LES DEVINETTES D'ÉPINAL

Le but de ce TD est de cacher la deuxième image dans la première. Dans notre exemple, l'image comporte beaucoup de blanc et la manipulation ne sera pas optimale, mais si les images sont davantage colorées, les résultats devraient être encore meilleurs.

Pour comprendre le principe, **imaginons que chaque couleur soit codée sur** $[\![0,99]\!]$.

Lorsque nous allons fusionner les images, nous perdrons de l'information (de deux images on n'en fait qu'une seule, ce qui revient à diviser le nombre total de pixels par 2).
Chaque couleur est codée par deux informations : le chiffre des unités et celui des dizaines. Puisque nous ne pouvons garder que la moitié de l'information, nous ne conserverons que le chiffre des dizaines qui est prépondérant par rapport aux unités. Ainsi, 81 sera remplacé par 80 (ce qui revient à ne garder que le chiffre 8).

Le pixel de la nouvelle image sera codé (pour chaque couleur) par un nombre à deux chiffres xy. Pour x, nous prendrons la valeur des dizaines de la première image et pour y celle des dizaines pour la deuxième image.

Par exemple si pour un pixel, la couleur rouge est de 81 pour la première image, et 42 pour la seconde, on codera le nouveau pixel par 84.

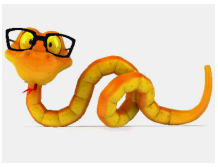
Le but de cette question est de faire la même chose, mais avec des couleurs codées sur $[\![0,255]\!]$ au lieu de $[\![0,99]\!]$.

Ainsi, au lieu de prendre un seuil de 10 pour séparer dizaines et centaines, on prendra un seuil de 16. Ceux qui veulent comprendre ce choix peuvent lire l'encadré ci-dessous.

(a) image

(b) image2

(c) images combinées



(d) extraite1

(e) extraite2

Figure 1: Camouflage d'une image dans une autre, puis décodage

Écriture en base 10 :
Habituellement, nous écrivons les nombres en base 10. Cela requiert 10 symboles que nous appelons chiffres : 0, 1, 2, 3, $\cdots$ 8, 9.
À l'aide de ces dix chiffres nous pouvons écrire tous les nombres entiers suivant la méthode dite *positionnelle* : la position du chiffre indique sa puissance de 10. Ainsi, le chiffre le plus à droite désigne les unités, le chiffre voisin sur sa gauche donne les dizaines, le suivant donne les centaines....
Le nombre 875 se lit donc avec le calcul $8\times10^2+7\times10^1+5\times10^0$.
Le nombre de chiffres autorisés donne la plage de données accessible. Par exemple si on a trois cases pour écrire notre nombre, on peut aller de 000 à 999, c'est-à-dire 10^3 nombres.

Écriture en base 2 :
Le système informatique est basé sur un système en base 2 qui est beaucoup plus simple à construire physiquement. Il n'y a besoin que de deux *chiffres* : 0 et 1 (creux et bosse, aimant vers le haut ou le bas...).
Les nombres sont donc stockés en base 2 et dans le cas de notre image, chaque couleur est écrite sous forme d'un octet (8 chiffres).
Chaque octet peut donc s'écrire $n=\overline{x_7x_6x_5x_4x_3x_2x_1x_0}$ avec $x_i\in\{0,1\}$.
La valeur de l'octet est obtenue par le même calcul qu'en base 10 : $n=x_7\times2^7+x_6\times2^6+\cdots+x_2\times2^2+x_1\times2^1+x_0\times2^0$.
Les nombres que l'on peut écrire ainsi vont de 0 à $2^8-1=255$.
L'idée de la fusion des deux images est de ne conserver que les quatre premiers chiffres de chaque couleur (les 4 suivants sont considérés mineurs) pour les recombiner (dans l'exemple en base 10, nous n'avions pris qu'un seul chiffre sur les deux).
Par exemple, si le rouge de l'image 1 est codé par 1001 1011 et le rouge de l'image 2 est codé par 0010 1111, alors on prendra pour l'image fusionnée la couleur 1001 0010.
Lors de la séparation des images, l'image 1 prendra alors la couleur 1001 0000 et l'image 2 : 1001 0000.

1) Programmer la fonction **fusion(image,image2)** qui fusionne les deux images.

2) Programmer la fonction **decombine(image)** qui décombine les deux images.

3) Essayer de trouver l'image cachée (voir fichier sur le site https://molin-mathmatiques.fr).