TP - TI : images discrètes

François Lepan

25 février 2013

1 Composantes d'une image couleur

1.1 Quelles sont les dimensions de la variable utilisée pour stocker l'image et que représentent-t-elles ?

Les dimensions de l'image sont égales à sa taille en pixels fois le nombre de couleurs (ici 800x600x3).

Pour un élément se situant à (23,23,z) de la matrice :
la valeur z = 1 correspond à la quantité de rouge du pixel (23, 23)
la valeur z = 2 correspond à la quantité de vert du pixel (23, 23)
la valeur z = 3 correspond à la quantité de bleu du pixel (23, 23)

1.2 Séparer les composantes rouge, verte et bleue

En niveau de gris Voici le code correspondant à la séparation des trois composantes :

```
img = imread("ti-semaine-3-mire.png");

// recupere les niveaux de gris pour la couleur rouge
imgRG = img(:,:,1);
// recupere les niveaux de gris pour la couleur bleu
imgGG = img(:,:,2);
// recupere les niveaux de gris pour la couleur vert
imgBG = img(:,:,3);

// affichage
imshow([imgRG, imgGG, imgBG]);
```

L'exécution du code précédent fournit la Fig. 1 sur laquelle on peut observer de gauche à droite les niveaux de gris pour la couleur rouge puis vert puis bleu.

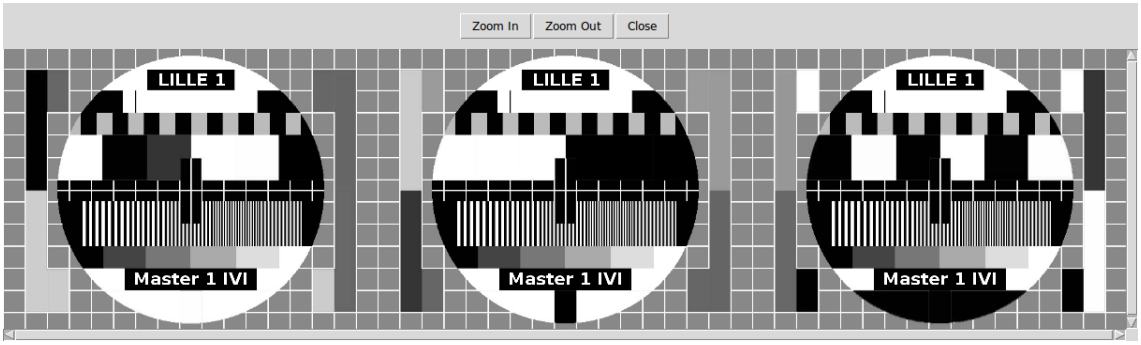

FIGURE 1 – Niveau de gris de gauche à droite des couleurs rouge, vert et bleu

En couleur Voici le code correspondant à la séparation des trois composantes :

```
function imgR = redColorsOf(img)
    imgR = img;
    imgR(:,:,2) = img(:,:,1)*0;
    imgR(:,:,3) = img(:,:,1)*0;
endfunction

function imgG = greenColorsOf(img)
    imgG = img;
    imgG(:,:,1) = img(:,:,1)*0;
    imgG(:,:,3) = img(:,:,1)*0;
endfunction

function imgB = blueColorsOf(img)
    imgB = img;
    imgB(:,:,1) = img(:,:,1)*0;
    imgB(:,:,2) = img(:,:,1)*0;
endfunction

img = imread("ti-semaine-3-mire.png");

// met les valeurs pour le vert et le bleu a 0
imgR = redColorsOf(img);
// met les valeurs pour le rouge et le bleu a 0
imgG = greenColorsOf(img);
// met les valeurs pour le vert et le rouge a 0
imgB = blueColorsOf(img);
// affichage
imshow([imgR, imgG, imgB]);
```

L'exécution du code précédent fournit la Fig. 2 sur laquelle on peut observer de gauche à droite les composantes couleur rouge puis vert puis bleu.

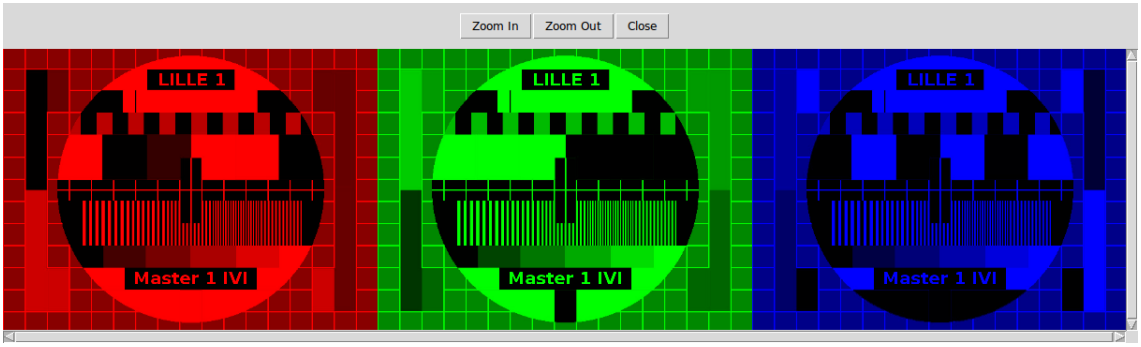

FIGURE 2 – Les composantes couleur rouge puis vert puis bleu.

Vérification de la récupération de l'image pour les trois composantes

```
imgRes = img;
imgRes(:,:,1) = img(:,:,1)*0;
imgRes(:,:,2) = img(:,:,2)*0;
imgRes(:,:,3) = img(:,:,3)*0;
imgRes(:,:,1) = imgR(:,:,1);
imgRes(:,:,2) = imgG(:,:,2);
imgRes(:,:,3) = imgB(:,:,3);
imshow(imgRes);
```

L'exécution du code précédent fournit la Fig. 3 qui est l'addition des trois composantes couleur rouge puis vert puis bleu de la Fig. 2.

FIGURE 3 – Image de base

2 Sur et sous-échantillonnage

2.1 En considérant que l'image de mire utilisée précédemment a été acquise avec une résolution de 72 pixels/pouce selon les deux directions, déterminer les dimensions du support initial de l'image.

Sachant que l'image fait 600x800 pixels il suffit de faire :
600 / 72 = 8.33 pouce
800 / 72 = 11.11 pouce
Et donc le support initial à pour dimension 8.33 x 11.11pouces

2.2 Sous-échantillonnage de l'image, fonction *sousEch(img,n)*

Voici la fonction correspondante :

```
function im = sousEch(img,n)
    img = img(:,:,1);
    img = im2double(img);

    c = size(img,1)/n
    l = size(img,2)/n

    new_img = zeros(c,l)

    for i = 1 : c
        for j = 1 : l

          for k = 1 : n
              for m = 1 : n
                  new_img(i,j) = new_img(i,j) + img((i*n)+(k-n),(j*n)+(m-n));
              end
          end

          new_img(i,j) = new_img(i,j) / (n*n);

        end
    end

    im =  new_img
endfunction

img = imread("ti-semaine-3-mire.png");
sousEchImg = sousEch(img,2);
imshow(sousEchImg);
```

L'exécution du code précédant nous fournit la Fig. 4. Cette figure représente un sous échantillonnage de facteur 2 de la Fig. 3 pour sa composante rouge.

Le sous-échantillonnage de facteur 2 dans la hauteur et dans la largeur correspond à prendre quatre pixels de l'image de base et de les transformer en un seul pixel sur la nouvelle image avec pour valeur la moyenne des quatre pixels (*cf. Fig.*5).

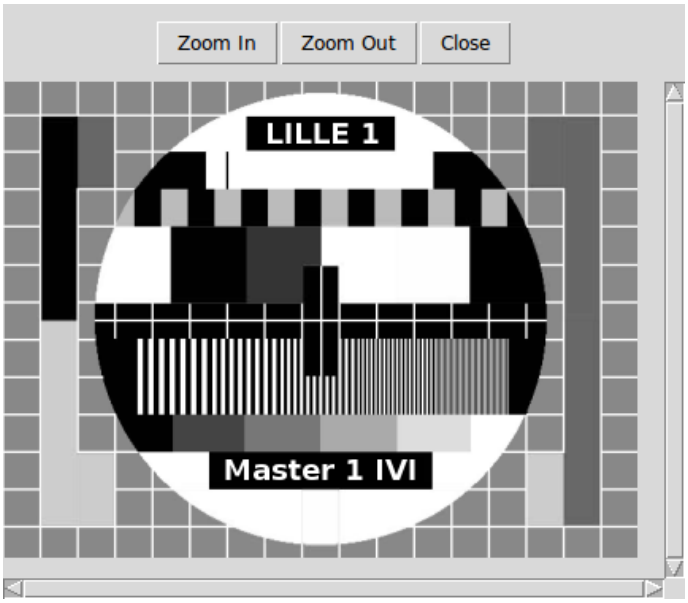

FIGURE 4 – Sous-échantillonnage de facteur 2 de la Fig. 3 pour sa composante rouge

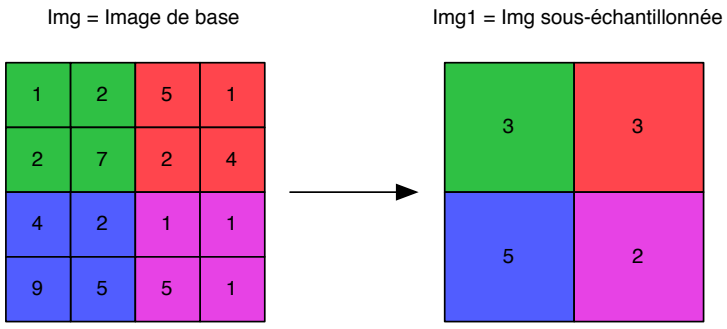

FIGURE 5 – Schéma du sous-échantillonnage

2.3 Sur-échantillonnage de l'image, fonction *surEch(img,n)*

Voici la fonction correspondante :

```
function im = surEch(img,n)

    img = img(:,:,1);
    img = im2double(img);

    c = size(img,1)
    l = size(img,2)

    new_img = zeros(c*n,l*n)

    for i = 1 : c
        for j = 1 : l

          for k = 1 : n
              for m = 1 : n

               new_img((i*n)+(k-n),(j*n)+(m-n)) = img(i,j)

              end
          end

        end
    end

    im =  new_img
endfunction

img = imread("ti-semaine-3-mire.png");
surEchImg = surEch(img,2);
imshow(surEchImg);
```

L'exécution du code précédant nous fournit la Fig. 6. Cette figure représente un sur-échantillonnage de facteur 2 de la Fig. 3 pour sa composante rouge.

Cette image est coupée car trop grande (1600 * 1200 pixels) pour mon écran (1800 * 1000). Mais cet agrandissement est normal vu que pour un pixel on le duplique quatre fois.

Le sur-échantillonnage de facteur 2 dans la hauteur et dans la largeur correspond à prendre un pixel de l'image de base et de le transformer en quatre pixels sur la nouvelle image qui auront la même valeur que le pixel de l'image de base (*cf. Fig.*7).

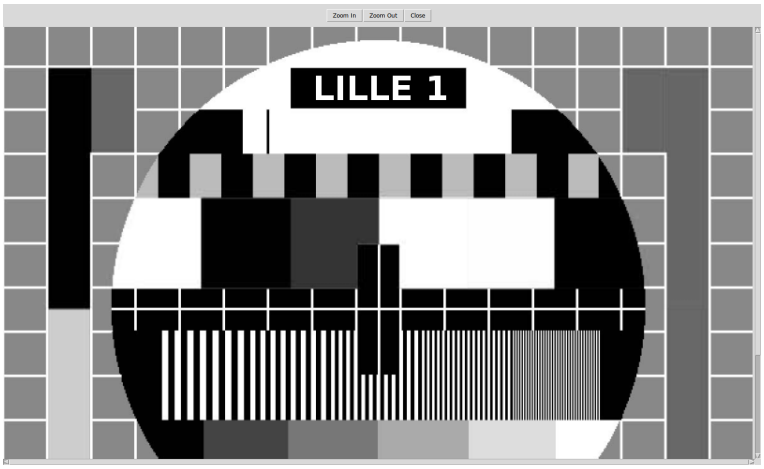

FIGURE 6 – Sur-échantillonnage de facteur 2 de la Fig. 3 pour sa composante rouge

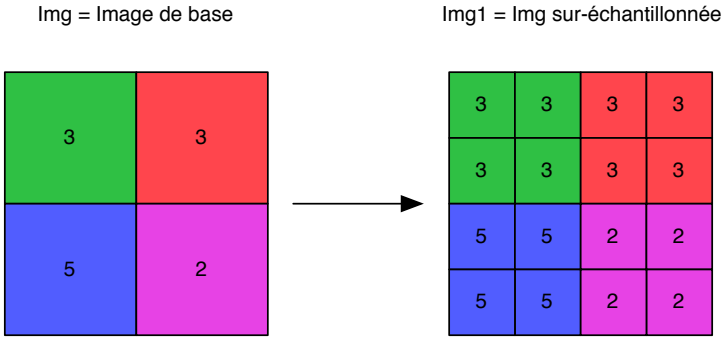

FIGURE 7 – Schéma du sur-échantillonnage

2.4 Sous-échantillonnage suivi d'un sur-échantillonnage

Voici le code correspondant :

```
img = imread("ti-semaine-3-mire.png");
ssEchImg = sousEch(img,2);
surEchImg = surEch(ssEchImg,2);
imshow(surEchImg);
```

En appliquant successivement le sous-échantillonnage suivi du sur-échantillonnage on obtient la Fig.8.

On peut observer un léger flou autour des changement de couleurs. Ceci est dû à une perte de donnée lors du sous-échantillonnage qui est répercutée sur l'image finale. (*cf. Fig.*9). On ne se rend pas bien compte du flou généré sur ce rapport c'est pourquoi je vous ai donné en plus dans cette archive l'image (*cf.*Fig.8).

Remarque : si on faisait l'opération dans le sens inverse (sur-échantillonnage puis sous-échantillonnage) il n'y aurait pas de perte de données (*cf.*Fig.10).

3 Quantification

3.1 Fonction *quantif(img,n)*

```
function im = quantif(img,n)
    quantificateur = 2^n

    img = img ./(256/quantificateur)

    round(img)

    img = img.*(256/quantificateur)

    im = img
endfunction
```

3.2 Calcul des images qui seraient obtenues avec une quantification sur 6 bits, 4 bits et enfin 1 bit de la composante verte de l'image de mire.

Voici le code et les images correspondantes :

```
img = imread("ti-semaine-3-mire.png");
img = img (:,:,2);

img = quantif(img,6);
imshow(img);

img = quantif(img,4);
imshow(img);

img = quantif(img,1);
imshow(img);
```

- quantification sur 6 bits : Fig.11
- quantification sur 4 bits : Fig.12
- quantification sur 1 bit : Fig.13

On observe que moins il y a de bit pour quantifier les couleurs moins il y a de couleurs différentes sur l'image obtenue. Pour 1 bit il n'y en a que 2 (2^1), pour 4 bits il y en a 16 (2^4) et pour 6 bit il y en a 64 (2^6).

3.3 Modification de l'image de Lena

Voici le code correspondant aux modifications des composantes de l'image :

```
img = imread("ti-semaine-3-lena.png");

// Composante rouge sous-échantillonnée d'un facteur 2 en horizontal
// et en vertical et quantifiée sur 5 bits
img1 = img(:,:,1);
```

```
img1 = sousEch(img1,2);
img1 = quantif(img1,5);

// Composante bleue sous-échantillonnée d'un facteur 4 en horizontal
// et en vertical et quantifiée sur 3 bits
img2 = img (:,:,3);
img2 = sousEch(img2,4);
img2 = quantif(img2,3);

// Sur-échantillonnage des composantes afin d'avoir des matrices de même taille
img1 = surEch(img1,2);
img2 = surEch(img2,4);

// On enregistre les resultats dans une nouvelle image
newimg = zeros(size(img,1),size(img,2),3);
newimg(:,:,1) = img1;
newimg(:,:,2) = im2double(img(:,:,2));
newimg(:,:,3) = img2;

imshow(newimg);
```

L'exécution du code précédant fournit la Fig.14. D'après ces résultats je pense que ma fonction *quantif(img,n)* n'est pas bonne car je devrais être en mesure de visualiser des changements par rapport à l'image initial. J'ai essayé de changer les valeurs de quantifications mais je n'ai pas vu de différences non plus.

4 Repliement de spectre

4.1 Période du motif

On observe qu'il y a 8 motifs sur cette images (8 bandes blanches et 8 bandes noires).Un motif représente donc une bande blanche suivie d'une bande noire.

De plus l'image fait 800 pixels en largeur, 1 pouce = 0.0254 mètre, et la résolution horizontale est de 150 pixels/pouce.

période en pixels

$$une\ periode = 800/8\ pixels$$
$$une\ periode = 100\ pixels$$

fréquence spatiale en cycle/pixels

$$frequence\ spatiale = 1/100\ cycle/pixels$$

période en mètres

$$\begin{aligned} une\ periode &= 100\ pixels \\ resolution &= 150 pixels/pouce \\ 1\ pouce &= 0.0254\ m \end{aligned}$$

$$\begin{aligned} une\ periode &= 100/150\ pouce \\ une\ periode &= 2/3\ pouce \\ une\ periode &\approx 2/3 * 0.0254\ m \\ une\ periode &\approx 0.016933333\ m \end{aligned}$$

fréquence spatiale en cycle/m

$$\begin{aligned} frequence\ spatiale &\approx 1/0.016933333\ cycle/m \\ frequence\ spatiale &\approx 59.055119273\ cycle/m \end{aligned}$$

4.2 Sous-échantillonnage de l'image du motif sinusoïdal afin de simuler des grilles d'échantillonnage de résolutions :

- 10 fois plus faibles : Fig.15
- 20 fois plus faibles : Fig.16
- 50 fois plus faibles : Fig.17
- 75 fois plus faibles : Fig.18

On observe que lorsqu'on diminue la taille de la grille d'échantillonnage jusqu'a 50 fois plus faibles le nombre de motif reste inchangé, seul les couleurs changent. Mais lorsqu'on diminue la taille 75 fois, il n'y a plus autant de motif.

Calculons sa fréquence pixels afin de voir si il y a un changement selon la taille de la grille d'échantillonnage. On observe qu'il y a 2.5 motifs sur cette figure.

$$\begin{aligned} une\ periode &= 800/2.5\ pixels \\ une\ periode &= 320\ pixels \\ frequence\ spatiale &= 1/320\ cycle/pixels \end{aligned}$$

On voit bien qu'il y a un changement dans la période et donc qu'il y a un seuil pour lequel la fréquence de la période diminuera si on diminue encore la taille de la grille.

5 Annexe

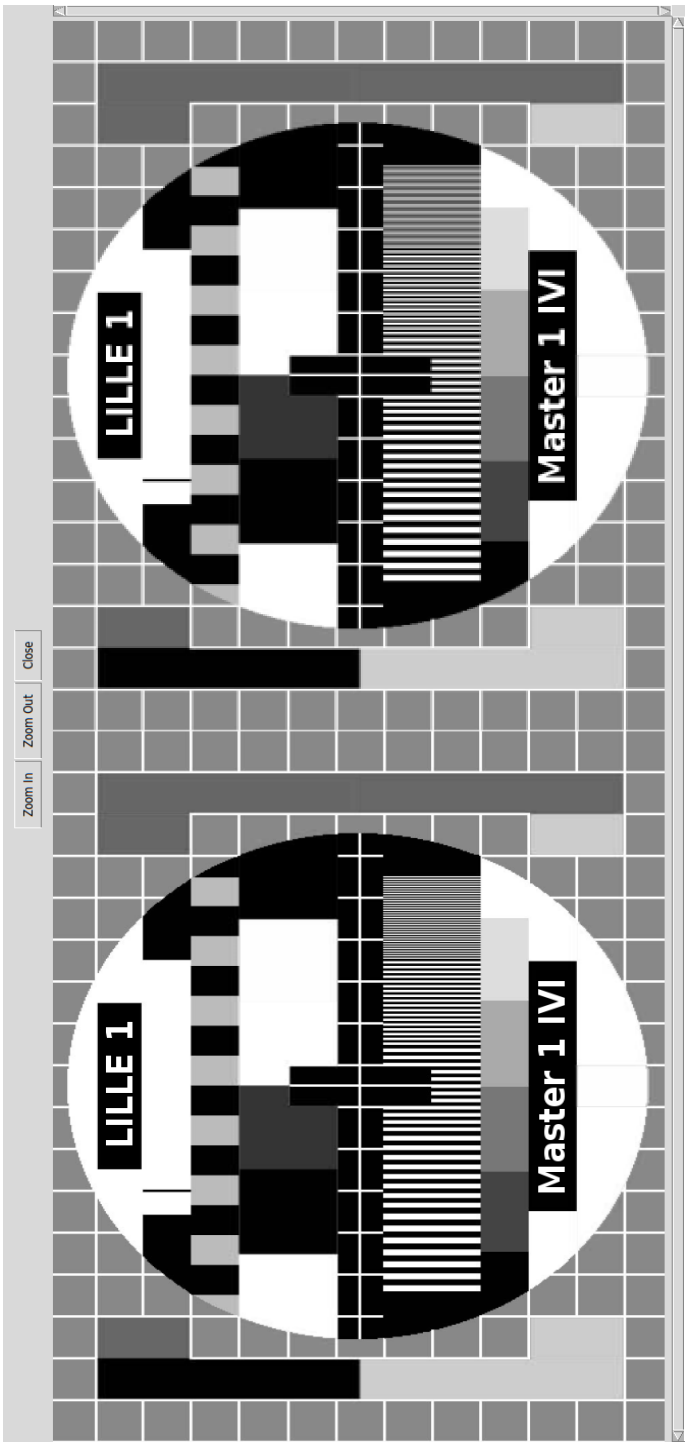

FIGURE 8 – De bas en haut : image composante rouge, image composante rouge sous-échantillonnée de facteur 2 puis sur-échantillonné de facteur 2

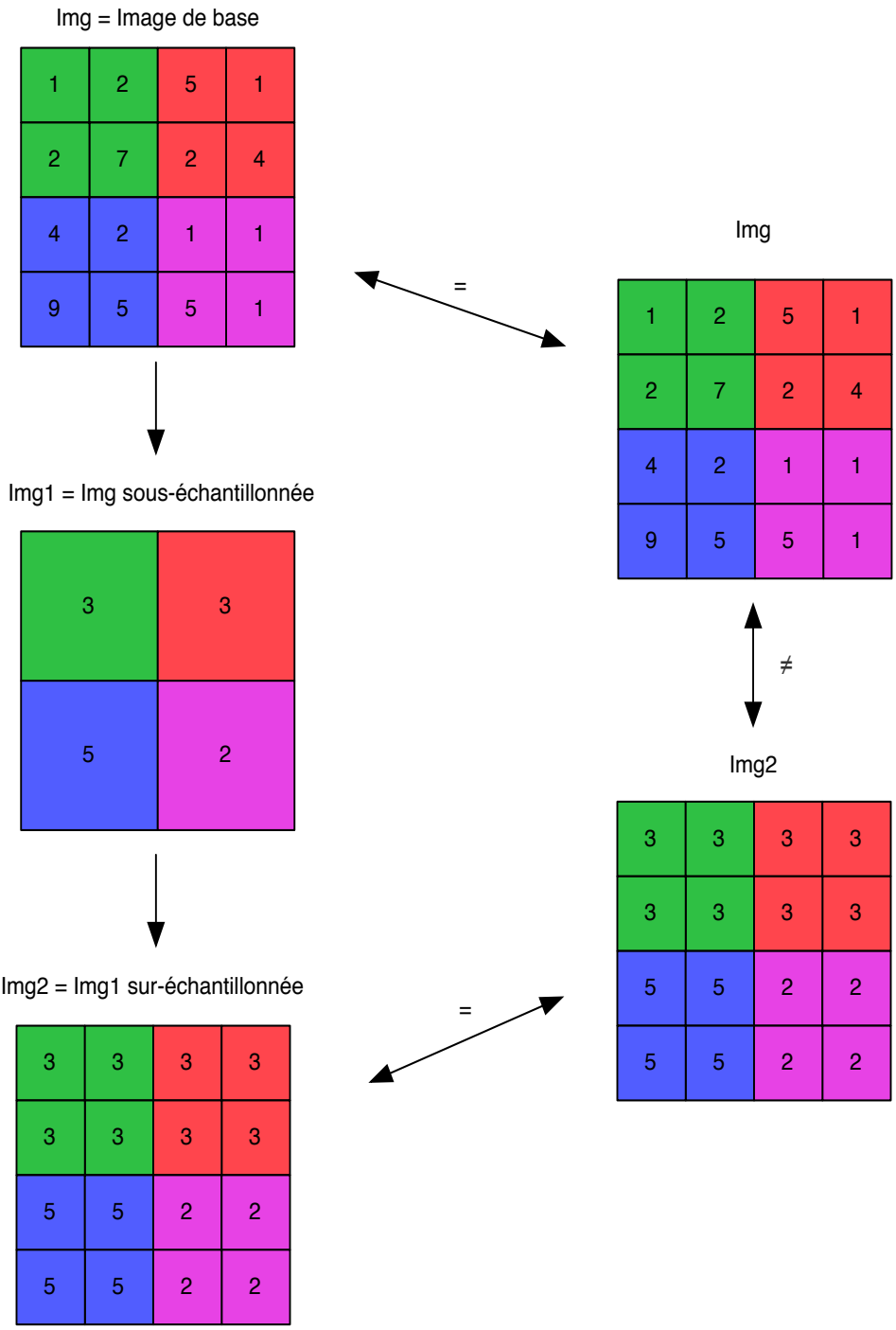

FIGURE 9 – Schéma de l'application successive du sous-échantillonnage suivi du sur-échantillonnage

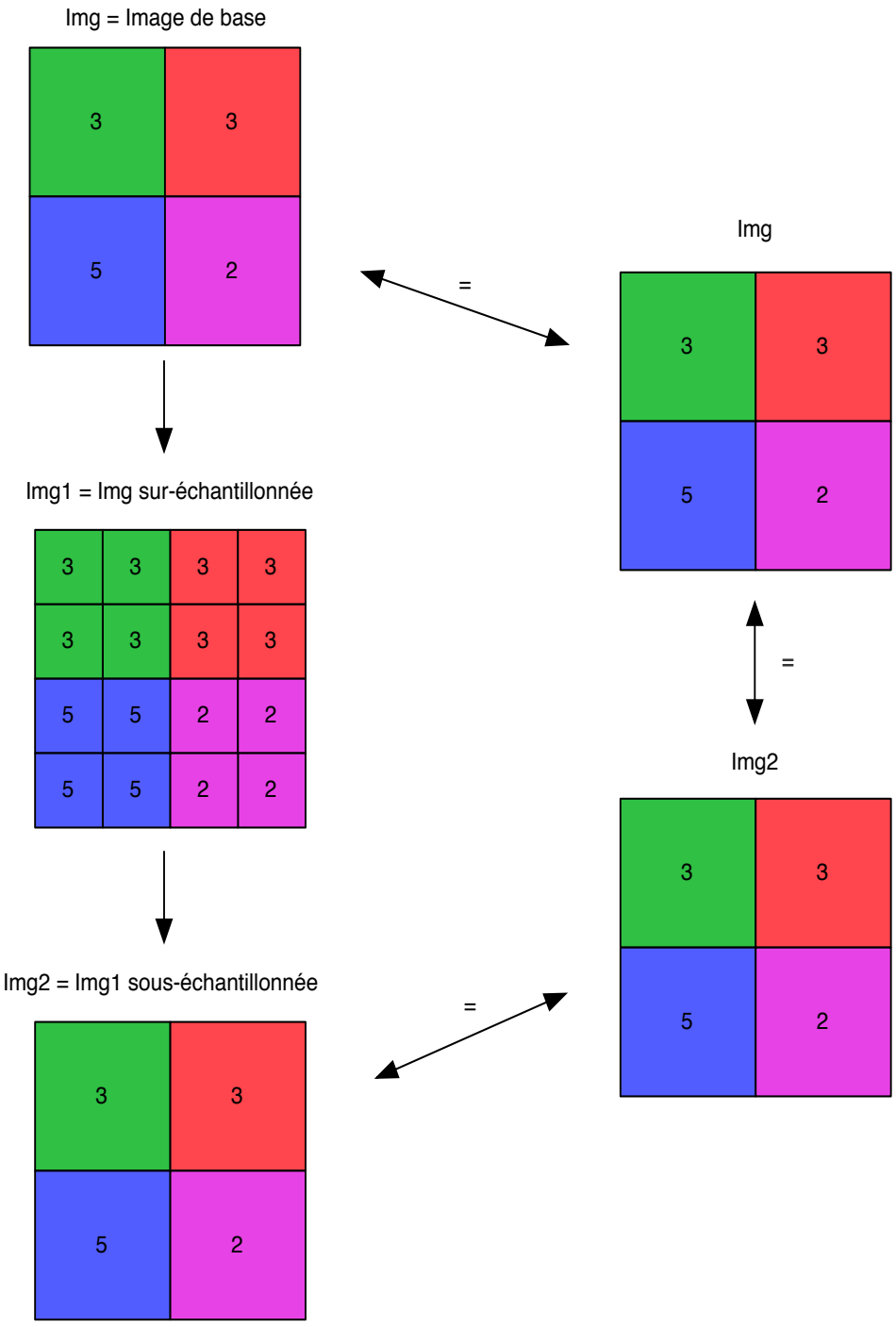

FIGURE 10 – Schéma de l'application successive du sur-échantillonnage suivi du sous-échantillonnage

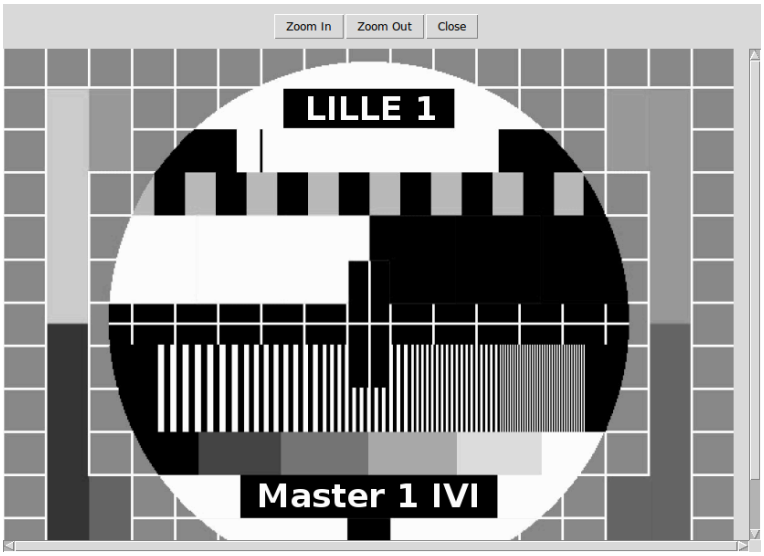

FIGURE 11 – Quantification sur 6 bits de la composante verte de l'image de mire

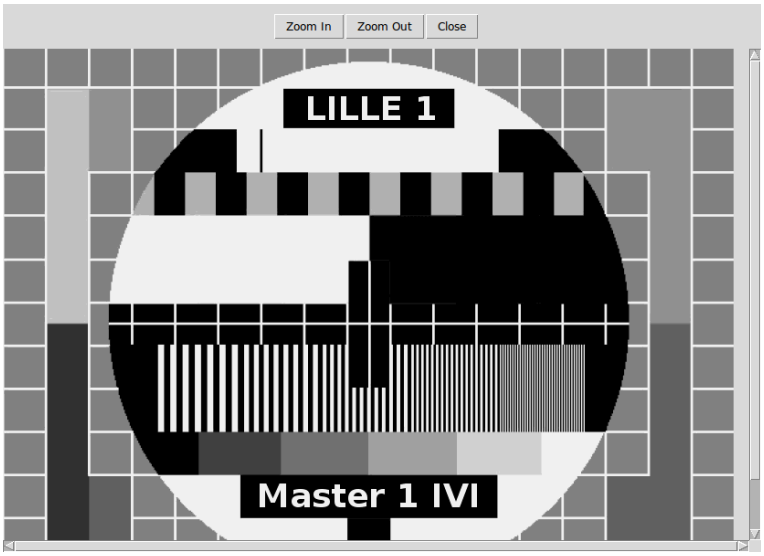

FIGURE 12 – Quantification sur 4 bits de la composante verte de l'image de mire

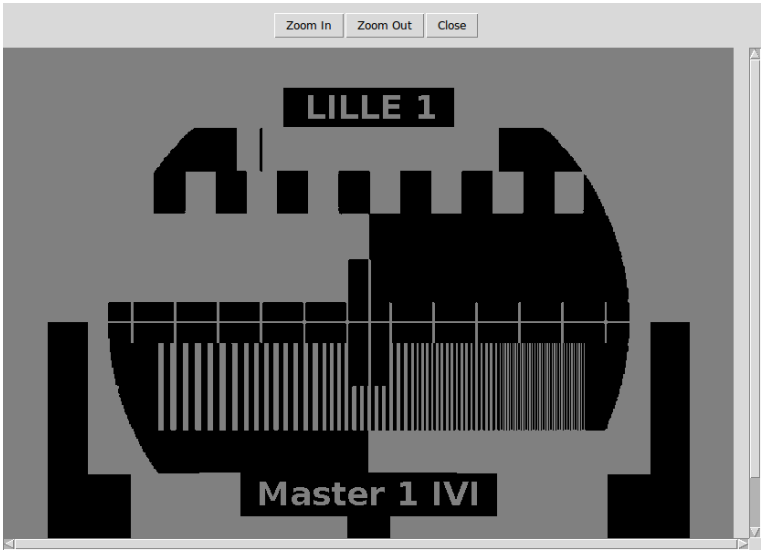

FIGURE 13 – Quantification sur 1 bit de la composante verte de l'image de mire

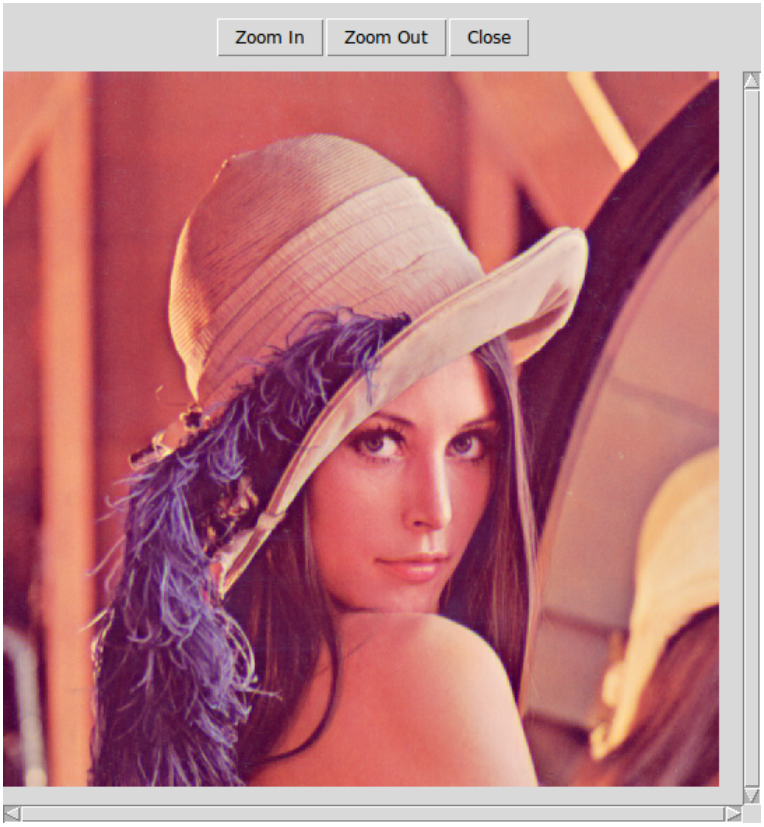

FIGURE 14 – Modification de l'image de Lena : composante rouge sous-échantillonnée d'un facteur 2 en horizontal et en vertical et quantifiée sur 5 bits, composante verte identique à l'original, composante bleue sous-échantillonnée d'un facteur 4 en horizontal et en vertical et quantifiée sur 3 bits

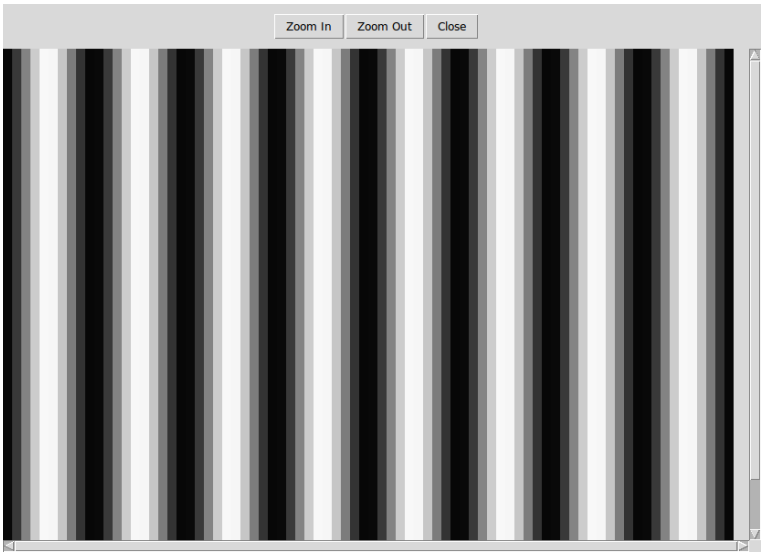

FIGURE 15 – Sous-échantillonnage de l'image du motif sinusoïdal afin de simuler une grille d'échantillonnage de résolution 10 fois plus petite

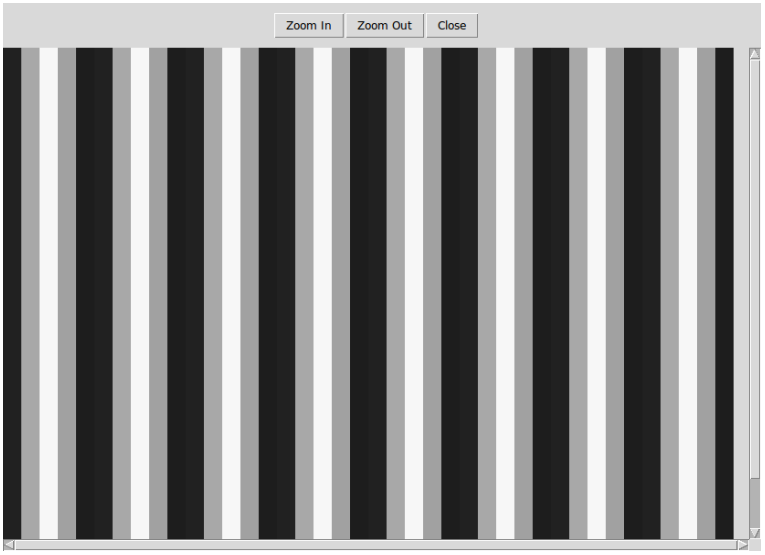

FIGURE 16 – Sous-échantillonnage de l'image du motif sinusoïdal afin de simuler une grille d'échantillonnage de résolution 20 fois plus petite

FIGURE 17 – Sous-échantillonnage de l'image du motif sinusoïdal afin de simuler une grille d'échantillonnage de résolution 50 fois plus petite

FIGURE 18 – Sous-échantillonnage de l'image du motif sinusoïdal afin de simuler une grille d'échantillonnage de résolution 75 fois plus petite