Informatique 1ere année, CPBX, TD5

Carole Blanc, Paul Dorbec

1 Une image peut en cacher une autre

On peut utiliser le fait que l'œil ne distingue pas les faibles différences de couleur pour dissimuler une image dans une autre. Prenons deux images de mêmes dimensions. Pour chacune des coordonnées (x, y), on va mélanger une partie du pixel (x, y) de l'image 1 avec une partie du pixel (x, y) de l'image 2 en faisant en sorte que l'image obtenue apparaisse presque comme l'image 1.

Pour simplifier, voyons ce qui se passe sur une seule composante de couleur, disons le rouge (même si en fait, on appliquera le même traitement aux trois couleurs). Imaginons que l'on veuille mélanger un pixel dans l'image 1 dont l'octet rouge vaut 190 qui s'écrit en binaire

10111110

avec l'octet rouge de l'image 2 qui vaut 121, c'est-à-dire

01111001

La technique que l'on va mettre en œuvre consiste à recombiner les cinq premiers chiffres binaires du premier pixel avec les trois premiers chiffres du second. De nos deux octets d'origine **10111**110 et **011**11001 on ne garde que les chiffres les plus significatifs, en mettant ceux de la première image en tête. Ce mélange des deux octets d'origine produit l'octet suivant :

10111011

On remarque qu'on a peu modifié la valeur par rapport à celle de l'image 1, parce qu'on a gardé les cinq bits les plus significatifs. Sur notre exemple, la différence est seulement de 3 : la nouvelle valeur est 187 au lieu de 190 dans l'image original. Au pire, on transforme les trois derniers bits de 000 à 111, ou vice-versa, ce qui fait une différence de 7, au plus. En résumé, la fusion des deux images ressemble beaucoup à la première image.

Maintenant on peut retrouver une couleur proche de celle du pixel de la seconde image grace aux 3 derniers bits, 011. On sait donc que la valeur du pixel original de la seconde image se situe entre 011 00000 et 011 11111. Si on utilise 011 00000 l'erreur est au maximum de 31 mais en ajoutant 00010000 à cette valeur on réduit l'erreur maximale à 16. Ça reste suffisant pour retrouver l'image cachée, même si la perte de précision la dégradera. Aussi, à partir d'un pixel de l'image modifiée, on reconstruit donc l'approximation du pixel de l'image cachée : 01110000. Ce nombre vaut 112 au lieu de 121 dans l'image 2 qu'on voulait cacher. On a donc fait une erreur de 9.

Question 1.1 *L'objectif est d'écrire une fonction permettant de découvrir une image cachée à partir d'une image source.*

1. *écrire une fonction* `valeur_derniers_bits(n)` *qui, à partir d'une valeur* `n` *codée sur un octet, retourne la valeur correspondant aux 3 derniers bits. Par exemple, si on transmet 187 à cette fonction, dont le code en binaire est 10111011, elle devra retourner le code des 3 derniers bits, c'est-à-dire 011 qui vaut 3.*
2. *écrire une fonction* `decaler(m)` *qui, à partir d'un entier* `m` *dont le code se représente sur 3 bits, retourne la valeur correspondant à ces 3 bits complétés par* 10000. *Par exemple, partant de 3 qui est codé sur 3 bits par 110, la fonction retournera 112, codé par* 11010000.
3. *écrire une fonction* `devoiler_image(img)` *qui retourne l'image cachée.*
4. *Utiliser cette fonction pour décoder l'image cachée dans* `stegano.png` *disponible sur la page suivante.*

Question 1.2 *écrire une fonction* `dissimuler_image(image1, image2)` *qui retourne une nouvelle image où l'image 1 cache l'image 2.*

2 Fusion et bannière

On cherche à réaliser une fusion de deux images. Récupérez l'image lune.png et plage.png sur le site du cours pour les exercices suivants.

Question 2.1 *Écrivez une fonction* `fusion_lineaire(img1,img2)` *qui retourne une nouvelle image qui est un dégradé linéaire vertical de l'image 1 vers l'image 2. Chaque couleur de pixel correspond à une moyenne des couleurs des pixels de l'image 1 et de l'image 2, pondérée en fonction de la hauteur du pixel. Pour une position verticale y du pixel, la nouvelle valeur sera calculée avec une fonction* $\frac{yx1+(h-y)x2}{h}$. *On souhaite ainsi produire une image dégradée dont le haut ressemble à l'image 1 et le bas à l'image 2. En testant avec* `lune.png` *comme première image et* `plage.png` *comme deuxième image, vous obtiendrez une image de grosse lune juste au dessus de l'eau.*

Question 2.2 *De la même façon, écrivez une fonction* `fusion_quadratique(img1,img2)` *qui utilise un ratio* $\frac{y^2x1+(1-y^2x2}{y^2+(1-y)^2}$.

Question 2.3 *Pour permettre de bien comparer les quatre images, créer enfin une fonction* `banniere(img1,img2)` *qui renvoie une image contenant en largeur dans l'ordre de gauche à droite :*

1. *l'image 1.*
2. *l'image 2.*
3. *la fusion linéaire des deux.*
4. *la fusion quadratique des deux.*

3 Taguer une image

Dans les exercices suivants, on cherche à écrire un texte sur une image. Le texte en noir est fourni dans une image dont le fond est blanc, dans les exemples on utilise l'image bonjour.png.

3.1 Placement

L'image qui contient le texte et l'image que l'on veut taguer arcEnCiel2.png ne sont pas nécessairement de la même taille.

Question 3.1 *Écrivez une fonction* `insertionPossible(img1,img2, x,y)` *qui prend en entrée* `img1` *(l'image à taguer),* `img2` *(l'image texte) et les coordonnées* `(x,y)` *du pixel dans* `img1` *à partir duquel on veut insérer* `img2`*. Cette fonction retournera vrai ou faux selon que* `img2` *peut être insérée entièrement sur* `img1` *ou non. Autrement dit elle vérifie si* `img2` *ne déborde pas de* `img1` *quand on place le pixel de coordonnées (0,0) de* `img2` *en (x,y) dans* `img1`*.*

3.2 Écriture : première idée

Question 3.2 *Écrivez une fonction* `insereTexteNoir(img1,img2, x,y)` *qui prend en entrée* `img1` *(l'image à taguer),* `img2` *(l'image texte) et les coordonnées* `(x,y)` *du pixel dans* `img1` *à partir duquel on veut insérer* `img2`*. Si l'insertion est possible, cette fonction modifiera* `img1` *en y insérant le texte contenu dans* `img2`*.*

3.3 Écriture : amélioration du résultat

Même si le texte est majoritairement écrit avec des pixels noirs le bord des lettres est aussi composé de pixels sombres non noirs. Le résultat produit par la fonction précédente ne donne pas un résultat satisfaisant car ces pixels sombres n'ont pas été retenus pour écrire le texte dans `img1`.

Question 3.3 *Écrivez une fonction* `insereTexteSombre(img1,img2, x,y)` *qui réalise l'insertion du texte en prenant en compte tous les pixels sombres du texte (c'est à dire les pixels non blancs, sans oublier les pixels noirs).*

3.4 Écriture : anti-aliasing

Le résultat produit par la fonction précédente n'est toujours pas satisfaisant car l'incrustation du texte comporte des défauts d'aliasing sur les bords. Afin d'éliminer ces défauts, la couleur de chaque pixel incrusté sera obtenue en mélangeant la couleur du pixel du texte avec celle du pixel de l'image originale selon la formule suivante :

$$v_{new} = (v_{initial} * v_{texte})//255$$

où v représente chacun des 3 niveaux de couleurs d'un pixel.

Question 3.4 *Écrivez une fonction* `insereTexteNet(img1,img2, x,y)` *qui réalise l'insertion du texte en appliquant la méthode proposée. Pensez à utiliser un nouvel exemplaire de l'image* `arcEnCiel2.png` *avant de la taguer.*

3.5 Image mosaique

Question 3.5 *Dans le fichier que vous avez téléchargé au début de l'épreuve se trouve une fonction* `mosaique` *qui assemble 4 images en une mosaïque* `2x2` *(voir ci-dessous). Écrivez l'appel à cette fonction pour créer l'image* `resultat`.

FIGURE 1 – `mosaique`