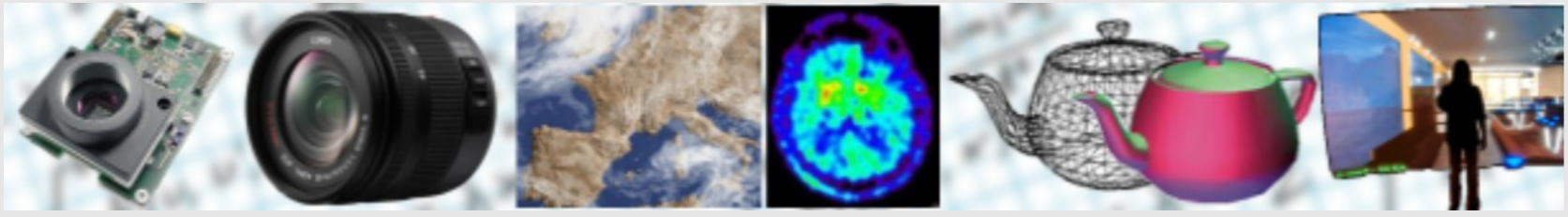


PjE – Projet Encadré IVI
Semaine 5 : analyse en composantes connexes

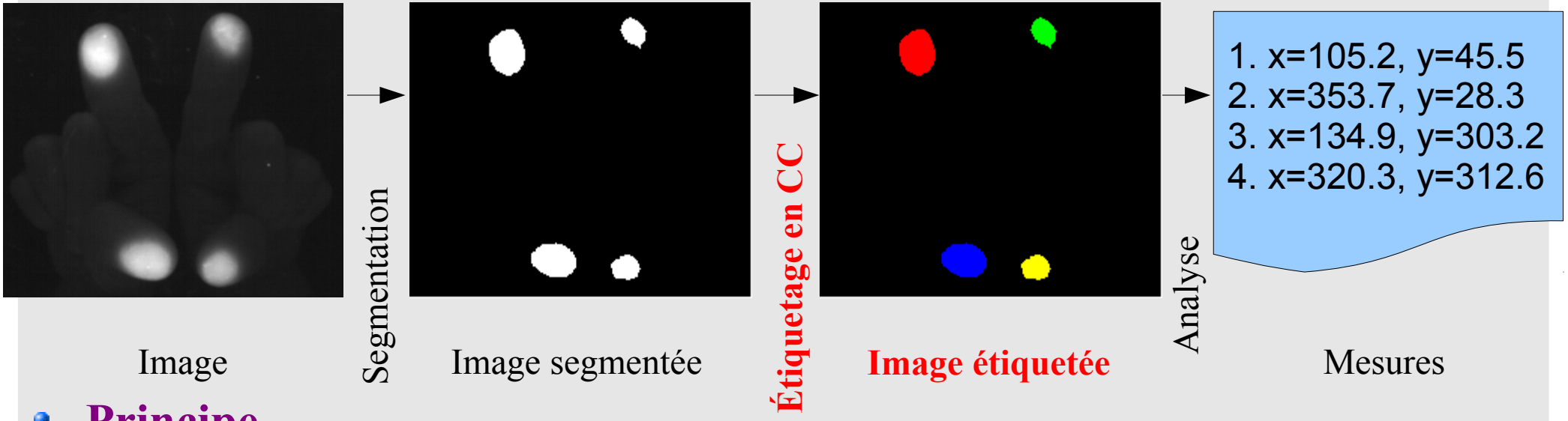
Master ASE : http://master-ase.univ-lille1.fr
Master Informatique : http://www.fil.univ-lille1.fr
Spécialité IVI : http://master-ivi.univ-lille1.fr

Plan du cours

- **1 – Analyse en composantes connexes**
  - **Définitions et principe**
    - composantes connexes
    - étiquetage
  - **Approche par remplissage (*flood-fill*)**
  - **Approche par double parcours (*two-pass*)**
- **2 – Bibliothèque cvBlobsLib**
  - **Présentation**
  - **Classes principales**

Présentation générale (1/3)

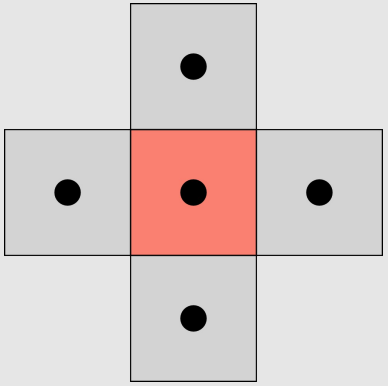
- **Cadre**



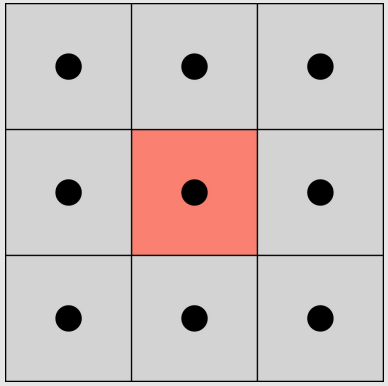
- **Principe**
  - **Partant d'une image binaire, trouver les groupes de pixels connectés, appelés composantes connexes (*connected components*) ou *blobs***
  - **On obtient une image dans laquelle chaque « objet » est identifié**
  - **Cette opération s'appelle analyse (ou étiquetage) en composantes connexes (*connected-component analysis* / *labeling*, ou encore *blob extraction*)**

Présentation générale (2/3)

- **Connexité**
  - **Pixels « connectés » ?**



4-connexité



8-connexité

***Rem.*: pour les régions, on choisit la 4-connexité**

  - **Exemples de composantes connexes**

0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	1	1	1
0	1	1	0	0	1	0	1	1
0	1	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Image binaire

0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	1	1	1
0	1	1	0	0	1	0	1	1
0	1	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Composantes 4-connexes

0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	1	1	1
0	1	1	0	0	1	0	1	1
0	1	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Composantes 8-connexes

Présentation générale (3/3)

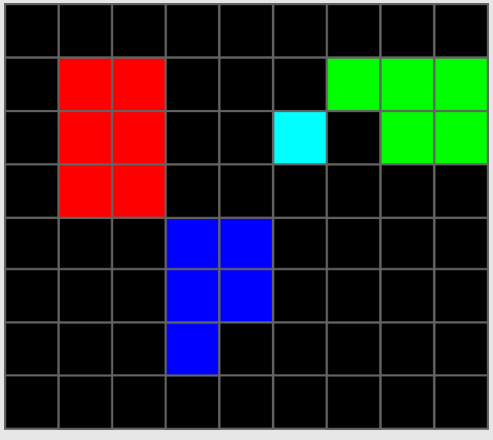
- **Étiquetage**
  - **Chaque composante connexe est identifiée de manière unique par une étiquette (*label*)**

0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	1	1	1
0	1	1	0	0	1	0	1	1
0	1	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0

  - Étiquettes numériques (entiers)

0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	2	2	2
0	1	1	0	0	3	0	2	2
0	1	1	0	0	0	0	0	0
0	0	0	4	4	0	0	0	0
0	0	0	4	4	0	0	0	0
0	0	0	4	0	0	0	0	0
0	0	0	0	0	0	0	0	0

  - Étiquettes de couleurs



Approche par remplissage (1/4)

- **Principe de l'approche par remplissage (*flood fill*)**
  - **E/S**
    - Paramètre d'entrée : image binaire ***B***
    - Résultat de sortie : matrice (ou *carte*) d'étiquettes ***L***
  - **Algorithme**
    - Créer une matrice d'étiquettes ***L***, de même taille que ***B***, initialisées à 0
    - Initialiser le compteur d'étiquettes *nbLabels* à 0
    - Tant qu'il y a des pixels à 1 dans ***B***
      - Trouver le prochain pixel $P(x,y)$ à 1 dans ***B***
      - Incrémenter *nbLabels* et donner à $\boldsymbol{L}(x,y)$ la valeur *nbLabels*
      - Mettre $\boldsymbol{B}(x,y)$ à 0
      - Traiter de la même manière les pixels 4-connexes avec P et à 1 dans ***B***, jusqu'à ce qu'il n'y en ait plus aucun

Approche par remplissage (2/4)

Exemple (1/3)

- Initialisation
  nbLabels=0

B

0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	0	1	1	1	0
0	1	0	0	1	0	0
0	0	0	0	0	0	0

L

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

- Étape 1
  nbLabels=1
  mettre à *nbLabels* l'élément de ***L*** correspondant au premier pixel à 1 dans ***B***

B

0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	0	1	1	1	0
0	1	0	0	1	0	0
0	0	0	0	0	0	0

L

0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

- Étape 2
  mettre à 0 le pixel qui vient d'être étiqueté et trouver ses voisins 4-connexes qui sont à 1

B

0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	1	1	0
0	1	0	0	1	0	0
0	0	0	0	0	0	0

L

0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

- Étape 3
  idem avec les nouveaux pixels

B

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	1	1	0
0	1	0	0	1	0	0
0	0	0	0	0	0	0

L

0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Approche par remplissage (3/4)

Exemple (2/3)

- Étape 4
  idem avec les nouveaux pixels
  (*il n'y a plus de voisin ici*)

- Étape 5
  trouver un autre pixel à 1 dans ***B***
  nbLabels=2
  mettre à *nbLabels* l'élément
  de ***L*** correspondant au pixel

- Étape 6
  mettre à 0 le pixel qui vient
  d'être étiqueté et trouver ses
  voisins 4-connexes qui sont à 1

- Étape 7
  idem avec les nouveaux pixels

B ***L***

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	1	1	0
0	0	0	0	1	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	1	1	0
0	0	0	0	1	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	0	2	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	1	1	0
0	0	0	0	1	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	0	2	2	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	0	2	2	2	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

Approche par remplissage (4/4)

Exemple (3/3)

- Étape 8
  idem avec les nouveaux pixels
  (*il n'y a plus de voisin ici*)

B

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

L

0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	0	2	2	2	0
0	1	0	0	2	0	0
0	0	0	0	0	0	0

- Fin (tous les pixels de ***B*** sont à 0)

Approche par double parcours (1/7)

- **Principe de l'approche par double parcours (*two-pass*)**
  - **E/S**
    - Paramètre d'entrée : image binaire ***B***
    - Résultat de sortie : matrice (ou *carte*) d'étiquettes ***L***
  - **Algorithme**
    - Premier parcours de l'image, dans le sens classique (*raster-scan order*) :
      À chaque pixel à 1 dans ***B***, on affecte
      - la plus petite étiquette parmi celles de ses voisins **haut** et **gauche**
        ou
      - une nouvelle étiquette si aucun de ces 2 voisins n'est encore étiqueté
    - Second parcours de l'image, dans le sens inverse :
      À chaque pixel précédemment étiqueté, on affecte la plus petite étiquette parmi la sienne et celles de ses voisins **bas** et **droite**.

Approche par double parcours (2/7)

Exemple-1er parcours (1/3)

- Initialisation
  nbLabels=0

B

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

L

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

- Étape 1
  les voisins haut et gauche du premier pixel à 1 dans ***B*** ne sont pas encore étiquetés
  ⇨ nouvelle étiquette (*nbLabels*=1)

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

- Étape 2
  le voisin gauche du pixel suivant à 1 dans ***B*** est déjà étiqueté à 1 ⇨ affecter cette même étiquette au pixel

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

- Étape 3
  nouvelle étiquette (*nbLabels*=2)

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Approche par double parcours (3/7)

Exemple-1er parcours (2/3)

- Étape 4
  affecter au pixel l'étiquette de son voisin gauche

- Étape 5
  nouvelle étiquette (*nbLabels*=3)

- Étape 6
  les voisins haut et gauche du pixel suivant à 1 dans ***B*** sont étiquetés différemment ⇨ affecter au pixel l'étiquette minimale

- Étape 7
  affecter au pixel l'étiquette de son voisin haut

B

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

L

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	2
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Approche par double parcours (4/7)

Exemple-1er parcours (3/3)

- **Étape 8**
  affecter au pixel l'étiquette de son voisin haut

B

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

L

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	2
0	3	0	0	0	0	0
0	0	0	0	0	0	0

- **Étape 9**
  nouvelle étiquette (*nbLabels*=4)

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	2
0	3	0	0	4	0	0
0	0	0	0	0	0	0

- **Étape 10**
  affecter au pixel l'étiquette de son voisin gauche

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	2
0	3	0	0	4	4	0
0	0	0	0	0	0	0

- **Étape 11**
  les voisins haut et gauche du pixel suivant à 1 dans ***B*** sont étiquetés différemment ⇨ affecter au pixel l'étiquette minimale

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	2
0	3	0	0	4	4	2
0	0	0	0	0	0	0

Approche par double parcours (5/7)

Exemple-2ème parcours (1/2)

- Image et carte initiales

B

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

L

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	2
0	3	0	0	4	4	2
0	0	0	0	0	0	0

- *1ère étiquette modifiée*
  Affecter au pixel l'étiquette de son voisin droit car elle est inférieure à la sienne

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	2
0	3	0	0	4	2	2
0	0	0	0	0	0	0

- *2ème étiquette modifiée*
  Affecter au pixel l'étiquette de son voisin droit car elle est inférieure à la sienne

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	2
0	3	0	0	2	2	2
0	0	0	0	0	0	0

- (*étiquette non modifiée*)

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	2
0	3	0	0	2	2	2
0	0	0	0	0	0	0

Approche par double parcours (6/7)

- **Exemple-2ème parcours (2/2)**
  - (*étiquette non modifiée*)

B

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

L

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	2
0	3	0	0	2	2	2
0	0	0	0	0	0	0

  - *3ème étiquette modifiée*
    Affecter au pixel l'étiquette de son voisin droit car elle est inférieure à la sienne et à celle de son voisin bas

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	1	1	0	0	0	2
0	3	0	0	2	2	2
0	0	0	0	0	0	0

  - Carte à l'issue du 2ème parcours
    Aucune autre étiquette n'est plus modifiée

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	1	1	0	0	0	2
0	3	0	0	2	2	2
0	0	0	0	0	0	0

  - ***Problème*** (*collisions*) : d'autres parcours (jusqu'à ce qu'il n'y ait plus de changement) sont nécessaires pour obtenir la carte finale. *Ex*. après 3ème parcours en sens classique :

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	1	1	0	0	0	2
0	1	0	0	2	2	2
0	0	0	0	0	0	0

Approche par double parcours (7/7)

Utilisation d'une table d'équivalences

Principe

- Lors du premier parcours, si 2 voisins portent des étiquettes différentes l_i et l_j, choisir l'une d'entre elles pour le pixel et mémoriser l'équivalence $l_i \equiv l_j$
- Lors du second parcours, réétiqueter les pixels selon la table d'équivalences (en renumérotant éventuellement les étiquettes pour qu'elles soient consécutives)

Exemple

- Étape 6 du 1er parcours
  mémoriser l'équivalence 1 ≡ 3
- Étape 11 du 1er parcours
  mémoriser l'équivalence 2 ≡ 4

Le second parcours suffit ensuite à obtenir la carte finale

B

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

L

0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	0	0	0	1
0	1	0	0	1	1	1
0	0	0	0	0	0	0

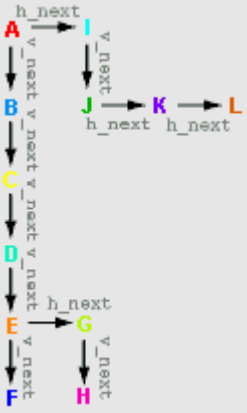
0	0	0	0	0	0	0
0	0	1	1	0	2	2
0	3	1	0	0	0	2
0	3	0	0	4	4	2
0	0	0	0	0	0	0

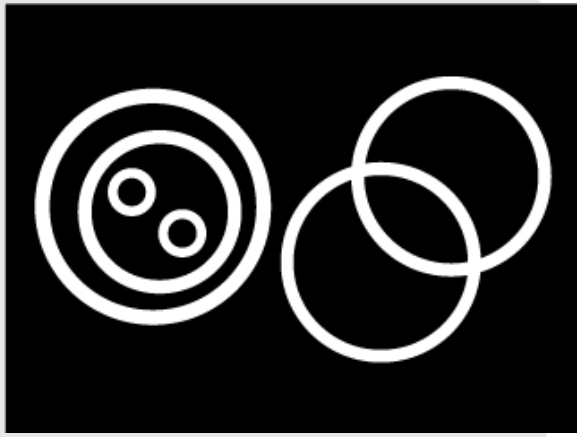
Composantes connexes sous OpenCV

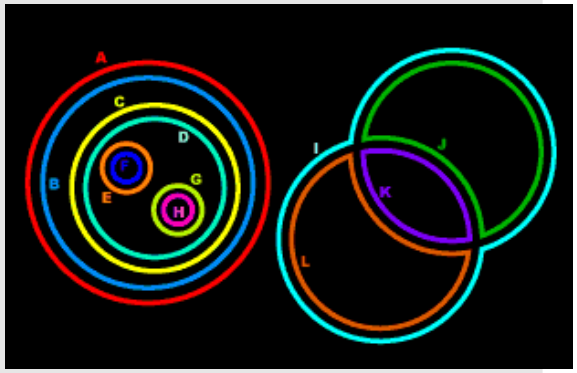
- **Séquence**
  - **Définition : liste chaînée d'autres structures**
  - **Structure**

```
typedef struct CvSeq {
  int      total;  // nombre total d'éléments
  int      elem_size;  // taille d'un élément (en octets)
  CvSeq*   h_prev;  // séquence précédente
  CvSeq*   h_next;  // séquence suivante
  CvSeq*   v_next;  // seconde séquence suivante
  ...
};
```

  - **Principale utilité : stockage de contours**







- **Composante connexe sous OpenCV**
  - **Structure**

http://jmpelletier.com/a-simple-opencv-tutorial

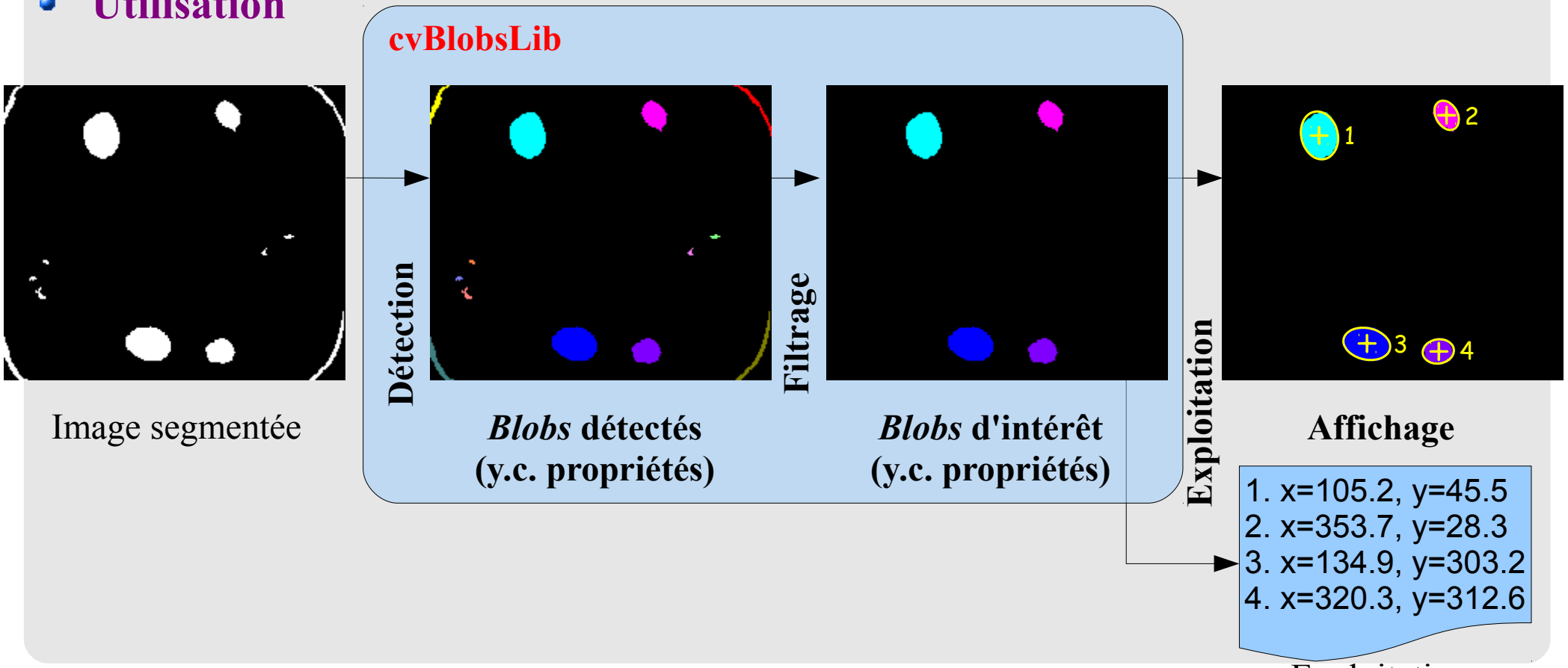
```
typedef struct CvConnectedComponent {
  double      area;  // surface
  CvScalar    value;  // couleur moyenne
  CvRect      rect;  // boîte englobante (bounding box)
  CvSeq*      contour;  // séquence (de CvPoint) stockant le contour de la composante
};
```

Présentation de cvBlobsLib (1/2)

- **Présentation**
  - **Basée sur l'API d'OpenCV**
  - **Également *open source* et multi-plateformes**
  - **Offre une API de haut niveau pour la détection des objets (« *blobs* »)**
  - **La détection des objets est basée sur les discontinuités dans l'image**
  - **Développée en C++ (sous VC++6, mais disponible pour .NET et Linux)**
- **Fonctionnalités**
  - **Extraction des composantes connexes dans une image binaire (ou en niveaux de gris). Ces composantes sont désignées par le terme « *blobs* »**
  - **Filtrage des *blobs* obtenus pour ne retenir que les *blobs* réellement intéressants de l'image**
- **Mise en œuvre**
  - **Dépôt web : http://opencv.willowgarage.com/wiki/cvBlobsLib**
  - **Intégrer au projet les sources de la bibliothèque**

Présentation de cvBlobsLib (2/2)

- **E/S**
  - **Entrée : image binaire (image pré-traitée et segmentée)**
  - **Sortie : ensemble des *blobs* détectés dans l'image, avec leurs propriétés**
- **Utilisation**



Classes principales de cvBlobsLib (1/4)

- **CBlobResult (1/3) : Ensemble de *blobs* extraits d'une image**
  - **Extraction des *blobs* d'une image**

```
CBlobResult::CBlobResult(
    IplImage* source,
    IplImage* mask,
    uchar backgroundColor
)
```

    Forme les *blobs* à partir des composantes connexes de l'image `source`, *i.e.* des pixels connexes de niveaux différents de `backgroundColor` (souvent : 0)
  - **Calcul du nombre de *blobs* extraits**

```
int CBlobResult::GetNumBlobs( )
```

  - **Accès à un *blob* donné par son indice**

```
CBlob CBlobResult::GetBlob( int indexblob )
```

  - **Suppression de tous les *blobs* extraits**

```
void CBlobResult::ClearBlobs( )
```

Classes principales de cvBlobsLib (2/4)

CBlobResult (2/3)

Filtrage des *blobs* selon certains critères

```
void CBlobResult::Filter(
    CBlobResult & dst,
    int filterAction,
    COperadorBlob* evaluador,
    int condition,
    double lowLimit,
    double highLimit = 0
)
```

- Le critère de sélection est défini par
  - l'opérateur `evaluador`, défini par une classe dérivée de **`COperadorBlob`**, par *ex*. **`CBlobGetArea()`**
  - la `condition` ∈ { **`B_EQUAL`**, **`B_NOT_EQUAL`**, **`B_GREATER`**, **`B_LESS`**, **`B_GREATER_OR_EQUAL`**, **`B_LESS_OR_EQUAL`**, **`B_INSIDE`**, **`B_OUTSIDE`** }
  - la ou les limite(s) inférieure `lowLimit` et supérieure `highLimit`.
- Si `filterAction` = **`B_INCLUDE`**, seuls les *blobs* de `dst` répondant au critère sont conservés. À l'inverse, si `filterAction` = **`B_EXCLUDE`**, les *blobs* de `dst` répondant au critère sont éliminés.

Classes principales de cvBlobsLib (3/4)

- **CBlobResult (3/3)**
  - **Calcul de certaines propriétés des *blobs***

```
double CBlobResult::GetNumber(
    int indexblob,
    COperadorBlob* evaluador
)
```

Évalue, sur le *blob* d'indice `indexblob`, une propriété (selon l' « évaluateur » utilisé `evaluador`, de classe dérivée de **`COperadorBlob`**, *ex.* **`CBlobGetArea()`**)

- **Cblob : Un *blob* extrait d'une image**
  - **Plusieurs méthodes permettent de calculer les propriétés du *blob*, *ex.:***
    - `double Area( )`
    - `double Perimeter( )`
    - `CvBox2D GetEllipse( )` // *ellipse englobante*
    - ...
  - Équivalence entre

`blobs.GetNumber(i, CBlobGetArea())` et `blobs.GetBlob(i).Area()`

Classes principales de cvBlobsLib (4/4)

- **Principaux opérateurs dérivés de COperadorBlob**
  - **Propriétés du *blob* lui-même**
    - **`CBlobGetArea()`** : surface
    - **`CBlobGetPerimeter()`** : périmètre
    - **`CBlobGetCompactness()`** : compacité
    - **`CBlobGetMoment()`** : moment (p,q)
    - **`CBlobGetMinX()`**, **`CBlobGetMinY()`** : coordonnées minimales
    - **`CBlobGetMean()`** : niveau de gris moyen
    - **`CBlobGetXYInside(cvPoint p)`** : teste si le point p appartient au *blob*
  - **Ellipse englobante**
    - **`CBlobGetAxisRatio()`** : ratio entre le grand axe et le petit axe
    - **`CBlobGetMajorAxisLength()`** : longueur du grand axe de l'ellipse
    - **`CBlobGetOrientation()`** : angle (en radians) du grand axe avec l'axe x
    - **`CBlobGetXCenter()`**, **`CBlobGetYCenter()`** : coordonnées du centre
    - **`CBlobGetDistanceFromPoint(x,y)`** : distance du centre au point (x,y)

Références (*en ligne*)

- **Ressources en anglais**
  - Article Wikipedia
    http://en.wikipedia.org/wiki/Connected_Component_Labeling
  - Blog de Steve Eddins (The Mathworks) sur l'analyse en CC
    http://blogs.mathworks.com/steve/2007/06/13/connected-component-labeling-wrapping-up
  - cvBlobsLib sur OpenCVWiki
    http://opencv.willowgarage.com/wiki/cvBlobsLib
- **Ressources en français**
  - Diaporama d'Alain Boucher
    http://www1.ifi.auf.org/ecole_ete/2003/DocVisionParOrdinateur/07-Images_binaires.ppt
  - Cours de Bruno Nazarian (*cf.* « Les images binaires »)
    http://bnazarian.free.fr/spip.php?article2
- **Autres approches pour les *blobs***
  - **cvblob**, autre bibliothèque (inclut le suivi des *blobs*)
    http://code.google.com/p/cvblob/
  - Gestion des composantes connexes en utilisant directement OpenCV
    http://www.associatedcontent.com/article/2717509/filtering_connected_components_using.html?cat=15