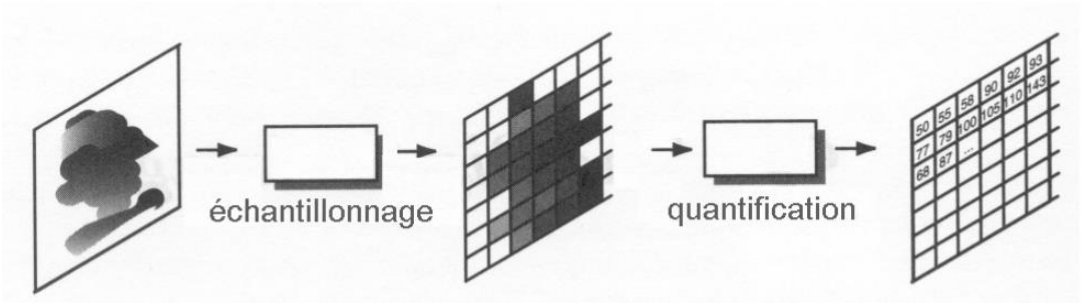
ISN S13: Le traitement d'images

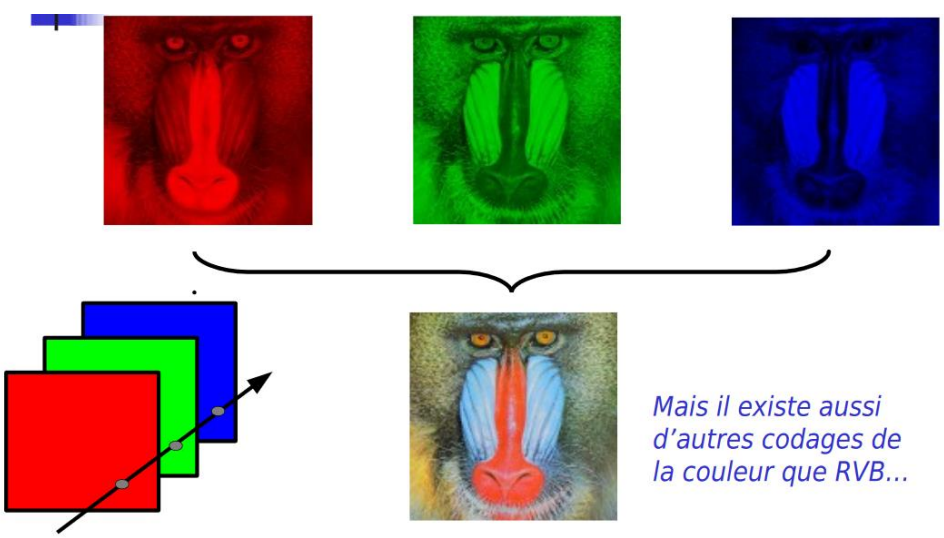
Objectif :
Découvrir l'utilisation des images sous Processing.

1. Le format d'une image

Une image numérique est une matrice de pixels. Pour une image en niveau de gris, chaque pixel à une valeur numérique comprise entre :
0 = noir
255 = blanc



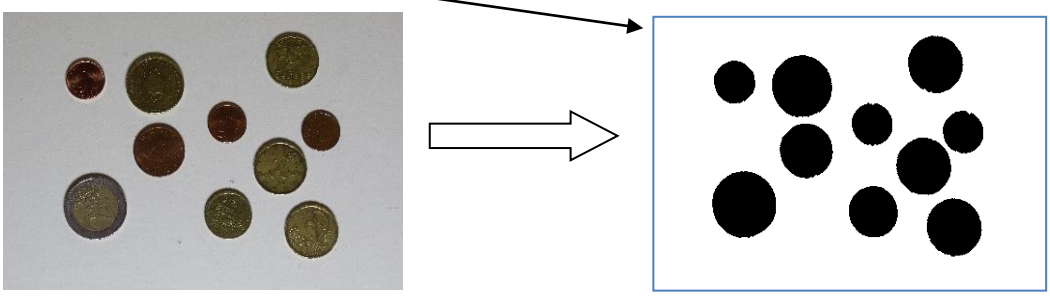
Si l'image est en couleur, chaque pixel a 3 composantes, le rouge, le vert et le bleu.



Ainsi, pour une image de 400 x 300 pixels en couleurs, il nous faudra 400 x 300 x 3 octets pour stocker cette image soit 360 ko.

2. Le traitement d'image

Dans la suite de ce TP nous allons apporter des modifications à l'image pour qu'un programme soit capable de compter les pièces de monnaie. Pour cela il faut supprimer toutes les informations inutiles contenues dans l'image qui pourraient fausser le comptage des pièces. L'idéal est d'obtenir une image comme celle-ci :



a) 1ère étape : conversion en niveau de gris

Pour faciliter les traitements ultérieurs de l'image, on va convertir celle-ci en niveau de gris, comment fait-on ? Pour chaque pixel, on calcule la moyenne des 3 composantes RVB et on assigne à ces 3 composantes la moyenne calculée.

Pseudo-code

```
POUR x = 0 à largeur de l'image – 1
        POUR y = 0 à hauteur de l'image – 1
                r ← composante rouge du pixel (x,y)
                v ← composante verte du pixel (x,y)
                b ← composante bleue du pixel (x,y)
                moy ← (r + v + b) / 3
                composante rouge du pixel (x,y) ← moy
                composante verte du pixel (x,y) ← moy
                composante bleue du pixel (x,y) ← moy
        FIN_POUR
FIN_POUR
```

Heureusement, il existe dans Processing une fonction qui fait cela, c'est la méthode **filter()** de la classe **PImage** (voir https://processing.org/reference/PImage_filter_.html).

I) Modifiez le programme image_processing.pde après le commentaire « REPERE 1 » pour convertir l'image en niveau de gris et l'afficher en haut à droite de la fenêtre. Enregistrez et tester le programme. Qu'observez-vous ? (copie d'écran)

b) 2ème étape : seuillage

On remarque que les pièces de monnaie sont plus foncées que le fond de l'image, l'idée est de convertir les pixels plus sombres qu'un certain **seuil** en noir, et les pixels plus clairs que ce seuil en blanc, ainsi on aura des pièces noires sur un fond blanc.

Pseudo-code

```
POUR x = 0 à largeur de l'image – 1
        POUR y = 0 à hauteur de l'image – 1
                r ← composante rouge du pixel (x,y)
                SI r < seuil
                        composante rouge du pixel (x,y) ← 0
                        composante verte du pixel (x,y) ← 0
                        composante bleue du pixel (x,y) ← 0
                SINON
                        composante rouge du pixel (x,y) ← 255
                        composante verte du pixel (x,y) ← 255
                        composante bleue du pixel (x,y) ← 255
                FIN_SI
        FIN_POUR
FIN_POUR
```

Là aussi, la méthode **filter()** de la classe **PImage** va nous rendre service, (voir https://processing.org/reference/PImage_filter_.html).

Mais il faut choisir ce seuil. Une méthode simpliste consiste à calculer la moyenne des pixels et d'utiliser cette moyenne comme seuil.

Pseudo-code de la fonction moyenne(image)

```
DEBUT
        somme ← 0
        POUR x = 0 à largeur de l'image – 1
                POUR y = 0 à hauteur de l'image – 1
                        r ← composante rouge du pixel (x,y)
                        somme ← somme + r
                FIN_POUR
        FIN_POUR
        nb_pixel ← largeur de l'image * hauteur de l'image
        moyenne ← somme / nb_pixel
        RETOURNER moyenne
FIN
```

Rien dans Processing ne permet de faire cela, il va falloir « *mettre les mains dans le cambouis* ». Pour cela on pourra utiliser les fonctions suivantes :

- Méthode get() de la classe PImage (https://processing.org/reference/PImage_get_.html)
- Fonction red() (https://processing.org/reference/red_.html)

II) Implémenter la fonction moyenne() selon le pseudo-code ci-dessus, cette fonction se trouve à la fin du programme.
Modifiez le programme image_processing.pde après le commentaire « REPERE 2 » pour :

- calculer le seuil grâce à la fonction moyenne()
- appliquer un seuillage sur l'image grâce à la méthode filter() de la classe PImage
- afficher l'image en bas à gauche de la fenêtre.

Enregistrez et tester le programme.
Qu'observez-vous ? (copie d'écran)

c) 3ème étape : filtrage

Il nous reste maintenant à supprimer les quelques points noires dans les zones blanches et les quelques points blancs dans les zones noires. Pour cela il existe le filtrage par **érosion** et par **dilatation**.

- Filtrage par **érosion** : les pixels blancs qui ont un voisin noir deviennent noirs (cela va éliminer les pixels blancs isolés).

- Filtrage par **dilatation** : les pixels noirs qui ont un voisin blanc deviennent blancs (cela va éliminer les pixels noirs isolés).

Là encore, la méthode **filter()** de la classe **PImage** fait tout le boulot.

Par une succession d'érosions suivies de dilatations ou de dilatations suivies d'érosions, on parvient à éliminer tous les pixels parasites.

III) Modifiez le programme image_processing.pde après le commentaire « REPERE 3 » pour effectuer une série de filtrage érosion/dilatation jusqu'à faire disparaître les pixels parasites et afficher l'image en bas à droite de la fenêtre.
Enregistrez et tester le programme.
Qu'observez-vous ? (copie d'écran)

d) 4ème étape : comptage

Maintenant que nous avons une image débarassée des parasites et des informations inutiles, nous allons pouvoir compter les piéces.

Pour cela nous allons compter les pixels noirs, mais comme il y en a plusieurs par pièces, on va effacer (mettre en blanc) les points noirs que l'on compte, les points noirs voisins, les points noirs voisins de ces voisins, etc… on va utiliser la **récursivité**.

Pseudo-code de la fonction compter(image)

```
DEBUT
   somme ← 0
   POUR x = 1 à largeur de l'image – 2
      POUR y = 1 à hauteur de l'image – 2
         SI composante rouge du pixel (x,y) = 0
            somme ←somme + 1
            effacerRecursivementPixel(x,y) ;
         FIN_SI
      FIN_POUR
   FIN_POUR
   RETOURNER somme
FIN
```

Vous remarquerez que x ne varie plus de 0 à (largeur de l'image – 1) mais de 1 à (largeur de l'image – 2), idem pour y. C'est pour ne pas tenir compte des points qui sont sur les bords de l'image. Ces points n'ont pas de voisin dans une direction or l'algorithme suivant ne teste pas cette éventualité.

Pseudo-code de la fonction effacerRecursivementPixel (image, x, y)

```
DEBUT
        composante rouge du pixel (x,y) ← 255
        composante verte du pixel (x,y) ← 255
        composante bleue du pixel (x,y) ← 255

        SI composante rouge du pixel (x-1, y) = 0
                effacerRecursivementPixel (x-1, y)
        FIN_SI

        SI composante rouge du pixel (x+1, y) = 0
                effacerRecursivementPixel (x+1, y)
        FIN_SI

        SI composante rouge du pixel (x, y-1) = 0
                effacerRecursivementPixel (x, y-1)
        FIN_SI

        SI composante rouge du pixel (x, y+1) = 0
                effacerRecursivementPixel (x, y+1)
        FIN_SI
FIN
```

IV) Implémenter les fonctions compte() et effacerRecursivementPixel() selon les pseudo-codes ci-dessus, ces fonctions se trouvent à la fin du programme.
Modifiez le programme image_processing.pde après le commentaire « REPERE 4 » pour compter et afficher le nombre de pièces.
Enregistrez et tester le programme.