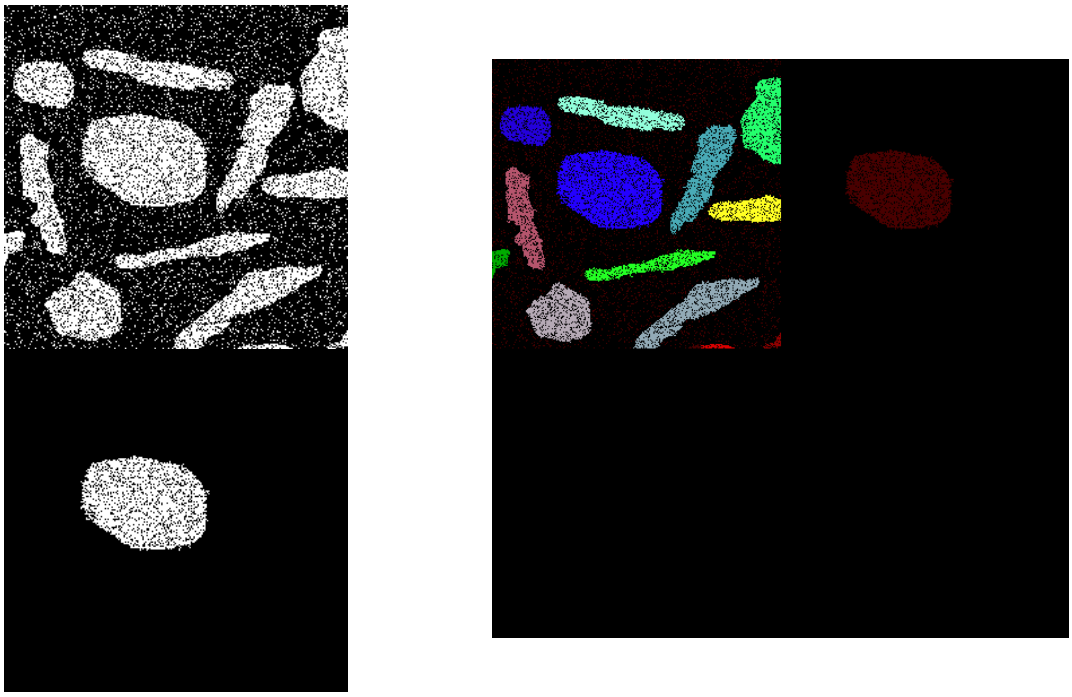
$$n_T + 1 = 2 + n_2 - n_1$$

La procédure complète est donnée ci-dessous.

```
# Labelling each particle with its number of holes (up to 1).
def holesLabelling(imIn, imOut):
    """
    Labels each particle in imIn with a value equal to its number of holes +1.
    The result is put in 32-bit image imOut.
    """

    # Working images.
    imWrk1 = imageMb(imIn)
    imWrk2 = imageMb(imIn, 32)

    # Initializing the label image with 2.
    convertByMask(imIn, imOut, 0, 2)
    # Determining the 2nd configurations in the connectivity number calculation.
    # and adding their number to the label image.
    hitOrMiss(imIn, imWrk1, 2, 5)
    measureLabelling(imIn, imWrk1, imWrk2)
    add(imOut, imWrk2, imOut)
    # Determining the 1st configurations and subtracting them to get the
    # number of holes + 1.
    hitOrMiss(imIn, imWrk1, 66, 1)
    measureLabelling(imIn, imWrk1, imWrk2)
    sub(imOut, imWrk2, imOut)
```

On vérifiera que les éléments structurants programmés dans les opérateurs hitOrMiss correspondent bien aux configurations recherchées. On remarquera également que l'origine des éléments structurants a été choisie de façon judicieuse pour que le résultat de la transformée soit toujours inclus dans l'ensemble de départ.

On peut, à titre d'illustration, appliquer cette procédure à l'image *noise*:



En haut à gauche, l'image noise initiale. A droite, la labellisation des composantes connexes par leur nombre de trous + 1 (les quatre quadrants de l'image label sont représentées). En bas à gauche, une composante connexe de l'image initiale. Son label dans l'image label est égal à 485. Son nombre de connexité mesuré directement est égal à -483. Cette composante connexe a donc bien 484 trous.

(e) La transformation qui ne conserve que les particules ayant plus de n trous est-elle une ouverture algébrique?

Cette transformation est bien entendu anti-extensive (il n'y a pas plus de particules dans l'ensemble transformé que dans l'ensemble initial) et elle est idempotente (toutes les composantes connexes de $\Psi_n(X)$ ont plus de n trous et aucune n'est éliminée en appliquant à nouveau l'opérateur). En revanche, cette transformation n'est pas croissante. En effet considérons une composante connexe X_1 contenant n+1 trous et une composante connexe X_2 la contenant mais sans trou (il suffit de prendre pour X_2 la particule X_1 dont on a bouché les trous). On a alors:

$$X_1 \subset X_2$$
$$\psi(X_1) = X_1 \text{ et } \psi(X_2) = \varnothing$$

On a donc:

$$\psi(X_1) \supset \psi(X_2)$$

La transformation n'est pas une ouverture.

Maintenant, tout est question de point de vue... Si je considère que je travaille non plus dans un espace de points mais dans l'espace où chaque élément est une composante connexe, un sous-ensemble X de cet espace est composé par une suite non ordonnée de composantes connexes X_i. Un ensemble X_1 de cet espace est inclus dans un ensemble X_2 si et seulement si toutes les composantes connexes qui appartiennent à X_1 appartiennent également à X_2. Dans cette espace (infini) formé de toutes les composantes connexes possibles, la transformation ψ est croissante, anti-extensive et idempotente. C'est donc une ouverture...

En conclusion...
Ce problème de tri de composantes connexes en fonction de leur nombre de trous, si on veut en donner une solution aussi exhaustive que possible, conduit à utiliser un nombre important d'opérateurs morphologiques: opérateurs géodésiques, amincissements, opérateurs homotopiques, lignes de partage des eaux, etc. Des concepts topologiques doivent également être introduits. On est aussi confronté à divers problèmes relatifs aux effets de bord et à certaines caractéristiques des opérateurs utilisés. La recherche de solutions à la fois élégantes, rapides et exactes n'est pas aisée. C'est la raison pour laquelle j'ai tenu à donner un corrigé de ces exercices le plus complet possible. Je suis néanmoins conscient que toutes ces difficultés ne peuvent pas être surmontées pendant le temps qui était imparti pour résoudre ces exercices.

L'ensemble des procédures introduites dans ce corrigé ont été regroupées dans un module Python intitulé **holes_handling.py** joint à ce corrigé. Attention, ces procédures ne fonctionnent qu'en trame hexagonale...