Chapitre 2

Notions de traitement d'images

Transformation ponctuelle

Table de conversion (des couleurs)

Introduction (1/2)

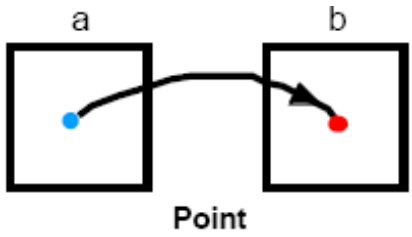
- **3 types d'opérations en traitement d'image:**

  - m : indice des lignes

  - n : indice des colonnes

  • Transformation point à point

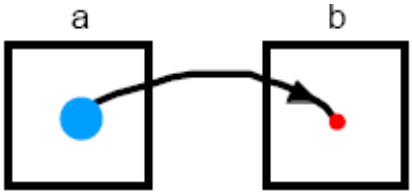
  ● = [m=m_0, n=n_0]



Point

  • Transformation locale vers un point

  ● = [m=m_0, n=n_0]



Local

En traitement d'image, on peut considérer 3 types de transformations. Les deux premiers types de transformation sont présentés sur la figure ci-dessus :

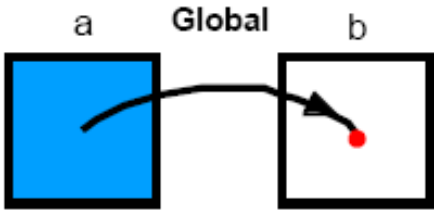
- ***de point à point***, telle que la valeur P(m_0, n_0) d'un pixel de l'image résultat « ***b*** » dépende uniquement de la valeur P(m_0', n_0') d'un pixel de l'image d'entrée « ***a*** ». Typiquement, les coordonnées (m_0', n_0') sont égales aux coordonnées (m_0, n_0) ;

- ***d'une zone locale vers un point***. Cette fois la valeur d'un pixel de l'image ***b*** ou d'un élément de ***b*** dépend d'un ensemble de pixels de ***a*** pris au sein d'une fenêtre R(m, n). Cette fenêtre est typiquement formée d'un nombre limité de pixels localisés autour du pixel (m_0', n_0') de l'image d'entrée ***a***. Par exemple, R(m, n) peut-être un bloc rectangulaire de taille (K × L) pixels, ou plus généralement un voisinage de pixels autour de (m_0', n_0') et qui conserve sa forme et sa taille quelles que soient les coordonnées (m_0', n_0').

Introduction (2/2)

- **Transformation globale vers un point**

● = [m=m_o, n=n_o]



Caractérisation de ces transformations

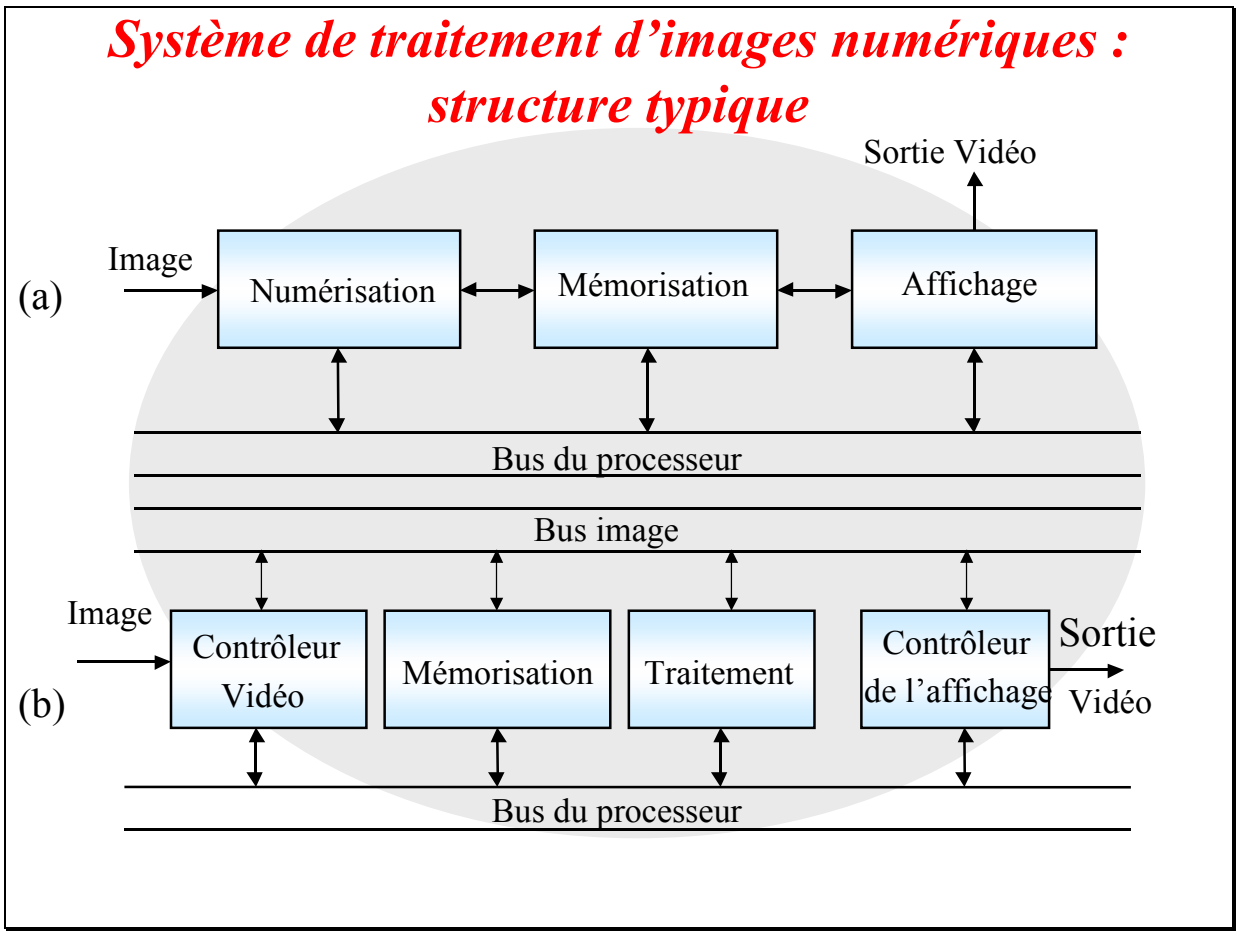
Opération	Caractérisation	Complexité par pixel
• Point	- la valeur de sortie, à une coordonnée spécifique, dépend uniquement de la valeur d'entrée à la même coordonnée.	constant
• Locale	- la valeur de sortie, à une coordonnée spécifique, dépend des valeurs d'entrée dans le voisinage de cette même coordonnée.	P^2
• Globale	- la valeur de sortie, à une coordonnée spécifique, dépend de toutes les valeurs de l'image d'entrée.	N^2

Taille Image : N x N ; taille du voisinage : p x p ; complexité: en nombre d'opérations par pixel

Le dernier type de transformation est présenté sur la figure ci-dessus.

- ***transformation globale vers un point***. La valeur calculée d'un élément (m_0, n_0) de la sortie dépend de la totalité des pixels de l'image d'entrée ***a***. C'est typiquement la cas lorsque l'on transforme globalement l'image du domaine spatial vers le domaine fréquentiel. La transformation de Fourier discrète (TFD) ou la transformation en cosinus discrète (TCD) sont des exemples bien connus (et qui seront développés dans un chapitre ultérieur) où chaque composante fréquentielle $X(\nu_X, \nu_Y)$ est fonction de l'ensemble des pixels de l'image à transformer.

Si nous considérons la complexité du traitement en terme de nombre d'opérations nécessaires pour obtenir une donnée de l'image de sortie, cette complexité est très liée au type de traitement effectué : de 1 à N^2 (pour une image carrée ($N \times N$)), ou de 1 à P^2 si on considère un opérateur mettant en œuvre une fenêtre de taille ($P \times P$) pixels, tel une convolution (cf. filtrage linéaire).
Un dernier aspect à considérer concerne les temps d'accès en mémoire image, plus ou moins réguliers, ce qui peut ralentir sérieusement le temps de traitement de l'image.



La structure typique d'un système de traitement d'image est illustré par la figure ci-dessus.

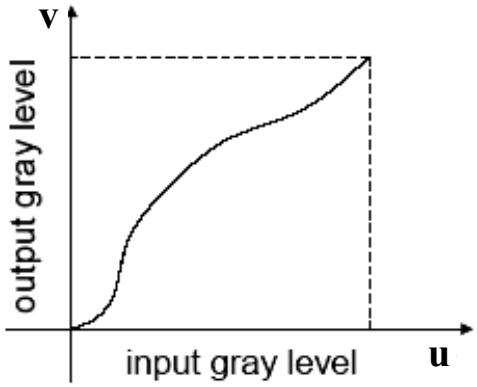
Nous découvrons en haut un exemple simple d'une chaîne de traitement constituée de la numérisation du signal vidéo analogique (échantillonnage puis quantification) produisant les données de l'image numérique. Ces données peuvent ensuite être stockées en mémoire image à la fréquence d'échantillonnage de la vidéo. Elles peuvent enfin être transmises à la seconde partie du système et/ou affichées à l'aide d'un moniteur.

Le traitement de l'image numérique est effectué à l'aide d'un processeur.
En bas, une unité dédiée au traitement d'image présente une architecture particulière. Elle est composée d'un processeur capable de réaliser des opérations (aussi bien linéaires que non-linéaires) en temps réel sur des fenêtres de taille limitée (par exemple (3×3) ou (5×5)), des transformations ponctuelles sur l'image avant et/ou après son traitement (utile à l'affichage).

Transformation ponctuelle

Transformations ponctuelles

• fait correspondre à un niveau donné de gris ou de couleur **u** du signal d'entrée **e**, un nouveau niveau de gris ou de couleur **v** du signal de sortie **s**, i.e. **v = f (u)**.



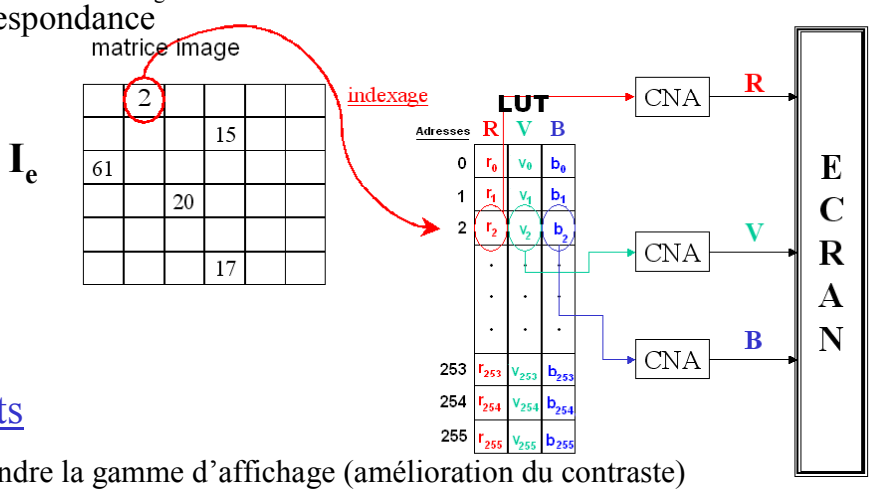
• les transformations ponctuelles sont utilisées, souvent en visualisation, pour mettre en évidence des pixels satisfaisant à une propriété donnée.

Une transformation ponctuelle peut facilement être réalisée à partir de données numériques. Elle correspond à la transformation d'une valeur (scalaire) en une autre (scalaire ou vectorielle pour une image couleur). Elle est entièrement déterminée par la fonction donnant les valeurs de sortie correspondant à chacune des valeurs d'entrée. L'usage d'une transformation ponctuelle permet, entre autres, de modifier l'apparence de l'image à l'affichage (exemple : affichage de fausses couleurs à partir d'une image en niveaux de gris), ou encore de modifier le niveau entrant de gris ou de couleur afin de compenser certaines non-linéarités dues au système d'affichage (moniteur). Une autre utilisation possible est la modification du contraste de l'image afin de le rehausser.

Transformation ponctuelle pour l'affichage

Principe (pour une image en niveau de gris)

Chaque pixel de l'image d'entrée I_e, ayant le niveau de gris N_g, a une niveau de gris transformé en $T(N_g)$ dans l'image de sortie I_s grâce à l'utilisation d'une table de mise en correspondance



Buts

- Étendre la gamme d'affichage (amélioration du contraste)
- Correction non-linéaire de l'image
- Binarisation de l'image
- Segmentation (découpage de l'image en régions composées de pixels ayant la même valeur)

Le principe de base pour la transformation ponctuelle d'une image numérique (monochrome) est de considérer la valeur d'entrée d'un pixel comme une adresse et de lire le contenu de la mémoire à cette adresse. Pour cela, les valeurs originales doivent avoir été converties préalablement en valeurs entières positives sur une échelle allant typiquement de 0 à $2^L - 1$ (par quantification et codage des niveaux de gris sur 2^L valeurs différentes). Pour un codage sur 8 bits par pixel, la taille de la mémoire sera de 256 octets.

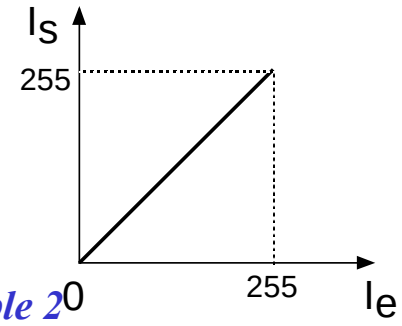
A chaque index « i » va correspondre une couleur créée par la combinaison r_i, v_i et b_i des couleurs primaires Rouge-Vert-Bleu. En effet, les valeurs r_i, v_i et b_i sont envoyées dans trois convertisseurs numérique-analogique (CNA) de façon à venir exciter respectivement les canons Rouge, Vert, et Bleu de l'écran. On obtient alors à l'affichage une couleur créée par la synthèse additive de trois couleurs primaires (technologie cathodique).

Ce type de mémoire est appelée « table de mise en correspondance » (ou table de conversion, appelée aussi LUT signifiant en anglais *Look-Up Table*). Elle est construite avant son utilisation : ni son contenu, ni les pixels de l'image ne sont modifiés durant la transformation.

La faible taille de la table permet éventuellement de la modifier pendant un temps de synchronisation vidéo (temps pendant lequel la vidéo n'est plus affichée à l'écran TV), c'est une façon de réaliser un fondu-enchaîné de transition entre 2 plans vidéo. Typiquement, pour des images monochromes, un système dispose d'une LUT à l'entrée et d'une seconde LUT en sortie du système de traitement d'images numériques. Dans le cas d'une image couleur, il faut donc considérer deux ensembles de trois LUTs : une LUT pour chacune des trois composantes couleur RVB.

Exemples de transformations ponctuelles « typiques »

Exemple 1



La transformation identité : $I_s = I_e$

Exemple 2

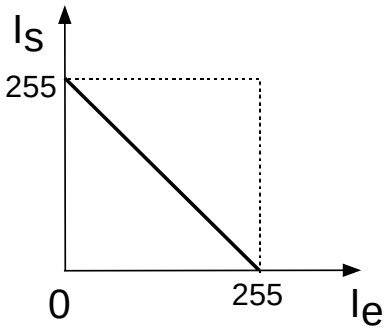


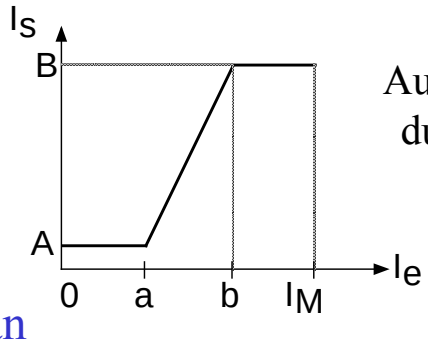
Image en “inverse vidéo”

inversion de l’échelle en niveau de gris

Pour ne pas modifier les valeurs d’une image, il faut utiliser une LUT « transparente » où le contenu à l’adresse du niveau de gris Ng est Ng. C’est la transformation Identité (exemple 1). Les niveaux de gris de Is correspondent donc aux niveaux de gris de Ie. Pour réaliser une inversion vidéo (exemple 2) où une petite valeur en niveau de gris est transformée en une valeur élevée et inversement, le contenu de la LUT à l’adresse Ng est (2^L - Ng –1).

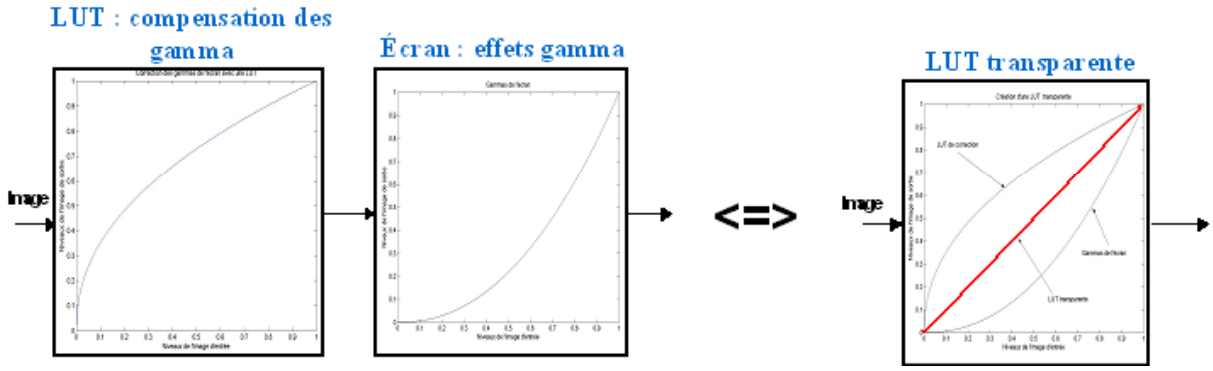
Exemples de transformations en niveaux de gris

Correction de
de la dynamique des
niveaux de gris
(*Re-scaling*)



Augmentation
du contraste

Correction du gamma écran



T2 o T1 = Opérateur Identité

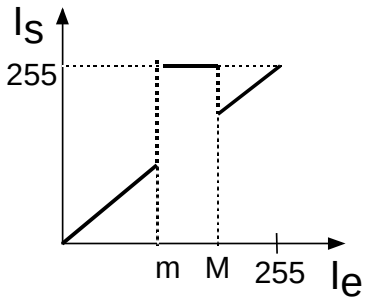
Deux exemples de transformations pour des images en niveaux de gris :

<u>figure du haut</u> : tous les pixels de l'image d'entrée Ie ayant une valeur inférieure à « a » (respectivement supérieure à « b ») auront la valeur « A » (respectivement « B ») dans l'image de sortie Is. Les niveaux de gris u_e dans l'image Ie ayant une valeur comprise entre a et b sont transformés en une nouvelle valeur v_s comprise entre A et B. $v_s = \left[\frac{u_e - a}{b - a}\ (B - A)\right] + A$.
Le but est de faire croître la dynamique des valeurs des pixels de l'intervalle de départ [a, b] et de rehausser le contraste. Pour l'augmentation du contraste, on a (B-A) > (b-a).

<u>figure du bas</u> : elle montre une correction typique d'affichage. Un tube cathodique est naturellement non-linéaire : l'intensité lumineuse reproduite à l'écran est une fonction non-linéaire de la tension d'entrée. La ***correction gamma*** peut être considérée comme un procédé permettant de compenser ce phénomène pour obtenir une reproduction fidèle de l'intensité lumineuse. Les effets ***gamma*** sont modélisés par des fonctions du type $f(x)=x^{\gamma}$, où γ est un réel qui varie entre 2 et 2,5 dans le cas de la télévision. Pour compenser ces effets, on modélise les gamma de l'écran par une LUT, et on crée la LUT de correction inverse afin que la composée « LUT correction o transformation non-linéaire écran » génère une transformation identité.

Transformations continues par morceaux

• Exemple

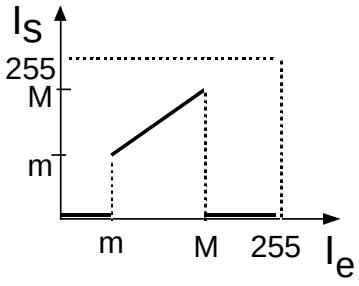


• Intérêt

Mise en œuvre de 2 seuils :

- les valeurs entre [0, ...,m[et]M,...,255] sont inchangées
- les pixels entre [m...M] sont mis à 255 (ces pixels sont « blancs »)

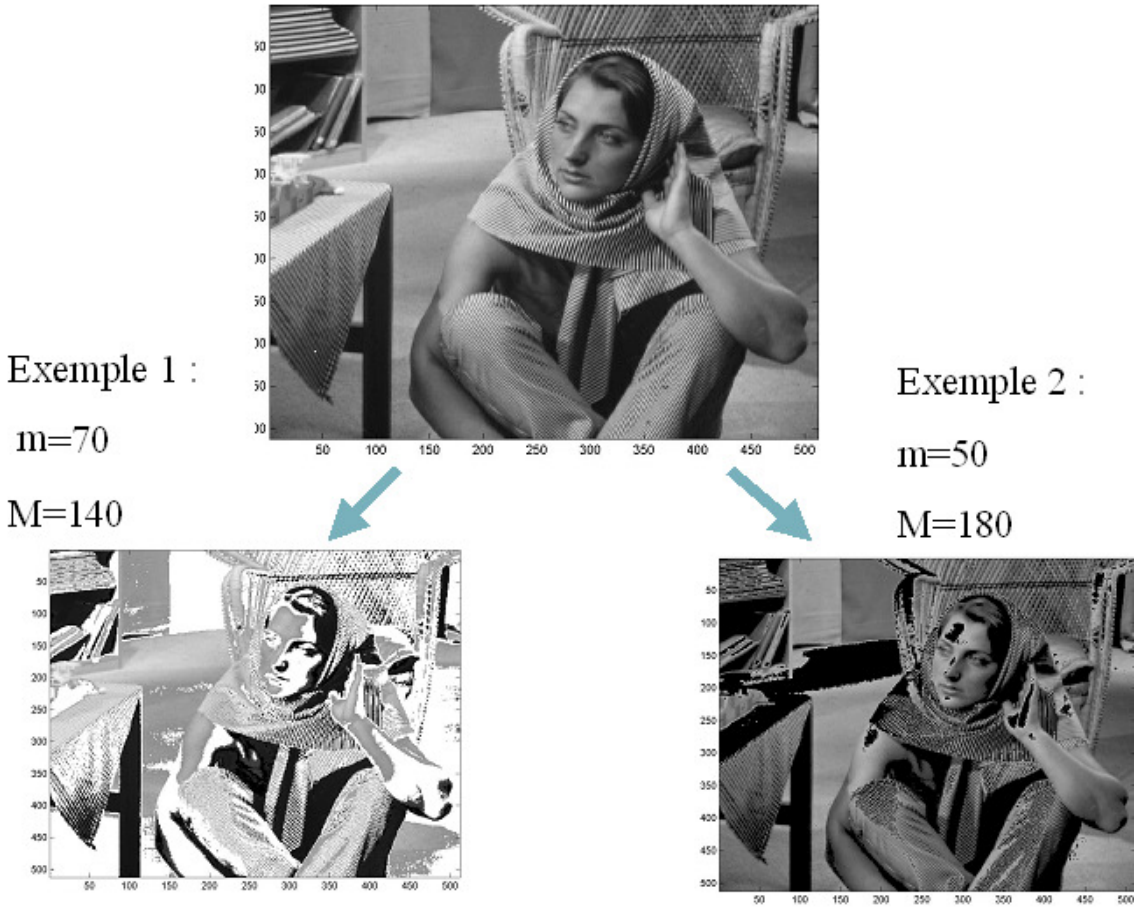
Autre transformation :



Deux autres exemples :

- figure du haut : tous les pixels de l'image d'entrée Ie dont les valeurs sont dans la gamme [m, M] sont mis à 255 (ils seront donc « blancs » à la visualisation). Pour les autres valeurs (supérieures à M ou inférieures à m) la valeur d'entrée demeure inchangée.
- figure du bas : la transformation « opposée » où seuls les pixels dont les valeurs sont appartiennent à l'intervalle [m, M] restent inchangés, les autres valeurs de pixels sont mises à 0 (« noirs »).

Ces deux cas sont des exemples de transformations continues par morceaux.



La figure ci-dessus présente, pour l'image *Barbara*, les résultats obtenus après application des LUTs continues par morceaux présentées précédemment.

Exercice Chapitre 2 – Outils de base du traitement d'images - LUT

Cet exercice est essentiellement un exercice d'observation (peu de programmation de votre part) durant lequel vous allez non seulement vous familiariser avec l'utilisation de Matlab pour le traitement d'image (ce qui vous aidera pour les pratiques suivantes) mais aussi apprendre à utiliser les outils de bases du traitement d'image.

Lancez Matlab et mettez à jour la liste des chemins dans le path browser.

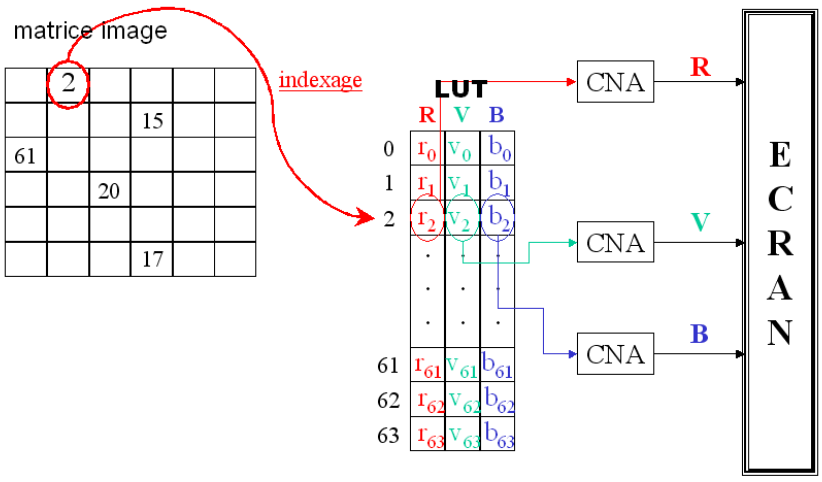
Création de LUT (*Look Up Table)*

1 – Début de session
À partir de la console Matlab, ouvrez un nouveau fichier « M-File » dans lequel vous allez saisir vos commandes et dont chaque ligne (se terminant par un « ; ») sera ensuite interprétée.

2 – Ouverture d'une image
Commencez par charger l'image *CLOWN_LUMI.BMP* en niveaux de gris avec ***imread*** (jetez un coup d'œil à l'aide). Observez le type des données.

3 – Affichage et trans-typage
L'affichage d'une image peut se faire avec les fonctions ***image***, ***imagesc*** et ***imshow***. Observez les différences, faites également un essai en trans-typant vos données.
(cf. Exercice « *Prise en main Matlab* » du chapitre 1 : essayez *image* = ***double***(*image*))

4 – Exemples de LUT
Pour l'affichage d'une image monochrome (tableau 2-D) Matlab utilise une LUT par défaut qui associe à chaque élément de la matrice image une couleur. Cette LUT par défaut présente 64 couleurs différentes en sortie (commande ***colormap*** pour afficher les niveaux de la LUT par défaut). Le fonctionnement est décrit sur la figure ci dessous.



Les différentes valeurs « i » de la matrice image vont être interprétées comme des index de la LUT par défaut. À chaque index « i » va correspondre une couleur créée par la combinaison r_i, v_i et b_i des couleurs primaires Rouge-Vert-Bleu. En effet, les valeurs r_i, v_i et b_i sont envoyées dans un Convertisseur Numérique Analogique (CNA) de façon à venir exciter respectivement les canons Rouge, Vert, et Bleu de l'écran. On obtient alors à l'affichage une couleur créée par la synthèse additive de trois couleurs primaires.

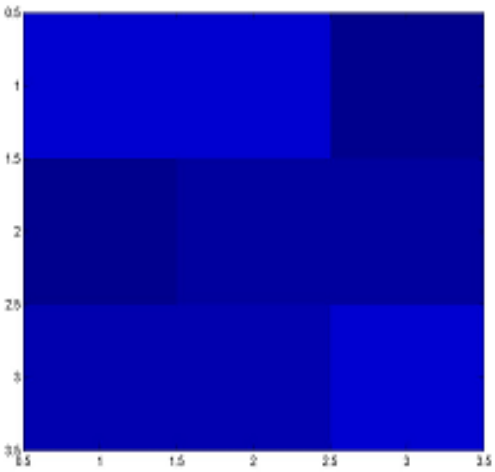
Sous Matlab, il est possible de créer ses propres LUT et de les appliquer à l'aide de la fonction ***colormap***. Les valeurs doivent cependant être normalisées sur l'intervalle [0, 1].
Prenons, par exemple, le cas simple d'une matrice M de taille 3×3 :

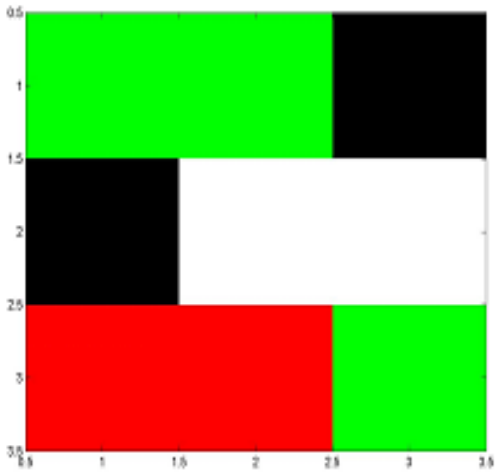
$$\begin{bmatrix} 5 & 5 & 1 \\ 1 & 2 & 2 \\ 3 & 3 & 5 \end{bmatrix}$$

Cette matrice comprend quatre valeurs distinctes. On se propose d'afficher : « 1 » en noir, « 2 » en blanc, « 3 » en rouge, et « 5 » en vert. Pour cela on crée une LUT « map4C » dont les sorties sont les quatre couleurs souhaitées. Pour appliquer cette LUT à l'image affichée, on tape la commande ***colormap(map4C)*** :

```
M=[5 5 1;1 2 2;3 3 5] ;
% Définition de la LUT
r=[0 1 1 0];
v=[0 1 0 1];
b=[0 1 0 0];
map4C=[r' v' b'];
image(M)
colormap(map4C)
```

Résultats avant et après l'utilisation de la commande ***colormap()*** :

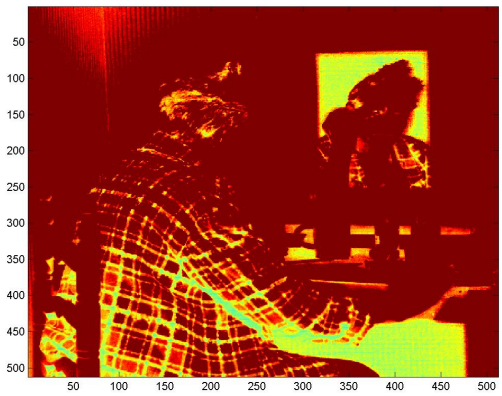




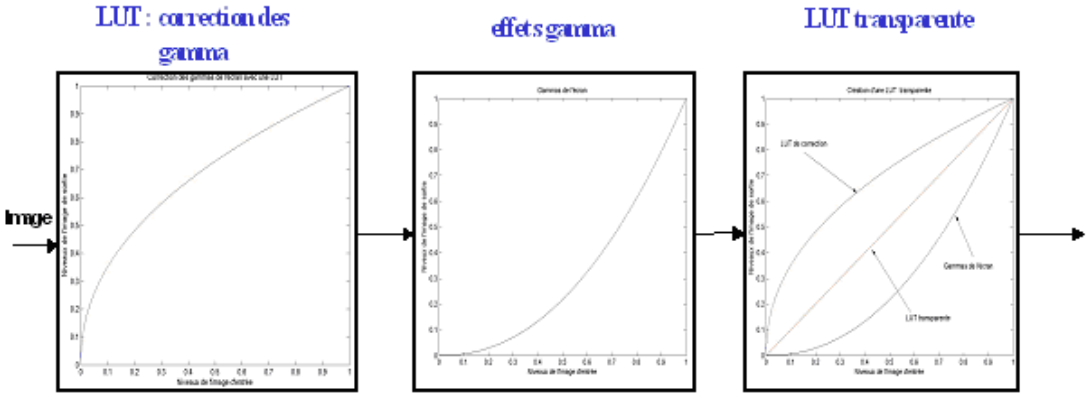
Pour afficher une image monochrome en niveau de gris, il faut donc utiliser une LUT dont les couleurs en sortie ne sont que des nuances de gris.
Dans ce cas : $\forall i \in [0, 255], r_i = v_i = b_i$ (généralement la valeur de luminance d'un pixel d'une image monochrome est codée sur 8 bits, donc 256 niveaux d'intensité possibles).

ACTION : Rapatriez le script ***lutndg.m*** dans votre répertoire de travail. Ce script sert à créer une LUT qui permet d'afficher une image monochrome en niveaux de gris (ndg). Après avoir ouvert et analysé ce fichier script, affichez une image monochrome de votre choix et tapez la commande ***lutndg.m***.

Voici les images avant et après utilisation de la LUT contenue dans le script ***lutndg.m*** :



On propose un autre exemple de création de LUT sous Matlab : un écran d'affichage génère des « effets gamma » qu'il est possible de compenser (transformation non-linéaire des niveaux de l'image d'entrée). Pour cela, on crée la LUT inverse à celle des effets gamma de l'écran afin que la composée « LUT correction o effets gamma » constitue une LUT transparente.

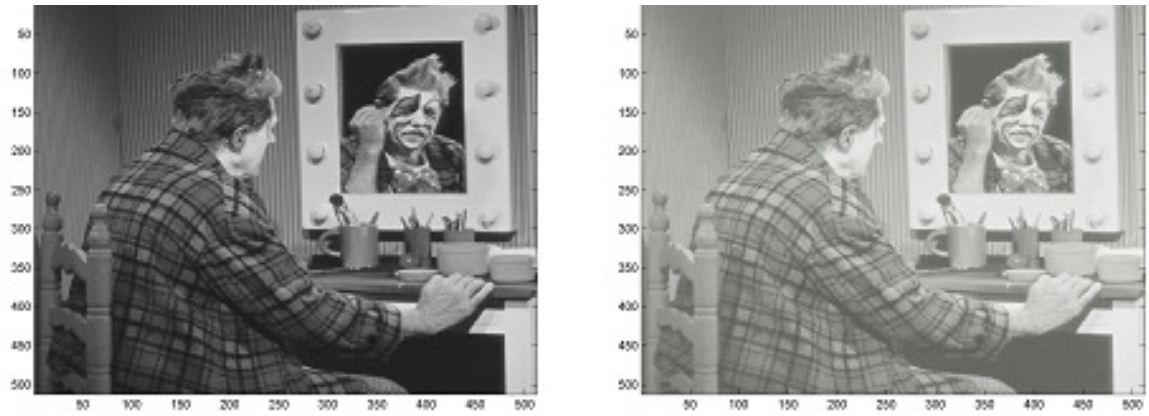


Voici un exemple de script pour comparer l'image avant et après compensation du gamma de l'écran :

```
I = imread('CLOWN_LUMI.BMP');
% LUT pour afficher l'image clown_lumi en niveaux de gris
r=0:1/255:1;
v=0:1/255:1;
b=0:1/255:1;
map=[r' v' b'];
image (I);
colormap(map)
% LUT si on souhaite compenser le gamma de l'écran
vect = 0:1/255:1;
gamma_r = 2.2;        % Niveau_sortie=Niveau_entrée^2.2 pour le rouge
gamma_v = 2.3;        % Niveau_sortie=Niveau_entrée^2.3 pour le vert
gamma_b = 2.1;        % Niveau_sortie=Niveau_entrée^2.1 pour le bleu
r = vect.^(1/gamma_r);
v = vect.^(1/gamma_v);
b = vect.^(1/gamma_b);
```

```
% on fabrique la LUT à l'aide des 3 vecteurs
map_gamma_inv = [r' v' b'];
% pour appliquer la LUT
figure
image(I)
colormap(map_gamma_inv);
```

Voici les résultats obtenus respectivement avant (image de gauche) et après (image de droite) l'utilisation d'une LUT pour compenser le gamma de l'écran:



ACTION : En vous inspirant des exemple précédents, synthétisez une LUT pour faire une inversion vidéo de l'image *CLOWN_LUMI*. Ce traitement consiste à réaliser le négatif (bien connu en photographie) de chaque plan de couleurs i.e. les niveaux 0 deviennent 255 et inversement, les niveaux 1 deviennent 254, ...

5 – Affichage des plans R-V-B

Chargez l'image couleur *CLOWN* dans votre répertoire de travail. Observez le type des données, et visualisez l'image. Visualisez plan par plan l'image couleur *CLOWN* en créant les LUTs adéquates pour les plans Rouge, Vert, et Bleu (on s'inspirera de l'exercice « *Éclatement d'une image couleur sous Matlab* » du chapitre 1).

Correction de l'exercice sur les LUT

1 – Ouvrez l'éditeur de commande par la méthode de votre choix (cf. exercice de prise en main Matlab, Chapitre 1).

2 – Dans votre répertoire de travail exécutez la commande:

```
I = imread('CLOWN_LUMI.BMP');
```

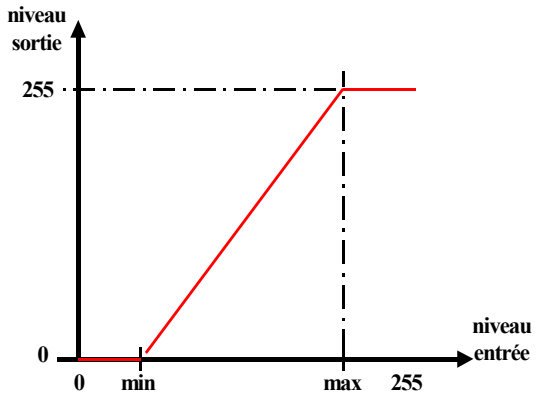
I est une matrice bidimensionnelle dans le cas d'une image en niveaux de gris. Le type de données de I est uint8 (unsigned integer).

3 – Il existe trois fonctions sous Matlab pour afficher des images : ***imshow***, ***image***, et ***imagesc***.

+ ***imshow*** affiche l'image contenue dans le fichier image ou la matrice correspondante. ***imshow*** appelle ***imread*** pour lire l'image depuis le fichier, mais les données de l'image ne seront pas stockées dans le workspace.

+ ***image*** affiche la matrice en argument comme une image. Elle n'accepte pas de fichier image en paramètre.
L'affichage repose sur les valeurs de la matrice ainsi que sur la carte des couleurs activées.

+ ***imagesc*** affiche la matrice en argument comme une image.
Contrairement à ***image***, elle effectue une remise à l'échelle des valeurs de la matrice avant l'affichage de manière à utiliser pleinement la carte des couleurs. Elle utilise donc la LUT suivante :



Afin de comparer visuellement ces trois fonctions, vous pouvez également taper la commande ***figure*** avant chacune des commandes ***image***, ***imagesc***, et ***imshow***. Ainsi Matlab ouvrira une nouvelle fenêtre d'affichage pour chaque nouvelle image.

```
% Visualisation des différences entre les fonctions d'affichage
figure
image(I) ;
figure
imagesc(I)
figure
imshow(I)
```

Exemple de trans-typage :

```
I = imread('CLOWN_LUMI.BMP');
J = double(I);
image(J);
```

4 – Voici un exemple de solution pour réaliser le négatif de l'image *CLOWN_LUMI* avec une LUT :

```
% Lecture des images
Im = imread('CLOWN_LUMI.BMP');
% Création de le LUT
r=1:-1/255:0;
v=1:-1/255:0;
b=1:-1/255:0;
lut=[r' v' b'];
image(Im)
colormap(lut)
```

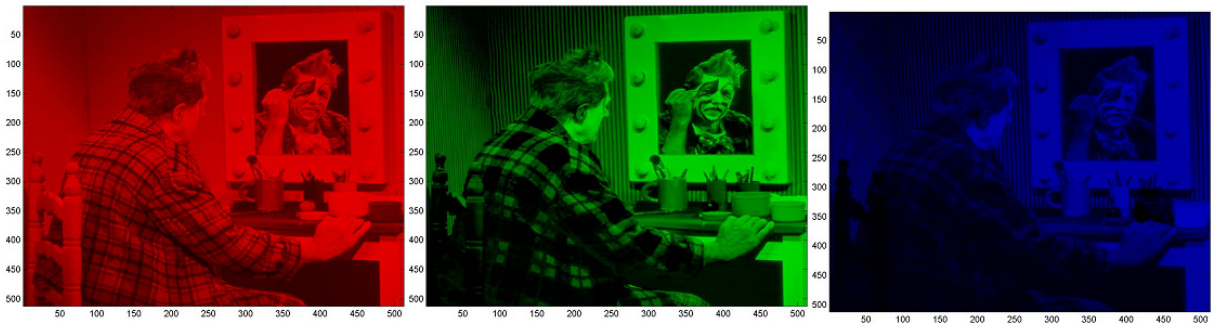
Voici les résultats obtenus :

Avec une LUT adéquate, il est donc possible d'afficher le négatif d'une image sans stocker aucune donnée supplémentaire de type « image » dans le workspace et sans modifier les données initiales de l'image.

5 – Dans le cas d'une image couleur la donnée est tri-dimensionnelle et de type uint8. Les trois plans d'une image couleur sont des plans monochromes. Pour les visualiser il faut créer des LUTs adéquates. Par exemple, pour visualiser le plan rouge, il faut créer une LUT dont les sorties ne soient que des niveaux de rouge.
Voici les commandes pour visualiser les 3 plans R-V-B de l'image couleur *CLOWN* :

```
Im=imread('CLOWN.BMP') ;
% Création des LUTs
vecteur=0:1/255:1;
r=vecteur;
v=vecteur*0;
b=vecteur*0;
lutr=[r' v' b'];
r=vecteur*0;
v=vecteur ;
lutv=[r' v' b'];
v=vecteur*0;
b=vecteur ;
lutb=[r' v' b'];
% Affichage des plans R-V-B
image(Im(:,:,1));       % Plan Rouge
colormap(lutr);
figure
image(Im(:,:,2));       % Plan Vert
colormap(lutv);
figure
image(Im( :, :,3));     % Plan Bleu
colormap(lutb);
```

Voici les résultats obtenus :



Remarque : Matlab est un outil de visualisation de données matricielles qui propose des LUT spécifiques. Ces LUT sont présentées à la rubrique « ***Supported Colormaps*** » de l'aide Matlab sur les ***colormap***. Notons cependant qu'il n'existe pas de LUT sous Matlab pour les images couleurs : il n'est pas possible de visualiser une image couleur en créant une LUT pour chacun des plans Rouge, Vert, Bleu.

Exemple d'histogramme pour une image en niveaux de gris

- Pour chaque niveau de gris, compter le nombre de pixels s'y référant
- Pour chaque niveau, tracer le graphe en bâton du nombre de pixels

(possibilité de regrouper les niveaux proches en une seule classe)

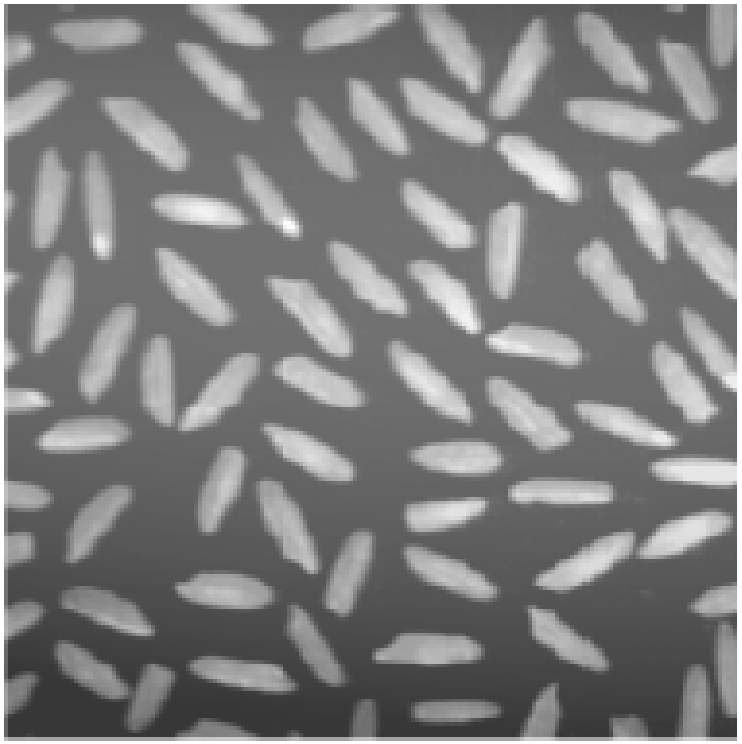
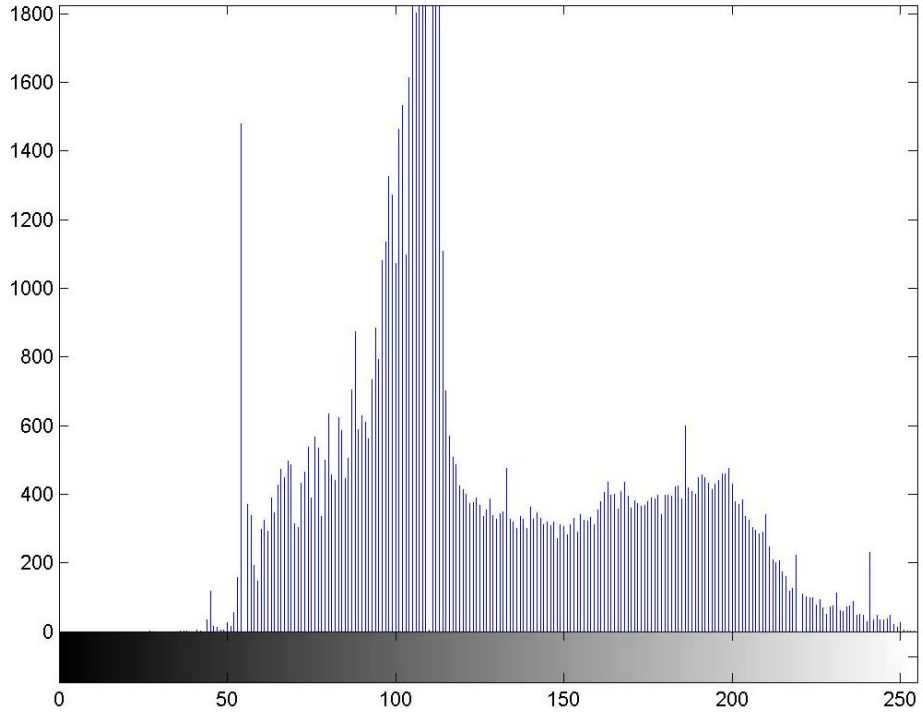


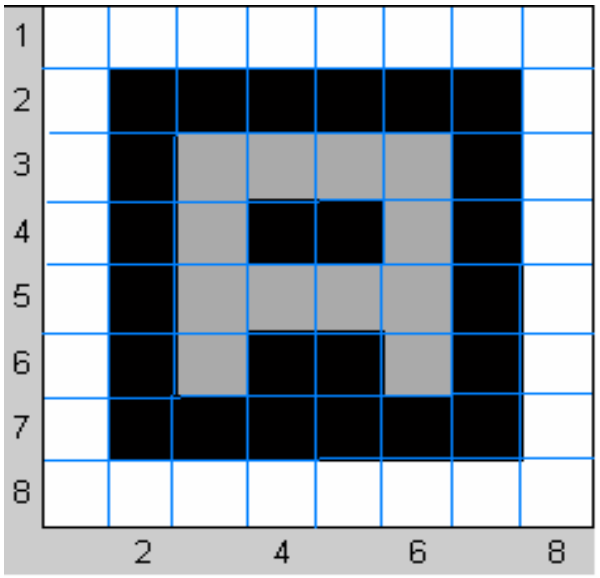
Image de 256 × 256 pixels, codés chacun sur 8 bits



Population de pixels pour chaque niveaux de gris [0 ; 255]

Exemple simple de calcul d'histogramme pour une image

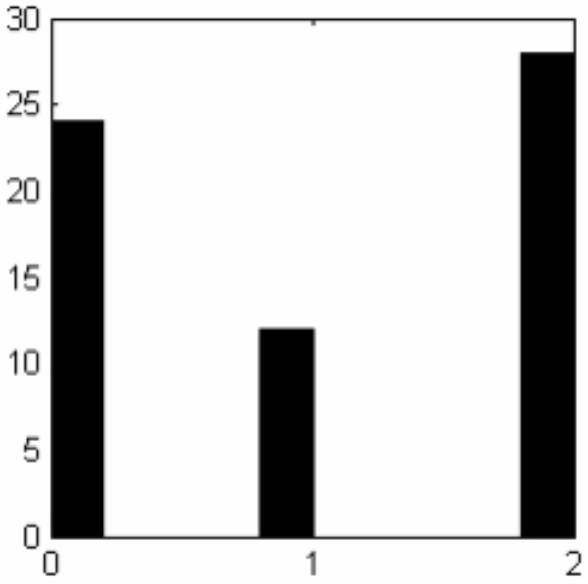
Image « A » en niveaux de gris



Matrice des valeurs de luminance des pixels de l'image « A »

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

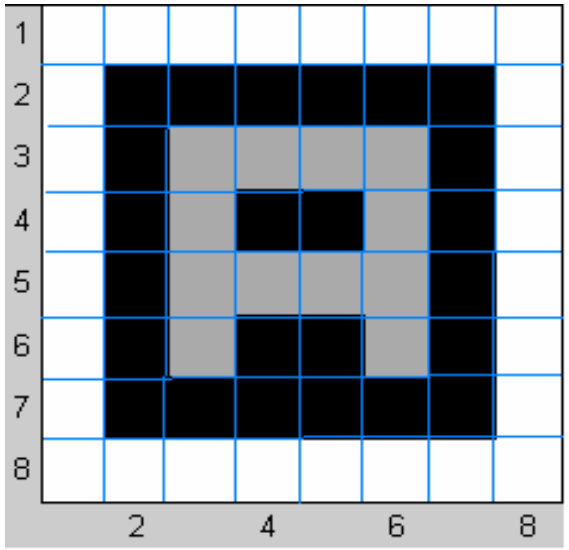
Histogramme de l'image « A »



- L'image « **A** » comporte 3 niveaux de gris différents : **0**, **1** et **2**.
- Compter le nombre de pixels pour chaque niveau de gris, à l'aide de la matrice des valeurs de luminance.
- Les niveaux **0**, **1** et **2** sont respectivement représentés par **24**, **12** et **28** pixels ⇒ représentation de cette population de pixels sur l'histogramme.

Histogramme cumulé d'une image

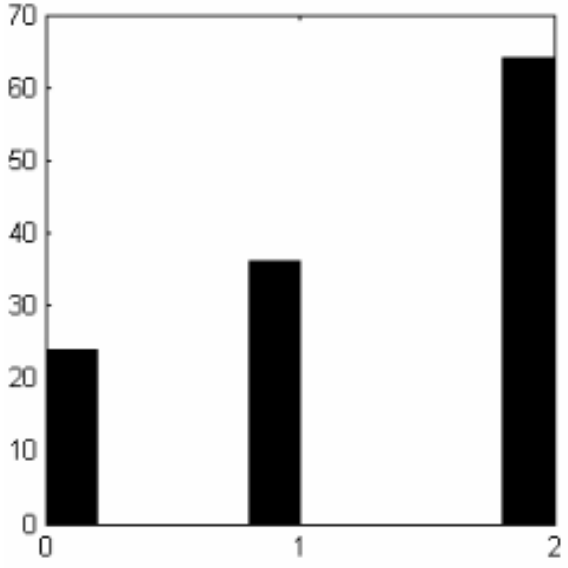
Image « A »



Valeurs de luminance de « A »

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

Histogramme cumulé de « A »



- Calcul d'un histogramme particulier faisant appel aux cumuls des niveaux de gris ⇒ **Histogramme cumulé**.
- Chaque bâton cumule le nombre de pixels du niveaux de gris concerné et des niveaux de gris inférieurs : les niveaux **0**, **1**, **2** sont donc représentés respectivement par **24**, **36** et **64** pixels.
- Utile pour certains traitements d'image tels que **l'égalisation d'histogramme** (⇒ amélioration de contraste).

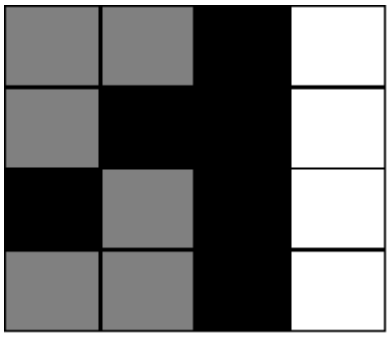
Exercice Chapitre 2 – Calcul d'histogrammes

Cet exercice va vous familiariser avec l'utilisation de la fonction Matlab ***imhist*** qui permet d'afficher l'histogramme d'une image. Un cas simple sur une image de faible taille est présenté afin de bien assimiler la notion d'histogramme. On vous propose ensuite de comparer les histogrammes de différentes images (monochromes et couleurs).

Histogramme

1 - Utilisez l'aide de Matlab (commande ***help*** ou ***helpwin***) pour connaître le rôle de la fonction ***imhist***.

2 – Créez la matrice *im1* qui représente l'image suivante :



avec les valeurs :
- 0 pour un pixel noir ;
- 127/255 pour un pixel gris ;
- 1 pour un pixel blanc.

Vérifiez que vos coefficients sont exacts en affichant *im1* avec la fonction ***imagesc*** (vous pouvez également créer une LUT pour obtenir un affichage en niveaux de gris. Pour appliquer cette LUT, utilisez la fonction ***colormap***).
Affichez et commentez l'histogramme de ***im1*** obtenu avec la commande ***imhist(Im1,3)***.

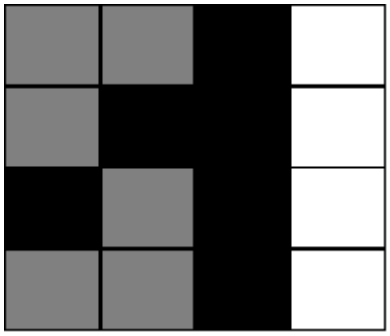
3 – Chargez les deux images monochromes « *FRUIT_LUMI* », et « *ISABE_LUMI* » dans votre répertoire de travail. Ouvrez ces deux images sous Matlab avec la fonction ***imread***. Visualisez et comparez les histogrammes avec la fonction ***imhist***.

4 – Chargez et ouvrez l'image couleur « *MANDRILL* ». Visualisez les histogrammes des différents plans R-V-B de cette image, en utilisant la fonction ***imhist*** plan par plan.

Correction de l'exercice : Calcul d'histogrammes

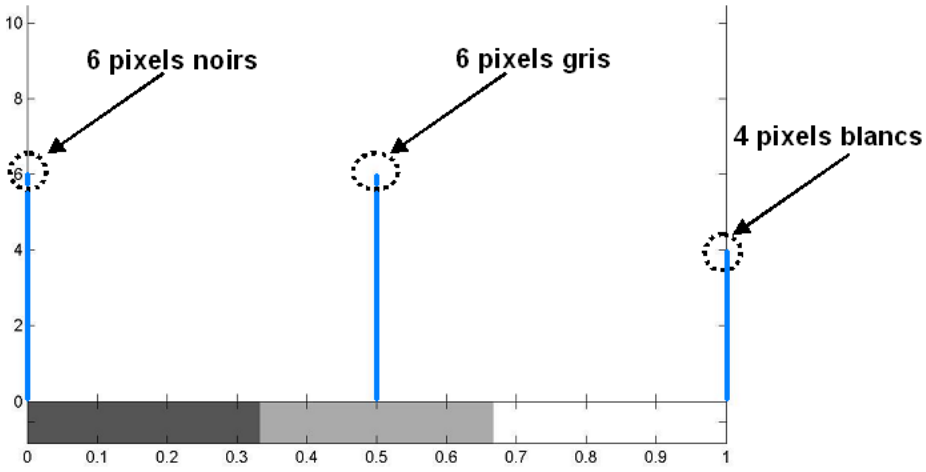
1 - La commande ***imhist*** permet de calculer et d'afficher l'histogramme des images en niveaux de gris. Pour les images couleurs la commande ***imhist*** est à utiliser plan par plan.

2 – Voici les commandes pour afficher l'histogramme de l'image 4×4 :



```
im1=[127/255 127/255 0 1;127/255 0 0 1;0 127/255 0 1; 127/255 127/255 0 1]
% Création de la LUT pour afficher en niveau de gris
r=[0 0.5 1];
v=[0 0.5 1];
b=[0 0.5 1];
map=[r' v' b'];
Création de la LUT pour afficher en niveau de gris
imagesc(im1);
colormap(map) ;
Affichage de l'histogramme
imhist(im1,3) ;
```

L'histogramme obtenu est le suivant :

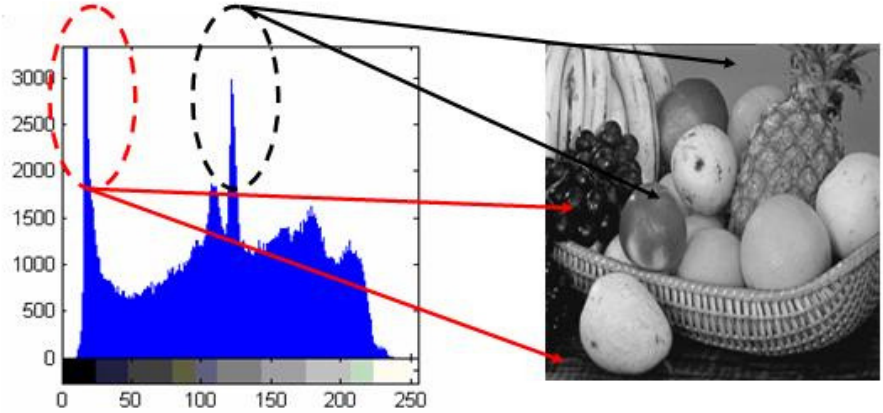


L'histogramme affiche le nombre de pixel pour chaque niveau de gris de l'image, il y'a donc 6 pixels noirs, 6 pixels gris et 4 pixels blancs (résultat facilement observable sur l'image 4×4 utilisée).

3 – Nous utilisons les images en niveaux de gris *FRUIT_LUMI.BMP* et *ISABE_LUMI.BMP*. Après avoir lu les fichiers images par la commande ***imread,*** nous calculons leurs histogrammes.

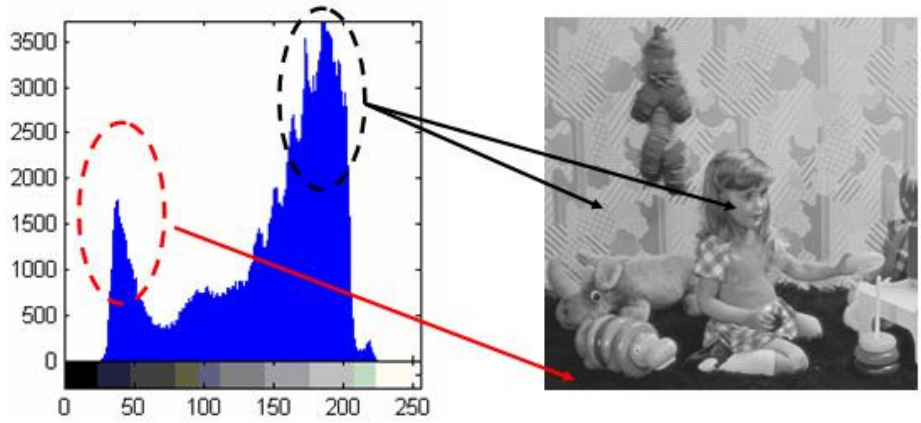
```
I = imread('FRUIT_LUMI.BMP') ;
J = imread('ISABE_LUMI.BMP') ;
imhist(I);
imhist(J);
```

Les deux pics de populations de pixels dans l'histogramme de l'image *FRUIT_LUMI* correspondent aux régions les plus sombres de l'image (sol, raisin, …) et aux régions de gris moyen (intensité proche de 127 pour les pommes, le fond, …).



Histogramme et image FRUIT_LUMI

Voici l'histogramme obtenu pour l'image *ISABE_LUMI* :

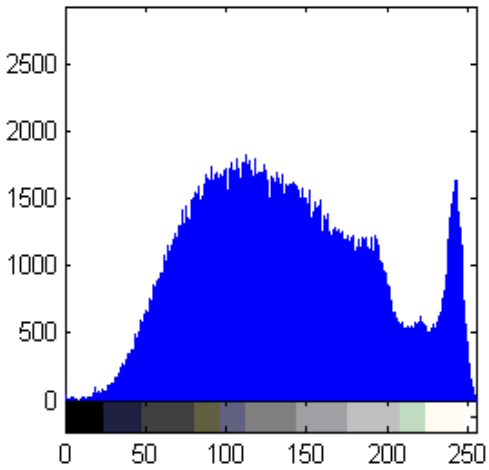


Histogramme et image ISABE_LUMI

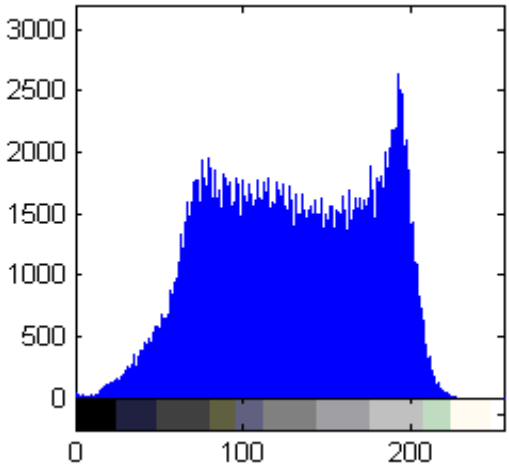
Dans ce cas les deux principaux pics de population de pixels correspondent au sol (partie la plus foncée) et à l'arrière plan de l'image (partie la plus claire).

4 –Pour les images couleurs, notamment l'image *MANDRILL.BMP,* on lit le fichier image avec la fonction ***imread*** puis on calcule son histogramme plan par plan.

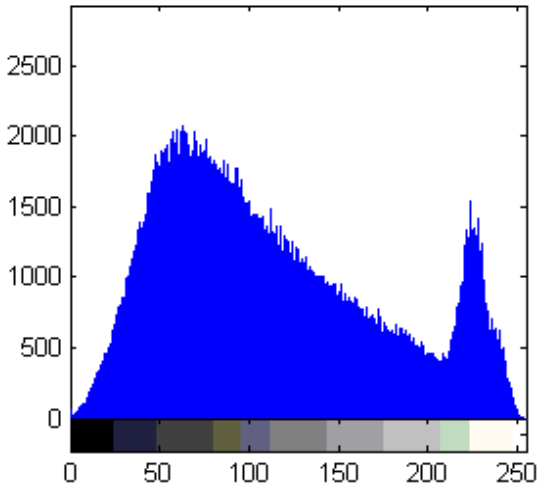
```
I = imread('MANDRILL.BMP') ;
imhist(I(:,:,1));  % 1er plan : Rouge
imhist(I(:,:,2));  % 2ème plan : Vert
imhist(I(:,:,3));  % 3ème plan : Bleu
```



MANDRILL.BMP (Plan1 : Rouge)



MANDRILL.BMP (Plan 2 : Vert)



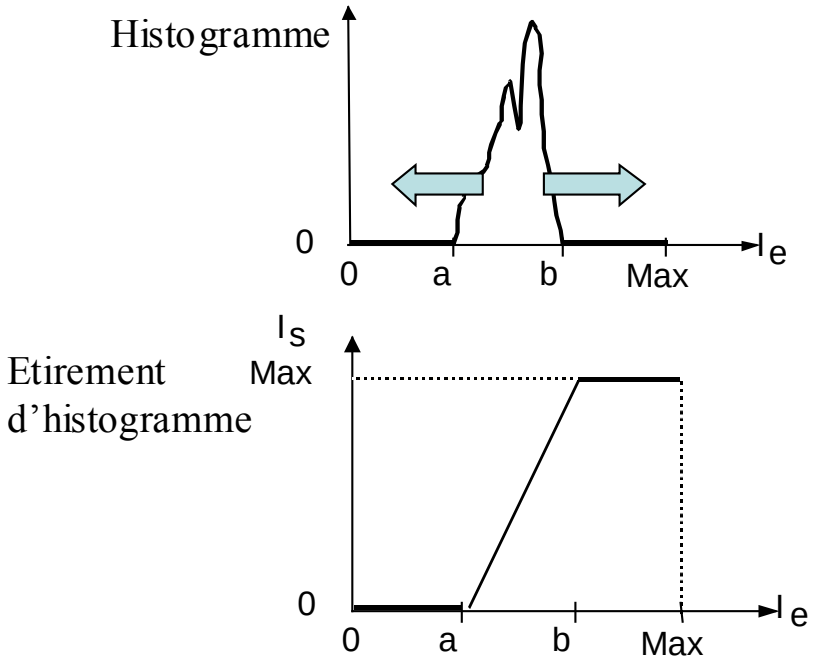
MANDRILL.BMP (Plan 3 : Bleu)

Chapitre 2

Notions de traitement d'images

Transformation d'histogramme

Augmentation du contraste par étirement d'histogramme

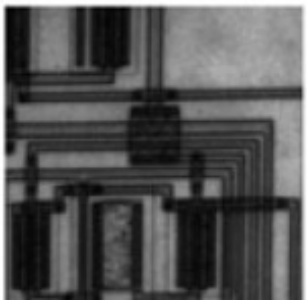


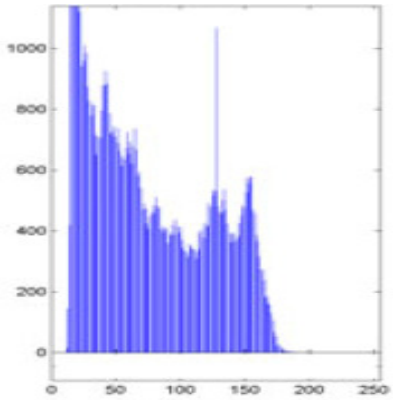
- <u>Étirement d'histogramme</u> :

Cette première transformation sur l'histogramme a pour objet l'augmentation du contraste d'une image. Pour cela, il convient d'augmenter sur l'histogramme (cf. figure du haut) l'intervalle [a,b] de répartition des niveaux de gris de l'image d'entrée « Ie ». On parle alors d'étirement d'histogramme. Du point de vue de la transformation (cf. figure du bas), un étirement maximal est réalisé dès lors que la répartition des niveaux de gris de l'image de sortie « Is » occupe l'intervalle maximal possible [0, Max]. Typiquement pour une image dont les niveaux sont codés sur 8 bits, l'intervalle [a, b] de Ie sera étiré jusqu'à l'intervalle [0, 255] pour Is.

Exemple d'augmentation du contraste

Image originale





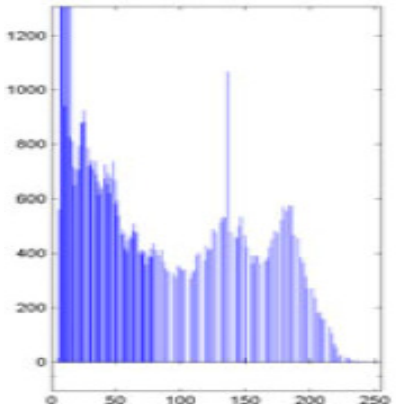
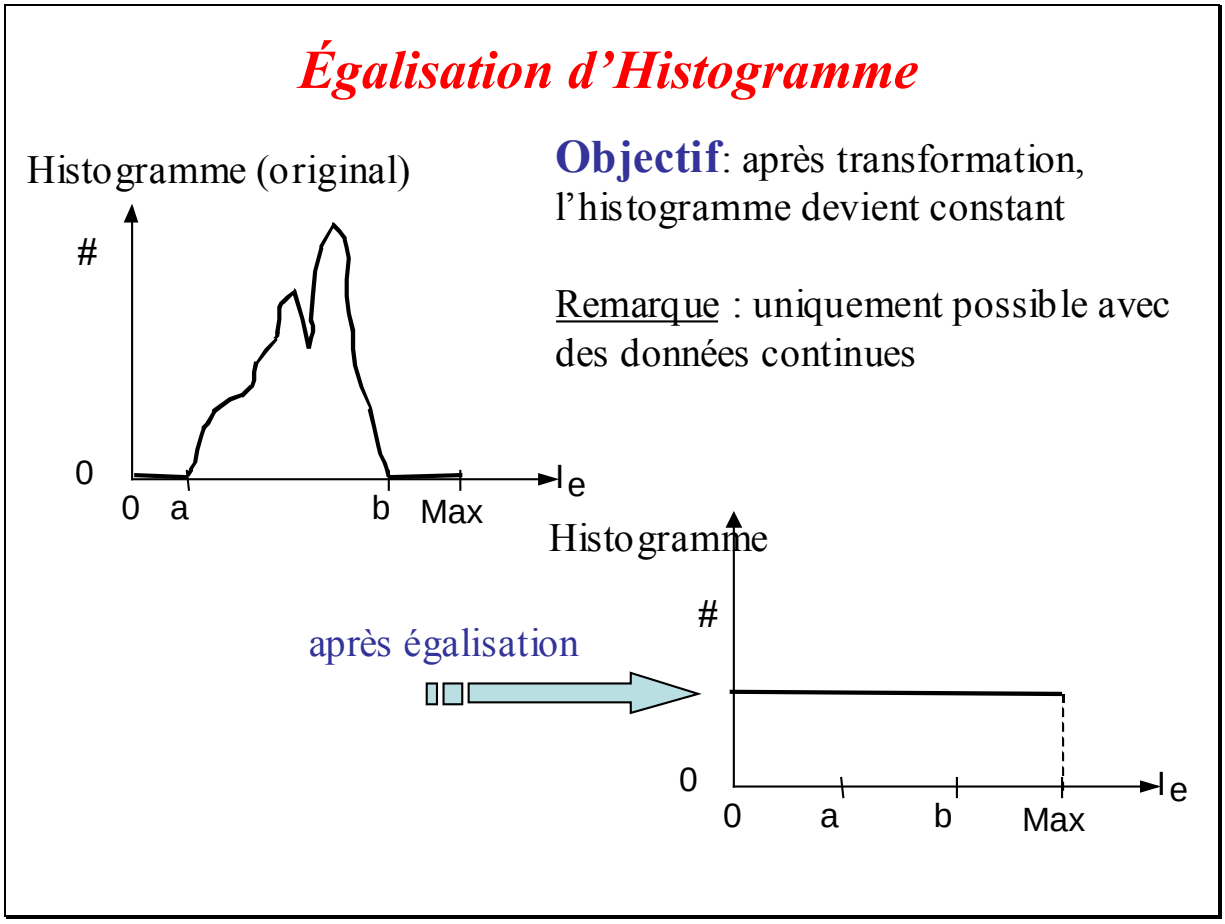


Image contrastée

La figure illustre l'étirement d'histogramme sur l'image '*Circuit*'. L'intervalle de l'image originale Ie est [12, 182]. Après étirement de l'histogramme, la répartition des niveaux de gris s'effectue sur l'intervalle [0, 255] et donc concerne toute l'échelle des niveaux de gris codés sur 8 bits. L'image obtenue après étirement possède un meilleur contraste. Le contenu de l'image relatif à des structures de circuits électroniques est mis en évidence.



- Égalisation d'histogramme :

La deuxième transformation que nous abordons maintenant a pour objet également l'augmentation du contraste d'une image. Il comprend l'étirement d'histogramme présenté précédemment avec en plus une répartition uniforme des niveaux de gris. Après transformation, l'histogramme devient constant : chaque niveau de gris est représenté dans l'image par un nombre constant de pixels. On parle aussi d'histogramme « plat ». Cette transformation n'est en théorie possible que dans la mesure où l'on dispose de données continues. Or le domaine spatial et, surtout, l'échelle des niveaux de gris sont des données discrètes. Dans la pratique donc, l'histogramme obtenu ne sera qu'approximativement constant.

Effet de la discrétisation sur l'égalisation

Observation : si l'image originale Ie comportent k niveaux de gris (avec k << Max), l'image de sortie Is en comportera au plus k;

Techniques : nous définissons l'histogramme cumulé d'une image Ie comme la fonction C_{Ie} sur [0, Max], avec des valeurs entières positives.

En particulier, nous avons $C_{Ie}(Max) = N$
où N est le nombre total de pixels dans Ie.

La fonction f qui réalise l'égalisation i.e. Is = f(Ie) est donnée par :

$$f(g) = Max \,.\, C_{Ie}(g) / N \qquad \text{(valeur entière arrondie)}$$

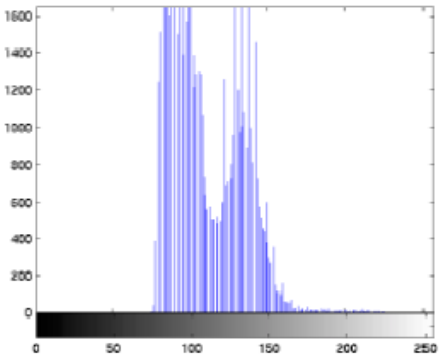
en particulier, nous avons f(Max) = Max.

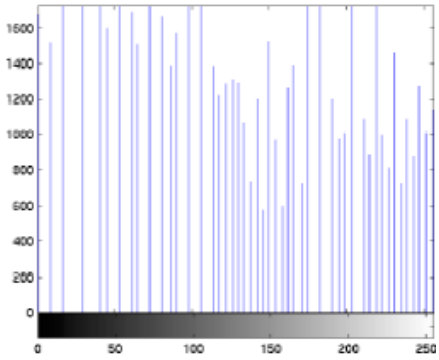
Afin de procéder à l'égalisation, nous avons recours à ce qui est appelé un histogramme cumulé. Cet histogramme particulier comptabilise pour un niveau de gris « g » donné le nombre de pixel ayant un niveau de gris **e** inférieur ou égal à **g**, ce nombre est noté $C_{Ie}(g)$. Sachant que « N » est le nombre total de pixels dans l'image Ie, $\frac{C_{Ie}(g)}{N}$ est donc la proportion du nombre de pixels de Ie ayant un niveau de gris inférieur ou égal à g. Le niveau de gris « f(g) » après égalisation sera alors la fraction de Max correspondant à cette proportion. Cette fraction est arrondie à l'entier le plus proche.
Dans ce cas, il est tout a fait possible que, pour deux niveaux de gris de départ g et g', on obtienne f(g) = f(g') et donc un nombre de niveaux de gris dans l'image Is inférieur à « k », le nombre de niveaux de gris dans l'image d'origine Ie.

Effet de l'égalisation d'histogramme

Image originale et son histogramme

Image et son histogramme après égalisation





Ce premier exemple d'égalisation permet visuellement de rehausser les contrastes de l'image. L'histogramme obtenu après égalisation s'étale bien sur toute l'échelle des niveaux de gris avec un espacement accru. Les données discrètes des niveaux de gris ne permettent pas d'obtenir un histogramme rigoureusement plat.

Augmentation du contraste :
exemples et comparaison

Original - Étirement d'histogramme - Égalisation

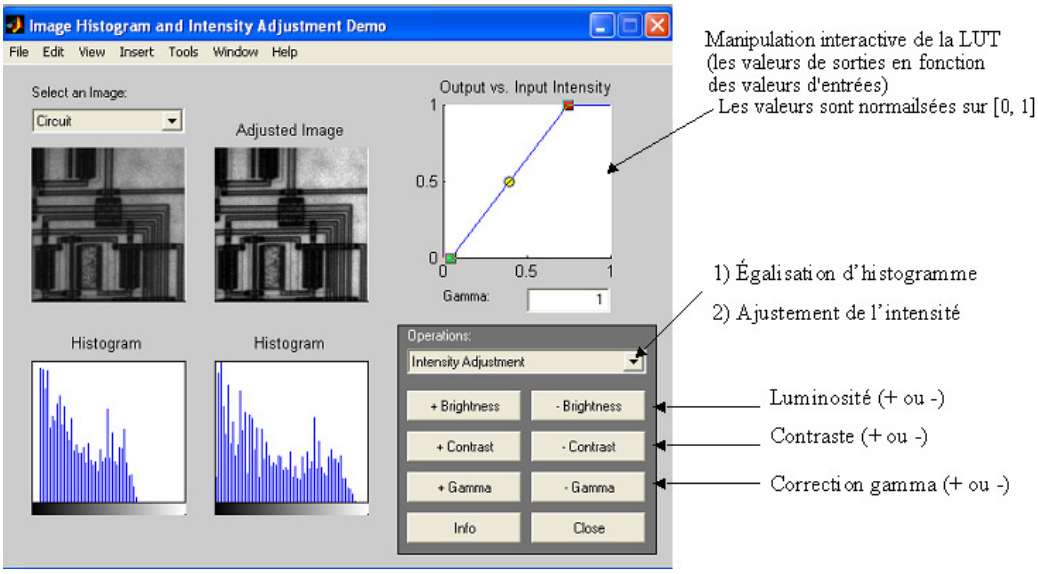
Ce deuxième exemple est accompagné du résultat d'un simple étirement d'histogramme pour effectuer la comparaison des deux transformations. Le rehaussement du contraste est plus marqué avec l'égalisation d'histogramme autorisant la détection de structures situées dans l'ombre. En fait, tout niveau de gris fortement représenté est étiré à l'inverse tout niveau de gris faiblement représenté est fusionné avec d'autres niveaux proches.

Exercice Chapitre 2 – Transformation d'histogrammes

Cet exercice est avant tout un exercice d'observation. Il s'agit ici d'utiliser la démonstration ***imadjdemo*** de Matlab pour apprécier de façon interactive les effets sur les images de différentes transformations : modification de LUT, ajustement de contraste, modification d'histogramme, …

Ajustement de contraste et Histogramme

À partir de la console Matlab, lancez la démonstration en tapant simplement la commande ***imadjdemo***. Cette fenêtre s'affiche :



Cette démonstration permet de faire de manière interactive des ajustements de luminosité, de contraste, de correction gamma et d'égalisation d'histogramme. Vous pouvez alors visualiser les LUT appliquées ainsi que les effets sur les images et les histogrammes.

Remarque : lorsque vous modifiez la visualisation d'une image avec une LUT, les données sources restent inchangées.

1 – Choisissez l'image « *Circuit* ». Modifiez la luminosité. Interprétez les résultats.

2 – De même, modifiez le contraste et la correction gamma. Interprétez les résultats. Essayez de réaliser ces modifications sur d'autres images que l'image *Circuit* et vérifiez vos interprétations.

Égalisation d'histogramme

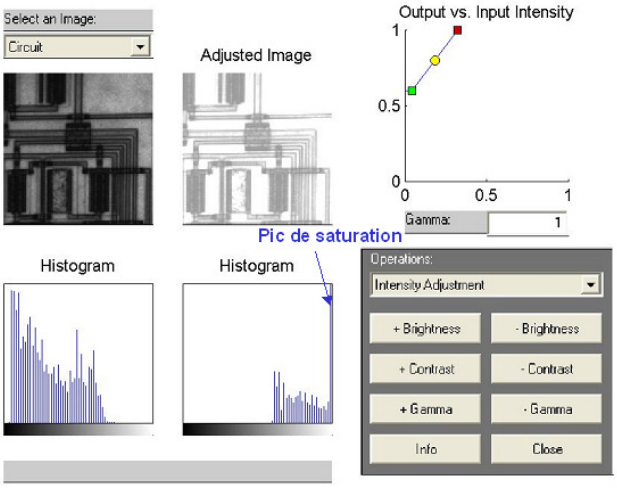
3 – Toujours en utilisant la démonstration ***imadjdemo***, effectuez une égalisation d'histogramme. Appréciez l'effet sur l'image et sur l'histogramme.

4 – Dans votre fenêtre de commande Matlab, allez dans l'aide pour connaître le rôle de la fonction ***histeq***. Chargez l'image *CLOWN_LUMI* dans votre répertoire de travail et égalisez son histogramme avec la commande ***histeq***. Affichez et comparez les images et les histogrammes avant et après égalisation.

Correction de l'exercice : Transformation d'histogramme

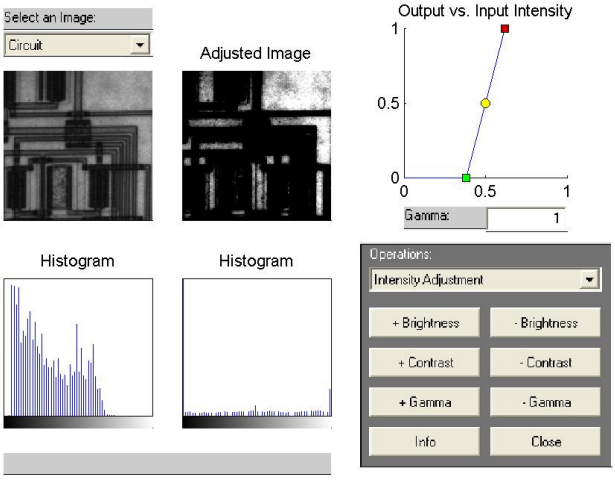
Nous allons voir comment se transforme une image lorsque l'on manipule les différents paramètres tels que la luminosité, le contraste, et les corrections gamma. La démonstration ***imadjdemo*** de Matlab permet de modifier ces paramètres de façon interactive sur des images types et de visualiser les différences entre l'image originale et l'image transformée. Cette démonstration présente également l'évolution de l'histogramme et de la LUT de l'image, on peut donc interpréter les résultats obtenus sur l'image finale.

1 – Voici l'image *Circuit* avant et après modification de la luminosité :



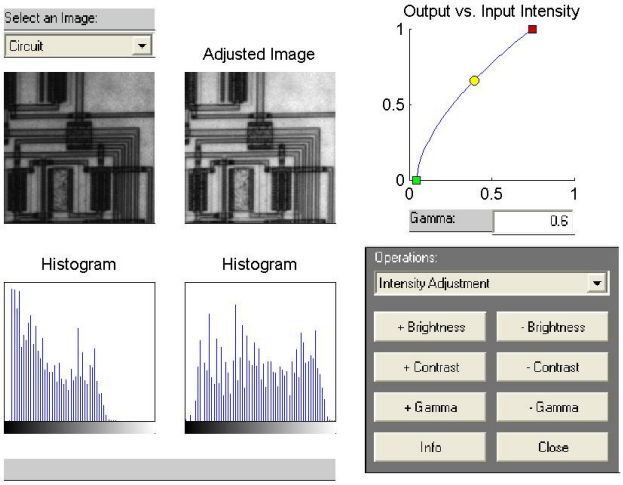
En augmentant la luminosité, on remarque que la LUT évolue de façon à ce que les pixels de très faibles intensités soient ramenés à une intensité beaucoup plus importante et que les pixels d'intensité moyennes et fortes sont tous saturés en sortie au niveau maximal d'intensité : 1 (plus la luminosité est forte, plus il y a de pixels de faible intensité saturés à 1). Sur l'histogramme, on aperçoit donc une forte concentration de population de pixels autour des hautes intensités (luminances) et un pic de population pour le niveau 1 provoqué par la saturation. L'image apparaît donc « éclaircie ». En diminuant la luminosité, on observe les phénomènes inverses et l'image est donc « noircie ».

2 – En manipulant le contraste, on obtient le résultat suivant :



En augmentant le contraste, on remarque sur la LUT que les pixels de faible intensité en entrée sont saturés à 0 et qu'inversement les pixels de forte intensité sont saturés à 1. Pour l'intervalle des pixels non saturés, la répartition des niveaux de gris s'effectue sur l'intervalle entier [0, 1] de manière linéaire. L'histogramme est donc étiré, et présente deux pics de population de pixels : un pic de population de pixel à 0 qui correspond à la saturation des pixels de faible intensité, et un pic de population de pixel à 1 qui correspond à la saturation des pixels de forte intensité. En revanche, pour les autres niveaux, on a exploité linéairement la dynamique. L'image obtenue après étirement possède donc un contraste plus accentué (pour les zones non saturées).

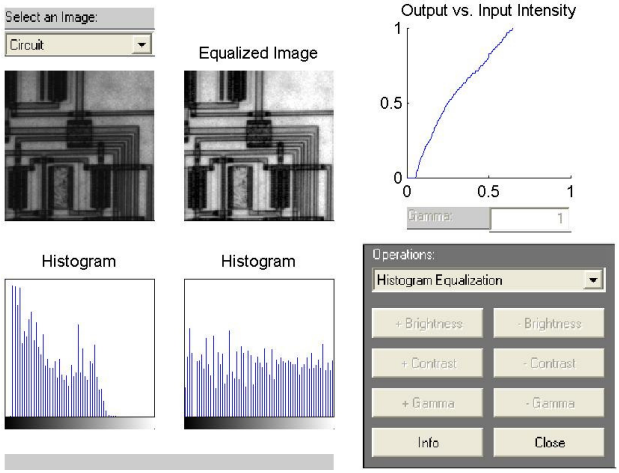
En modifiant les corrections gamma, on obtient le résultat suivant :



En imagerie, le facteur ***gamma*** modélise la non linéarité de la reproduction de l'intensité lumineuse. De fait, un tube cathodique est naturellement non-linéaire : l'intensité lumineuse reproduite à l'écran est une fonction non-linéaire de la tension d'entrée. la ***correction gamma*** peut être considérée comme un procédé permettant de compenser ce phénomène pour obtenir une reproduction fidèle de l'intensité lumineuse.

Les effets ***gamma*** sont modélisés par des fonctions du type $f(x)=x^{\gamma}$, où « x » est la valeur de luminance en entrée et γ un réel qui varie entre 2 et 2,5 dans le cas de la télévision. Ainsi, pour compenser ces effets, on applique une LUT de la forme $g(x)=x^{1/\gamma}$. Les pixels de faibles intensités dans l'image d'origine sont alors amenés à des intensités plus élevées et leur dynamique est augmentée. On détecte donc plus facilement que dans l'image d'origine les détails situés dans les zones sombres : le contraste est augmenté pour ces zones de faibles intensités.

3 – En égalisant l'histogramme, on obtient le résultat suivant :

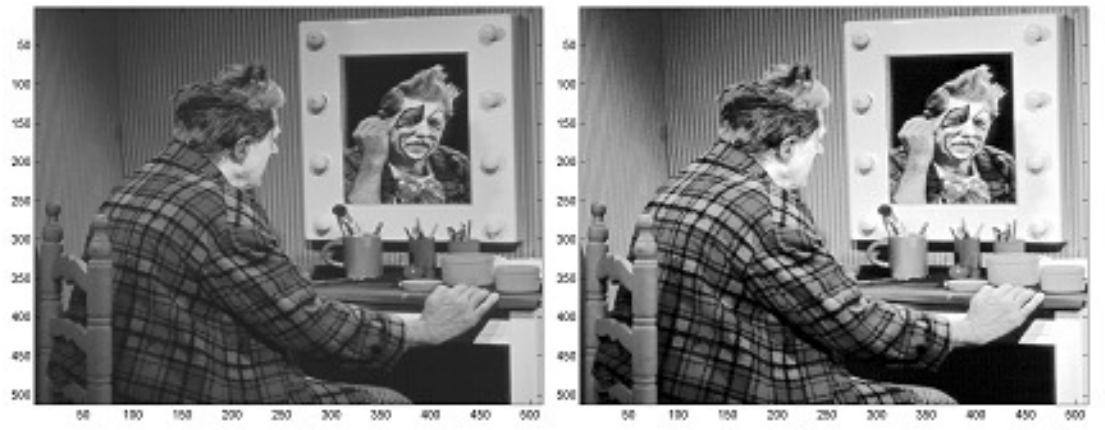


L'histogramme est étiré sur l'ensemble des niveaux de gris de façon presque uniforme : chaque niveau de gris est représenté dans l'image par un nombre de pixels qui tend vers une constante. À nouveau, l'image transformée présente un contraste accentué (étirement d'histogramme).

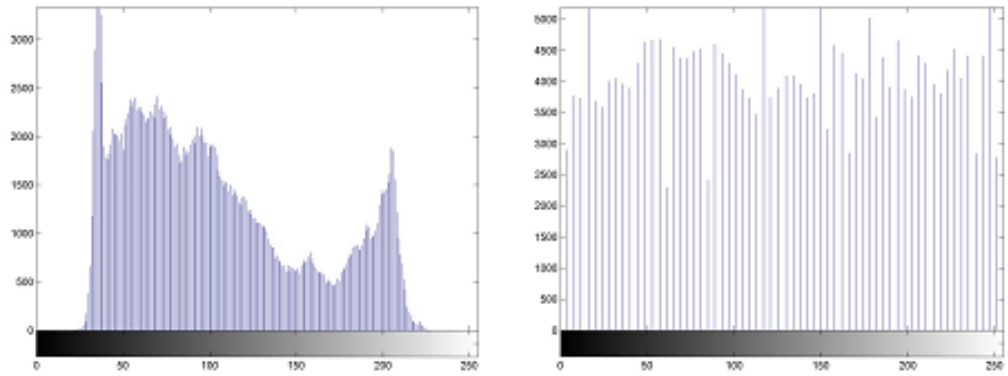
4 – Il s'agit d'effectuer la transformation précédente sur une image quelconque et sans utiliser la démonstration ***imadjdemo***. Après avoir rapatrié l'image *CLOWN_LUMI* dans votre répertoire de travail, entrez les commandes suivantes pour effectuer une égalisation d'histogramme et afficher les résultats:

```
% LUT pour afficher en niveaux de gris
r=0:1/255:1; v=r; b=r;
% Egalisation de l'image
I=imread('CLOWN_LUMI.BMP') ;
image(I)
colormap([r' v' b'])
figure
imhist(I);
J=histeq(I);
figure
image(J)
colormap([r' v' b'])
figure
imhist(J)
```

Voici les images obtenues avant et après égalisation :



Le contraste des niveaux de gris est effectivement nettement plus accentué sur l’image de droite obtenue après égalisation.
Voici les histogrammes obtenus avant et après égalisation :



L’histogramme de droite obtenu après égalisation est presque uniforme et s’étend sur l’ensemble des niveaux de gris.

Exercice Chapitre 2 – Binarisation

Cet exercice a pour objectif de vous présenter différentes réalisations de la binarisation d'une image sous Matlab. Chargez l'image *ISABE_LUMI.BMP* dans votre répertoire de travail et mettez à jour la liste des chemins dans le path browser.

1 – Lisez et affichez l'image (fonctions ***imread*** et ***image***). Affichez l'histogramme de l'image (fonction ***imhist***) afin de choisir un seuil de binarisation.

2 – La fonction ***im2bw*** (*image processing toolbox*) permet la binarisation d'une image. Consultez l'aide, créez et affichez l'image binaire de *ISABE_LUMI* à l'aide de la fonction ***im2bw***.

3 – On se propose maintenant de créer l'image binaire de *ISABE_LUMI* sans utiliser la fonction ***im2bw***. Créez un script de seuillage et de binarisation. Vous pouvez utiliser des instructions classiques (if, for, …), mais comme Matlab est un langage interprété, il est préférable de travailler en vectoriel (la commande ***find*** peut être utile ici).

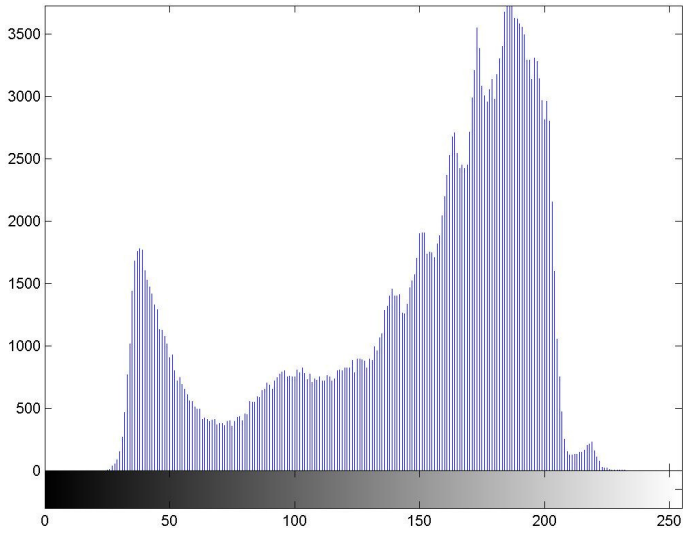
Correction de l'exercice : Binarisation

La binarisation est basée sur un seuillage brut. Cela signifie que si un pixel de l'image a une intensité supérieure à une certaine valeur de seuil, il lui sera attribué la couleur blanche sinon il sera noir. Ce procédé est réalisé sur chaque pixel de l'image. Nous obtenons donc une image comportant seulement deux niveaux (valeur 0 ou 1). L'image est binarisée.

1 – Après avoir rapatrié l'image ISABE_LUMI dans votre répertoire, entrez le script :

```
Im=imread('ISABE_LUMI.BMP');
r=0:1/255:1;
v=r;
b=r;
image(Im);
colormap([r' v' b']);
figure
imhist(Im);
```

On obtient alors l'histogramme suivant:

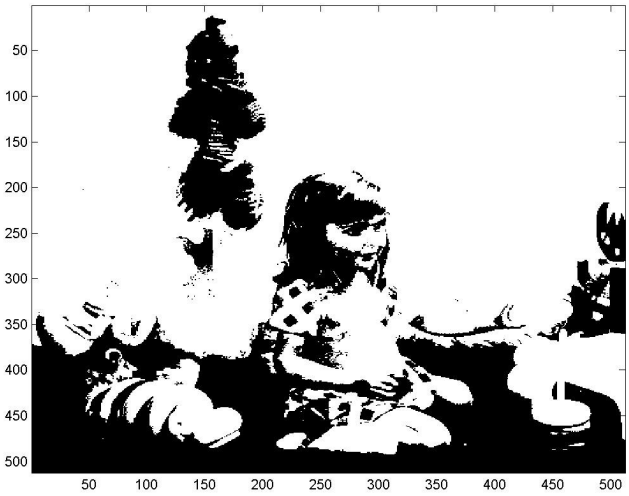


Le seuil de binarisation peut alors être choisi de différentes façons. Dans le cas de l'image *ISABE_LUMI*, on remarque que l'histogramme présente deux pics de population de pixels autour des niveaux de gris 75 et 190. On peut par exemple choisir la valeur de niveau de gris médiane entre ces deux pics. Néanmoins, il existe de nombreuses autres manières d'effectuer le seuillage : valeur moyenne de l'intensité sur l'ensemble de l'histogramme, valeur médiane de l'intervalle des niveau de gris [0, 255]), …

2 – On prend pour seuil de binarisation la valeur médiane 128 de l'intervalle des niveaux de gris [0, 255]. Cette valeur doit être normalisée sur l'intervalle [0, 1] pour être utilisée avec la fonction ***im2bw***. L'image binaire de *ISABE_LUMI* est alors obtenue par les commandes :

```
Im=imread('ISABE_LUMI.BMP');
ImBinaire=im2bw(Im,128/255) ;
imshow(ImBinaire);
```

On obtient alors l'image binaire suivante:



3 – On propose 3 autres scripts pour réaliser la binarisation (seuil=128).

a) Le premier script s'appuie sur l'utilisation de boucles « for ». Le but est de balayer l'ensemble des pixels de l'image. Un par un, on vérifie s'ils sont inférieurs ou supérieurs au seuil établi :

- si la valeur du pixel (i, j) de l'image avant binarisation est inférieure au seuil, le pixel (i, j) de l'image binaire sera noir (valeur 0) ;
- si la valeur est supérieure au seuil, le pixel (i, j) de l'image binaire sera blanc (valeur 1).

Voici le script Matlab pour réaliser ceci :

```
seuil = 128;
Im = imread ('ISABE_LUMI.BMP');
n= size(Im,1);    % Nombre de lignes de l'image
m= size(Im,2);    % Nombre de colonnes de l'image
      for i=1:n         % Balayage sur les lignes de l'image
            for j=1:m   % Balayage sur les colonnes de l'image
                        if Im(i,j) < seuil
                              ImBinaire(i,j) = 0;
                        else
                              ImBinaire (i,j) = 1;
                        end
            end
      end
imshow(ImBinaire);
```

b) Le second script utilise la fonction ***find*** de Matlab (évite les boucles imbriquées et permet un gain de temps de calcul). Cette fonction va chercher les coordonnées des pixels qui vérifient la condition passée en paramètre de la fonction ***find***.

```
Im=imread('ISABE_LUMI.BMP');
seuil = 128 ;
ImBinaire=Im;
ImBinaire(find(ImBinaire<seuil))=0;
ImBinaire(find(ImBinaire>=seuil))=1;
image(ImBinaire)
% Création d'une LUT pour afficher en noir et blanc
r=[0 1];
v=r;
b=r;
map=[r' v' b'];
colormap(map);
```

c) Le troisième script illustre également l'efficacité du calcul matriciel de Matlab (gain de temps de calcul par rapport aux boucles for) :

```
Im = imread ('ISABE_LUMI.BMP');
seuil = 128 ;
ImBinaire = Im > seuil ;
imshow(ImBinaire);
```

En entrant la commande « *ImBinaire* = *Im* > *seuil* », Matlab crée un masque logique *ImBinaire* de même taille que *Im*. Un pixel (i, j) du masque est à « 1 » quand la condition « Im(i, j)> *seuil* » est vraie et à « 0 » sinon. Le masque obtenu correspond donc à l'image binaire souhaitée.

Chapitre 2 – fondamentaux du traitement d'images : transformation ponctuelle

TEST

1 – Soit une **image d'entrée monochrome Ie**, de taille rectangulaire : **M** lignes et **N** pixels par ligne. Le signal associé (niveau de gris) est appelé $\mathbf{s_e(m, n)}$.
On effectue les trois traitements d'image suivants produisant :

- pour le premier, une image de sortie monochrome I_1, de taille (M × N). Le signal associé est appelé $s_1(m, n)$ et donné par : $s_1(m, n) = \sum_{k=m-2}^{m+2} \sum_{l=n-3}^{n+3} s_e(k, l)$.
- pour le second, une image de sortie monochrome I_2, de taille (M × N). Le signal associé est appelé $s_2(m, n)$ et donné par : $s_2(m, n) = 128 + [\ 255 - s_e(m, n)\ ] / 2$.
- pour le troisième, un tableau de sortie T_3, de taille (M × N). L'élément t(m, n) est donné par : $t(m, n) = \sum_{k=1}^{M} \sum_{l=1}^{N} a(k, l).s_e(k, l)$.

Indiquez pour chacun des trois traitements, le type de transformation : **globale**, **locale**, **ponctuelle**. Justifiez en une phrase chacune de vos réponse.

2 – Soit une image numérique monochrome représentée sur 8 bits (niveaux de gris allant de 0 à 255). Construire, sous la forme d'une fonction, le contenu d'une ***Look-Up Table*** (LUT) telle que :

a) elle permet de générer un effet d' « **inverse vidéo** » des niveaux de gris sur l'intervalle [a, b], avec a = 88 et b = 148.

b) elle permet de **visualiser en noir** les pixels de niveaux de gris sur l'intervalle [a, b] et **d'inverser** (« inverse vidéo ») ceux qui sont à l'extérieur de [a, b] (a = 88 et b = 148).

c) elle permet de **linéariser l'affichage** sur un moniteur TV dont la fonction de transfert qui caractérise le passage niveaux de gris (ndg) vers luminance (L) est de type :

$L/L_{MAX} = (ndg/255)^2$, avec $L_{MAX} = 70$.

3 – Soit une image monochrome, représentant une scène composée de **3 objets** :

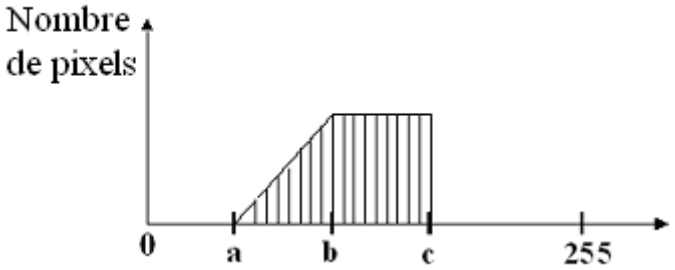
- le fond : niveaux de gris sur [0, a],
- un objet 1 : niveaux de gris sur [a, b],
- un objet 2 : niveaux de gris sur [b, 255].

On prendra a = 64 et b = 192.
En utilisant un ensemble de trois LUTs, associée chacune à une couleur fondamentale Rouge, Vert, Bleu (LUT_R, LUT_V, LUT_B), on souhaite que :

- les pixels appartenant au **fond** s'affichent dans la couleur **jaune** ;
- les pixels appartenant à l'**objet 1** s'affichent en **Magenta** ;
- les pixels appartenant à l'**objet 2** s'affichent en **Cyan**.

À chaque fois les niveaux de luminance seront conservés. Déterminez le contenu de chacune des 3 LUTs et représentez graphiquement celui-ci (i.e. trois graphiques $LUT_X=f_X(ndg)$)

4 – L'histogramme d'une image numérique monochrome I_0 est le suivant :



avec : a = 8 ; b = 16 ; c = 24.
On effectue un étirement d'histogramme pour accroître le contraste de l'image. Quel sera le **facteur d'étirement maximal** ?
Représentez le nouvel histogramme obtenu avec ce facteur.