Master SDA

Traitement de donnés multimédia

TP 3 : Amélioration du contraste de l'image en python

Encadré par :

- **Pr. Alioua Nawal**

Réalisée par :

- **BENAGUERRI Safaa**
- **AGUERCHI Saida**

1.Manipulations des transformations basiques :

1.1 Application des transformations T1 et T2

Les transformations définies sont les suivantes :

- **T1**: $g(i,j)=T1[f(i,j)]=255-f(i,j)$
- **T2**: $g(i,j)=T2[f(i,j)]=0$ si $f(i,j)<128$ et $g(i,j)=255$ si $f(i,j)\geq 128$

Ces transformations modifient l'intensité des pixels des images de la manière suivante :

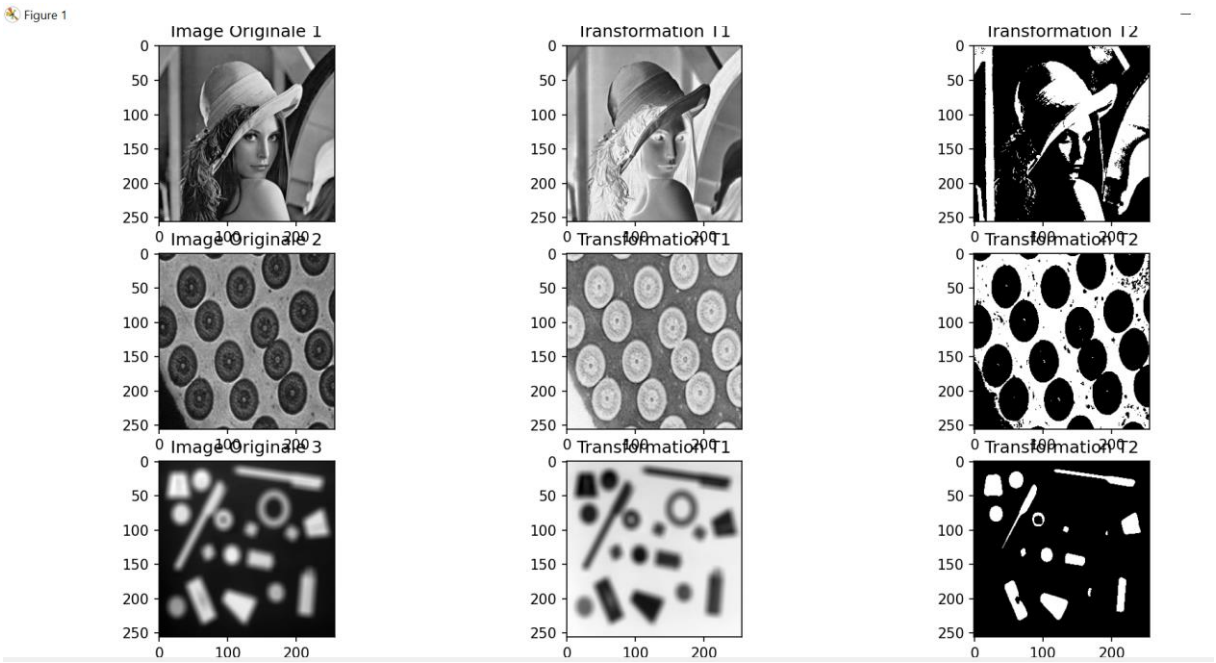
- **T1** est une inversion des niveaux de gris de l'image. Chaque pixel est transformé en $255-f(i,j)$, ce qui crée un effet de "négatif".
- **T2** est une transformation de seuil où tous les pixels ayant une valeur inférieure à 128 deviennent noirs (0), et tous ceux ayant une valeur supérieure ou égale à 128 deviennent blancs (255), ce qui transforme l'image en une sorte de binaire.

Le code python pour appliquer ces deux techniques :

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image1 = cv2.imread('lenam.bmp', cv2.IMREAD_GRAYSCALE)
image2 = cv2.imread('hrec.BMP', cv2.IMREAD_GRAYSCALE)
image3 = cv2.imread('objectm.bmp', cv2.IMREAD_GRAYSCALE)

# Transformation T1
T1_image1 = 255 - image1
T1_image2 = 255 - image2
T1_image3 = 255 - image3

# Transformation T2
T2_image1 = np.where(image1 < 128, 0, 255)
T2_image2 = np.where(image2 < 128, 0, 255)
T2_image3 = np.where(image3 < 128, 0, 255)

# résultats
plt.figure(figsize=(18, 12))

# Image 1
plt.subplot(3, 3, 1), plt.imshow(image1, cmap='gray'), plt.title('Image
Originale 1')
plt.subplot(3, 3, 2), plt.imshow(T1_image1, cmap='gray'),
plt.title('Transformation T1')
```

```
plt.subplot(3, 3, 3), plt.imshow(T2_image1, cmap='gray'),
plt.title('Transformation T2')

# Image 2
plt.subplot(3, 3, 4), plt.imshow(image2, cmap='gray'), plt.title('Image
Originale 2')
plt.subplot(3, 3, 5), plt.imshow(T1_image2, cmap='gray'),
plt.title('Transformation T1')
plt.subplot(3, 3, 6), plt.imshow(T2_image2, cmap='gray'),
plt.title('Transformation T2')

# Image 3
plt.subplot(3, 3, 7), plt.imshow(image3, cmap='gray'), plt.title('Image
Originale 3')
plt.subplot(3, 3, 8), plt.imshow(T1_image3, cmap='gray'),
plt.title('Transformation T1')
plt.subplot(3, 3, 9), plt.imshow(T2_image3, cmap='gray'),
plt.title('Transformation T2')

plt.tight_layout()
plt.show()
```

Résultat :



Remarque :

- **T1** crée un effet de négatif en inversant les couleurs.
- **T2** crée un effet binaire, où l'image devient une silhouette de haute ou basse intensité.

2.Manipulation d'histogrammes d'images

2.1 Affichage de l'histogramme d'une image simple

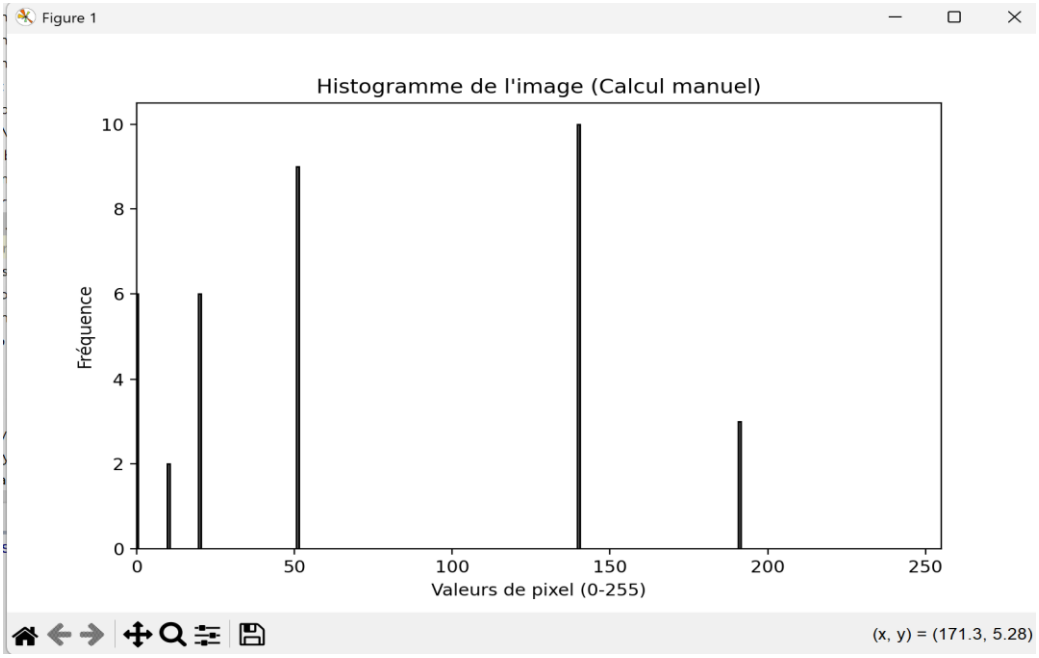
1. Calcul et affichage de l'histogramme d'une image simple :

Le calcul de l'histogramme d'une image consiste à compter combien de pixels ont une certaine intensité.

Code python:

```
import numpy as np
import matplotlib.pyplot as plt

image = np.array([
    [0, 140, 51, 191, 140, 51],
    [0, 51, 191, 140, 140, 51],
    [51, 140, 20, 20, 140, 0],
    [51, 140, 20, 20, 20, 140],
    [0, 140, 191, 0, 20, 51],
    [0, 10, 51, 10, 140, 51]
], dtype=np.uint8)


def compute_histogram(image):
    hist = np.zeros(256, dtype=int)
    for row in image:
        for pixel in row:
            hist[pixel] += 1
    return hist

# Caclcule
histogram = compute_histogram(image)

# Affichage
plt.figure(figsize=(8, 5))
plt.bar(range(256), histogram, color='gray', edgecolor='black')
plt.title("Histogramme de l'image (Calcul manuel)")
plt.xlabel("Valeurs de pixel (0-255)")
plt.ylabel("Fréquence")
plt.xlim([0, 255])
plt.show()
```

Résultat :



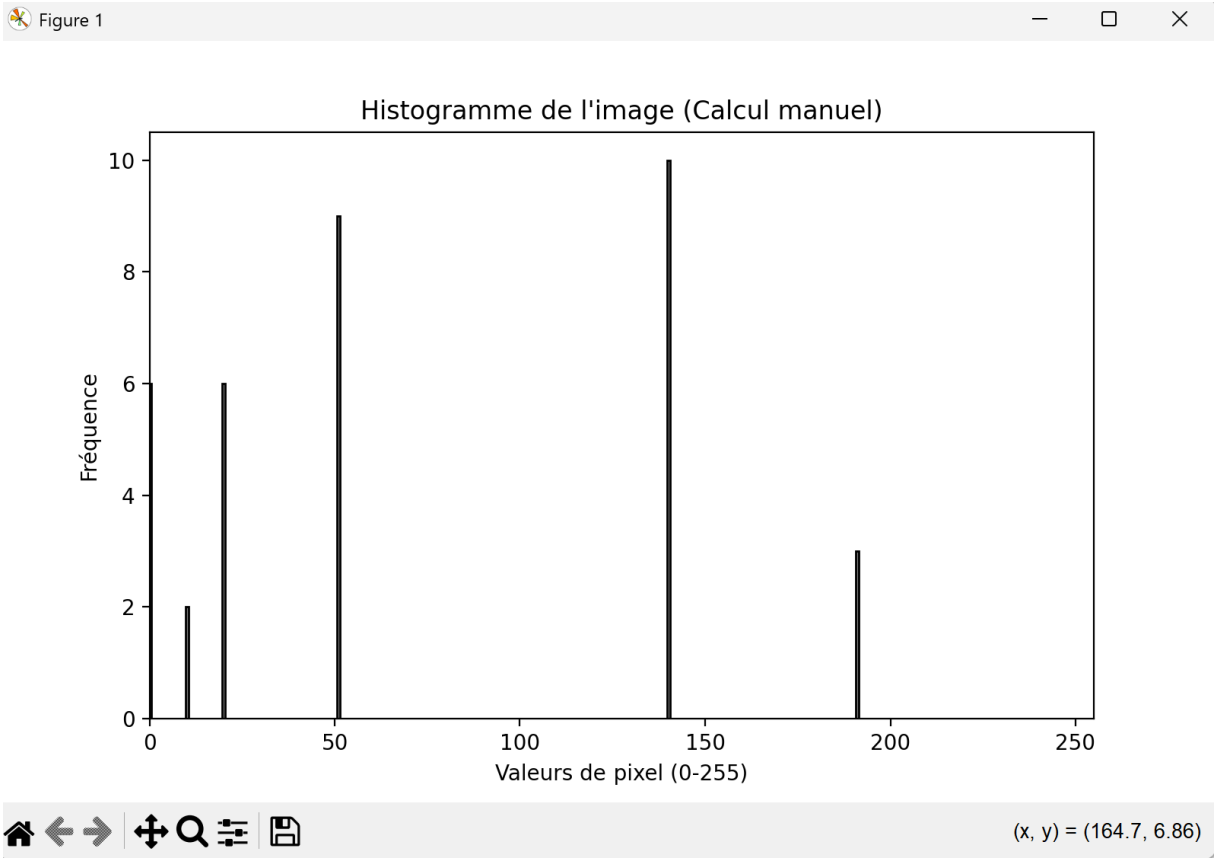
2. Utilisation d'une la fonction du calcul d'histogramme prédéfini

Pour cette partie, nous allons utiliser OpenCV qui fournit une méthode efficace pour calculer l'histogramme.

Code python :

```
import cv2

image = np.array([
    [0, 140, 51, 191, 140, 51],
    [0, 51, 191, 140, 140, 51],
    [51, 140, 20, 20, 140, 0],
    [51, 140, 20, 20, 20, 140],
    [0, 140, 191, 0, 20, 51],
    [0, 10, 51, 10, 140, 51]
], dtype=np.uint8)

# Calcul de l'histogramme avec OpenCV
hist_cv2 = cv2.calcHist([image], [0], None, [256], [0, 256])

# Affichage de l'histogramme
plt.figure(figsize=(8, 5))
plt.plot(hist_cv2, color='black')  # Trace un graphique au lieu d'un
histogramme en barres
plt.title("Histogramme de l'image (OpenCV)")
plt.xlabel("Valeurs de pixel (0-255)")
plt.ylabel("Fréquence")
plt.xlim([0, 255])
plt.grid()
plt.show()
```

Résultat :



Comparaison des deux méthodes :

- **La première approche** : permet de comprendre comment fonctionne un histogramme en comptant chaque pixel.
- **La seconde approche (OpenCV)** : est plus optimisée et rapide, particulièrement utile pour de grandes images.

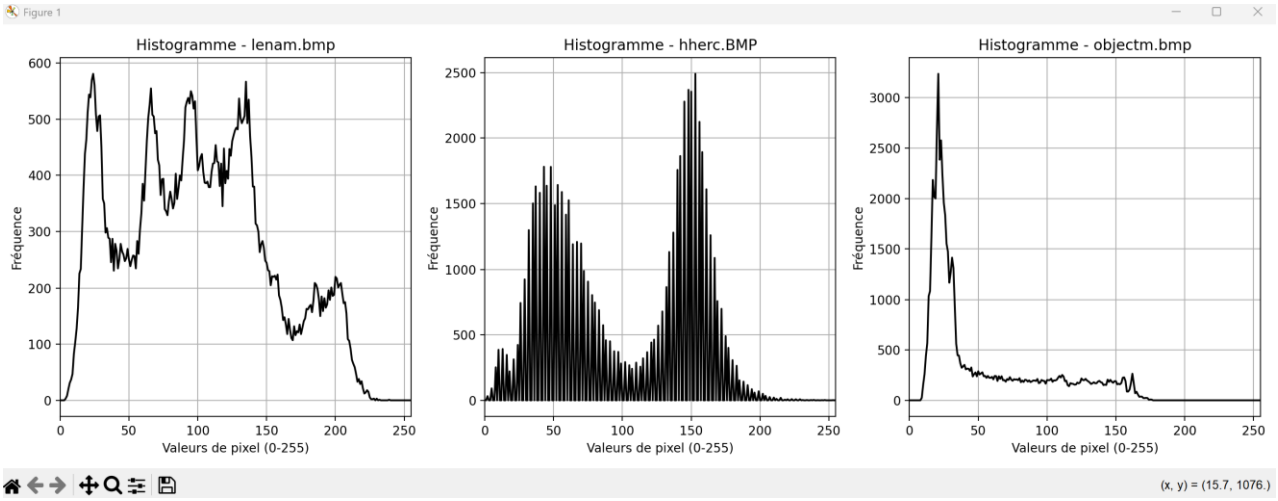
2.2 Affichage de l'histogramme des images de travail :

Code python :

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


image1 = cv2.imread('lenam.bmp', cv2.IMREAD_GRAYSCALE)

image2 = cv2.imread('hrec.BMP', cv2.IMREAD_GRAYSCALE)
image3 = cv2.imread('objectm.bmp', cv2.IMREAD_GRAYSCALE)

if image1 is None or image2 is None or image3 is None:
    print("Erreur : Impossible de charger une ou plusieurs images.")
```

```
else:

    def plot_histogram(image, title):
        hist = cv2.calcHist([image], [0], None, [256], [0, 256])
        plt.plot(hist, color='black')
        plt.title(title)
        plt.xlabel("Valeurs de pixel (0-255)")
        plt.ylabel("Fréquence")
        plt.xlim([0, 255])
        plt.grid()

    plt.figure(figsize=(15, 5))

    # Image 1
    plt.subplot(1, 3, 1)
    plot_histogram(image1, "Histogramme - lenam.bmp")

    # Image 2
    plt.subplot(1, 3, 2)
    plot_histogram(image2, "Histogramme - hherc.BMP")

    # Image 3
    plt.subplot(1, 3, 3)
    plot_histogram(image3, "Histogramme - objectm.bmp")

    plt.tight_layout()
    plt.show()
```

Résultat :



Comparaison des trois histogrammes :

Image	Répartition des pixels	Contraste	Observations
lenam.bmp	Large (0-255)	Élevé	Bonne répartition des tons
hherc.BMP	Moyenne (50-150)	Moyen	Image plus uniforme, manque de contrastes forts

objectm.bmp	Concentrée dans les basses valeurs (0-50)	Faible	Image très sombre

2.3 Transformation de l'histogramme

Étapes du programme :

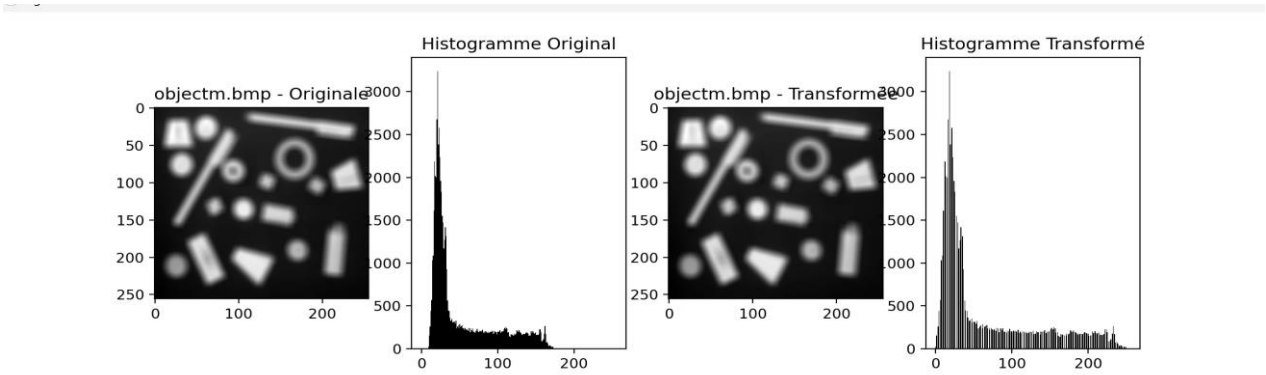
1. Charger l'image en niveaux de gris.
2. Appliquer la transformation pour étendre les valeurs de pixels entre 0 et 255.
3. Tracer l'image et son histogramme avant/après transformation.

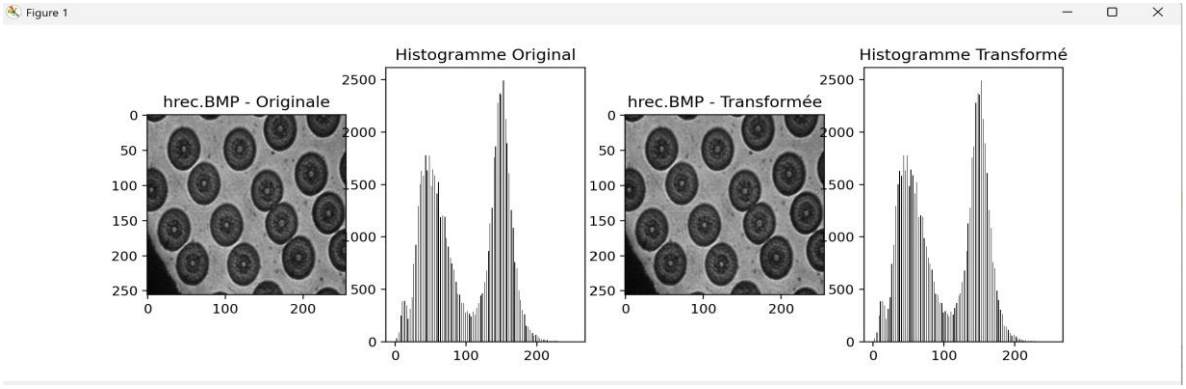
Code python :

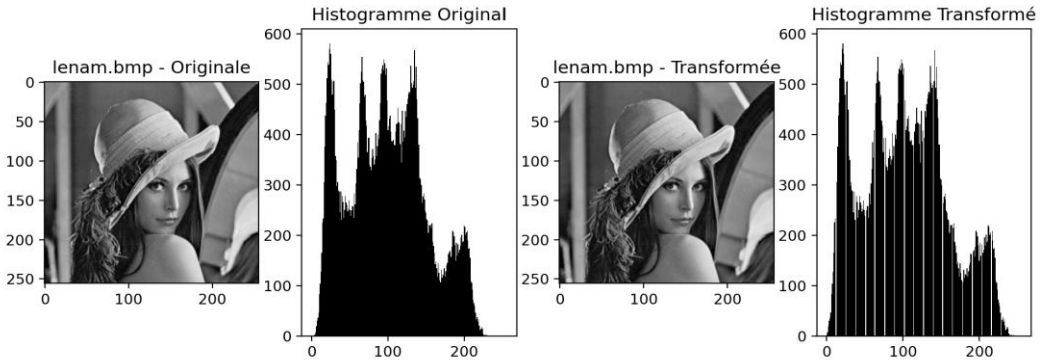
```
import cv2
import numpy as np
import matplotlib.pyplot as plt


def normalize_image(image):
    min_val = np.min(image)
    max_val = np.max(image)
    normalized = (image - min_val) * (255 / (max_val - min_val))
    return normalized.astype(np.uint8)

image1 = cv2.imread('lenam.bmp', cv2.IMREAD_GRAYSCALE)
image2 = cv2.imread('hrec.BMP', cv2.IMREAD_GRAYSCALE)
image3 = cv2.imread('objectm.bmp', cv2.IMREAD_GRAYSCALE)

if image1 is None or image2 is None or image3 is None:
    print("Erreur : Impossible de charger une ou plusieurs images.")
    exit()

norm_image1 = normalize_image(image1)
norm_image2 = normalize_image(image2)
norm_image3 = normalize_image(image3)

def show_image_histogram(original, transformed, title):
    plt.figure(figsize=(12, 4))

    # Image originale
    plt.subplot(1, 4, 1)
    plt.imshow(original, cmap='gray')
    plt.title(f"{title} - Originale")

    # Histogramme de l'image originale
    plt.subplot(1, 4, 2)
    plt.hist(original.ravel(), bins=256, range=(0, 255), color='black')
    plt.title("Histogramme Original")

    # Image transformée
    plt.subplot(1, 4, 3)
    plt.imshow(transformed, cmap='gray')
```

```
    plt.title(f"{title} - Transformée")

    # Histogramme de l'image transformée
    plt.subplot(1, 4, 4)
    plt.hist(transformed.ravel(), bins=256, range=(0, 255), color='black')
    plt.title("Histogramme Transformé")


# Affichage des images et histogrammes
show_image_histogram(image1, norm_image1, "lenam.bmp")
show_image_histogram(image2, norm_image2, "hrec.BMP")
show_image_histogram(image3, norm_image3, "objectm.bmp")

plt.show()
```

Résultat :







Remarques :

D'après les résultats obtenus sur les trois images (lenam.bmp, objectm.bmp, hherc.BMP), voici les observations :

1. **Image lenam.bmp :**
    - L'histogramme original était bien réparti, mais avec quelques pics dominants.
    - Après transformation, l'histogramme est plus uniformisé et étendu, augmentant ainsi le contraste.
    - Visuellement, l'image transformée semble plus détaillée, avec des différences de luminosité mieux accentuées.
2. **Image objectm.bmp :**
    - L'histogramme original est fortement concentré vers les basses valeurs, indiquant une image sombre avec des objets lumineux.
    - Après transformation, la répartition s'étale un peu plus, mais les pixels sombres restent dominants.
    - L'effet visuel ne change pas drastiquement car l'image contient principalement du noir et des formes blanches bien définies.
3. **Image hherc.BMP :**
    - L'histogramme original montre une distribution bimodale, indiquant deux grandes plages de niveaux de gris.
    - Après transformation, la forme globale reste similaire, mais les contrastes sont légèrement accentués.
    - L'amélioration est subtile car l'image d'origine avait déjà une bonne répartition des niveaux de gris.

2.4 Égalisation de l'histogramme :

Méthodes à appliquer :

1. **Égalisation avec cv2.equalizeHist()**

- Cette méthode globale redistribue les niveaux de gris de l'image pour obtenir un histogramme plus uniforme.
- Elle améliore le contraste en étalant les valeurs de l'histogramme sur toute la plage de 0 à 255.
- Très efficace sur des images où le contraste est faible.

2. Égalisation adaptative avec exposure.equalize_adapthist() (CLAHE)

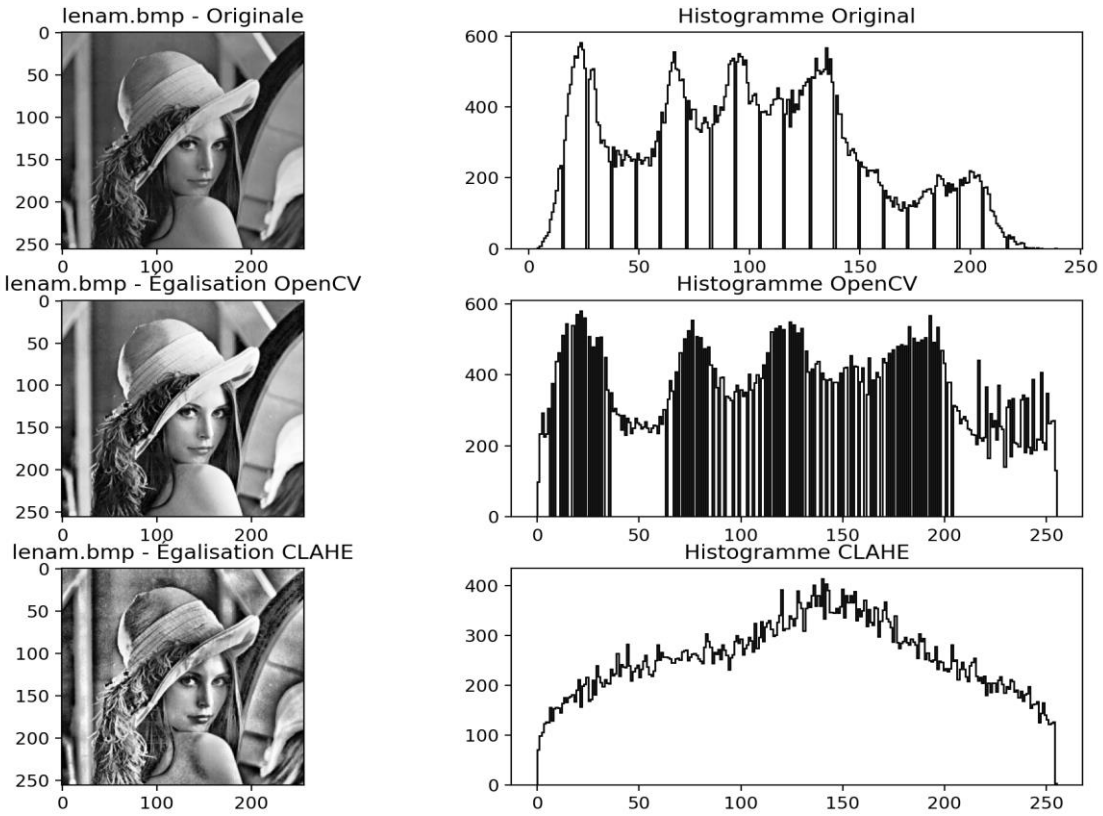
- Contrairement à cv2.equalizeHist(), cette méthode applique l'égalisation localement sur de petites régions appelées "tiles" (sous-blocs de l'image).
- Un paramètre **clip_limit** permet de limiter l'intensité de l'égalisation, évitant ainsi une suramplification du bruit.

Code python:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import exposure


image_filenames = ["lenam.bmp", "objectm.bmp", "hherc.BMP"]


for img_name in image_filenames:

    img = cv2.imread(img_name, cv2.IMREAD_GRAYSCALE)

    if img is None:
        print(f"Erreur : Impossible de charger {img_name}")
        continue

    # Égalisation globale avec OpenCV
    img_eq_cv2 = cv2.equalizeHist(img)

    # Égalisation adaptative avec skimage
    img_eq_clahe = exposure.equalize_adapthist(img, clip_limit=0.03)
    img_eq_clahe = (img_eq_clahe * 255).astype(np.uint8)

    # Tracer les résultats
    fig, axes = plt.subplots(3, 3, figsize=(12, 10))

    # Image originale + Histogramme
    axes[0, 0].imshow(img, cmap='gray')
    axes[0, 0].set_title(f"{img_name} - Originale")
    axes[0, 1].hist(img.ravel(), bins=256, histtype='step', color='black')
    axes[0, 1].set_title("Histogramme Original")

    # Égalisation OpenCV + Histogramme
    axes[1, 0].imshow(img_eq_cv2, cmap='gray')
    axes[1, 0].set_title(f"{img_name} - Égalisation OpenCV")
    axes[1, 1].hist(img_eq_cv2.ravel(), bins=256, histtype='step',
color='black')
    axes[1, 1].set_title("Histogramme OpenCV")
```

```
    # Égalisation Adaptative + Histogramme
    axes[2, 0].imshow(img_eq_clahe, cmap='gray')
    axes[2, 0].set_title(f"{img_name} - Égalisation CLAHE")
    axes[2, 1].hist(img_eq_clahe.ravel(), bins=256, histtype='step',
color='black')
    axes[2, 1].set_title("Histogramme CLAHE")

    for i in range(3):
        axes[i, 2].axis('off')

    plt.tight_layout()
    plt.show()
```

Résultat :



Analyse des résultats :

Égalisation avec cv2.equalizeHist() :

- L'image transformée est plus contrastée.
- L'histogramme est redistribué de manière plus uniforme.
- Peut créer des artefacts (perte de détails dans les zones très claires ou sombres).

Égalisation adaptative :

- Meilleure préservation des détails fins.
- Améliore localement le contraste sans exagérer les variations extrêmes.
- L'histogramme final est plus lissé et sans pics excessifs.
- Peut ajouter du bruit dans certaines zones (si clip_limit est mal réglé).

Comparaison des méthodes :

Méthode	Avantages	Inconvénients
cv2.equalizeHist()	Simplicité, améliore globalement le contraste	Peut exagérer les contrastes et perdre des détails
exposure.equalize_adapthist() (CLAHE)	Meilleur contrôle du contraste local, préserve les détails	Plus complexe, peut générer du bruit si mal configuré