KENNESAW STATE
UNIVERSITY

CS 4732
MACHINE VISION

ASSIGNMENT 2

IMAGE ENHANCEMENT

INSTRUCTOR

Dr. Sanghoon Lee

Your Name: Max Haviv
KSU ID: 001029496

1. ABSTRACT

In this assignment I learn multiple methods of enhancement to improve gray scale images contrast using multiple methods. These methods are log transformation, power law transformation and histogram equalization. In this assignment I use python, and matplotlib to visualize, read and visualize images. I also use numpy to create empty image arrays.

2. TEST RESULTS

2.1 Test Results for power law transformation

The fourierspectrum.pgm image was provided and used to display the differences that the two enhancement algorithms work. Figure 1 shows the transformations of the image using power law transformation, and figure 2 shows the image after using log transformation.

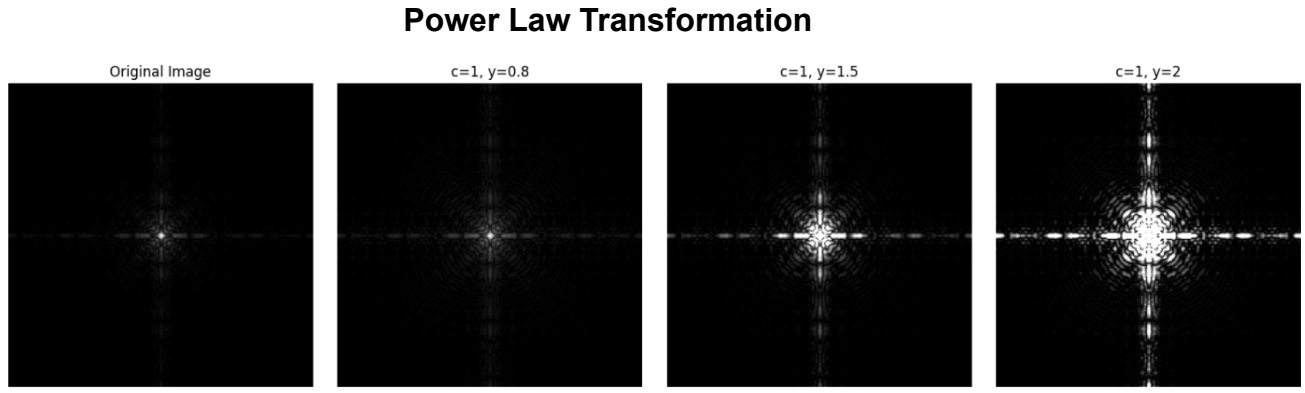


Fig (1) Original Image and three images detailing enhancement using power law transformation

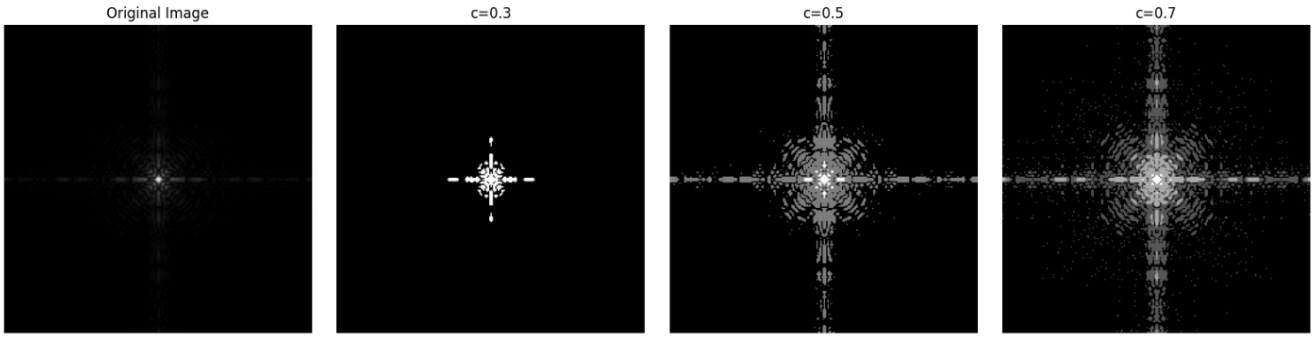


Fig (2) Original Image and three images detailing enhancement using log transformation

The difference between the two transformation methods is small but significant. First off, power law transformation allows for much more control over how quickly and aggressively the

contrast is adjusted while log transformation has significantly less control. Another difference is that with log transformation it starts with more intense gray levels while slowly decreasing to an equilibrium. Power law on the other hand starts with lower intensity levels and gets more and more intense as you increase the value y. Power law transformation also seems to be more precise in adding contrast while still staying true to the original image.

2.2 Test Results for Histogram Equalization

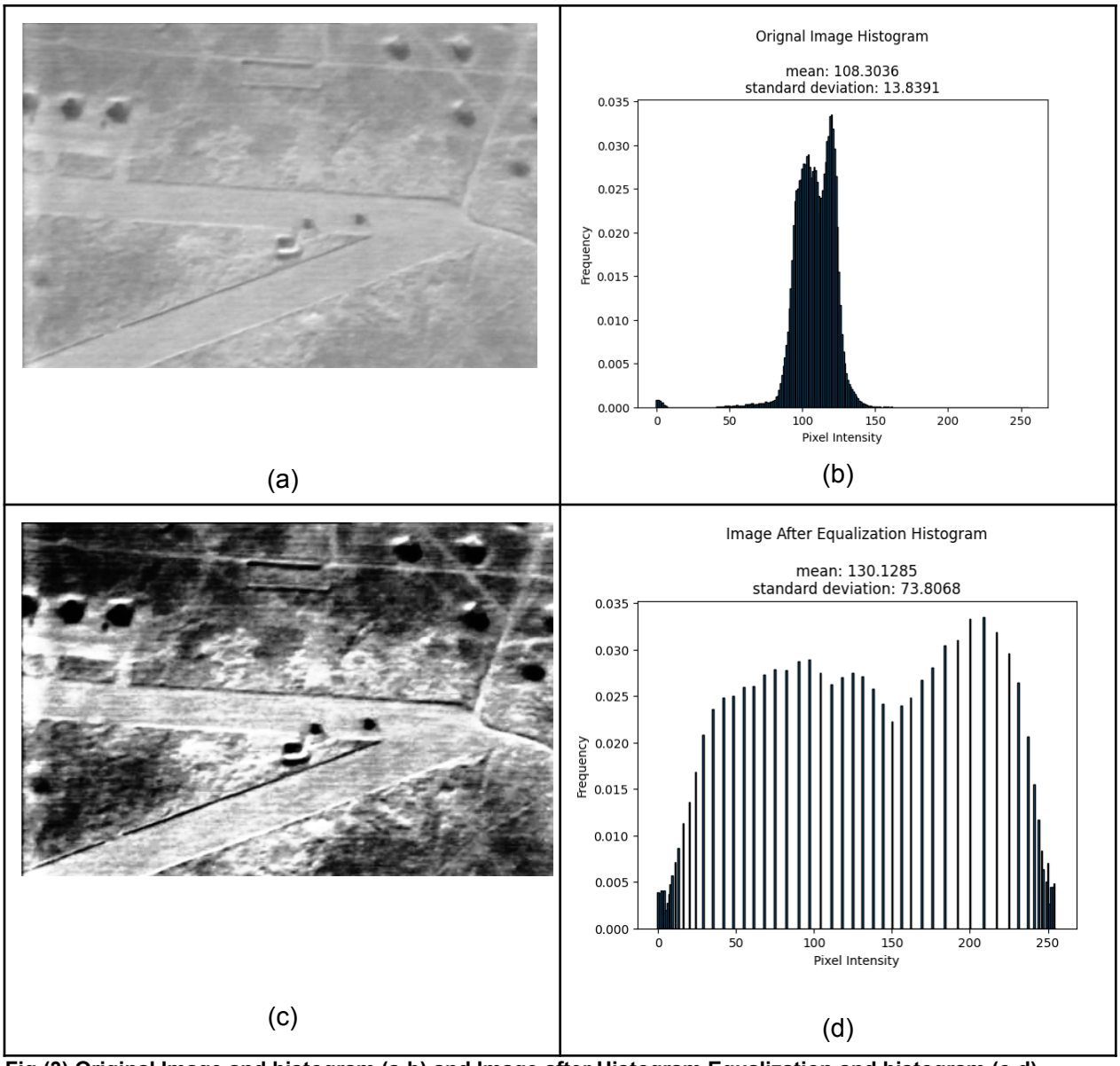


Fig (3) Original Image and histogram (a-b) and Image after Histogram Equalization and histogram (c-d)

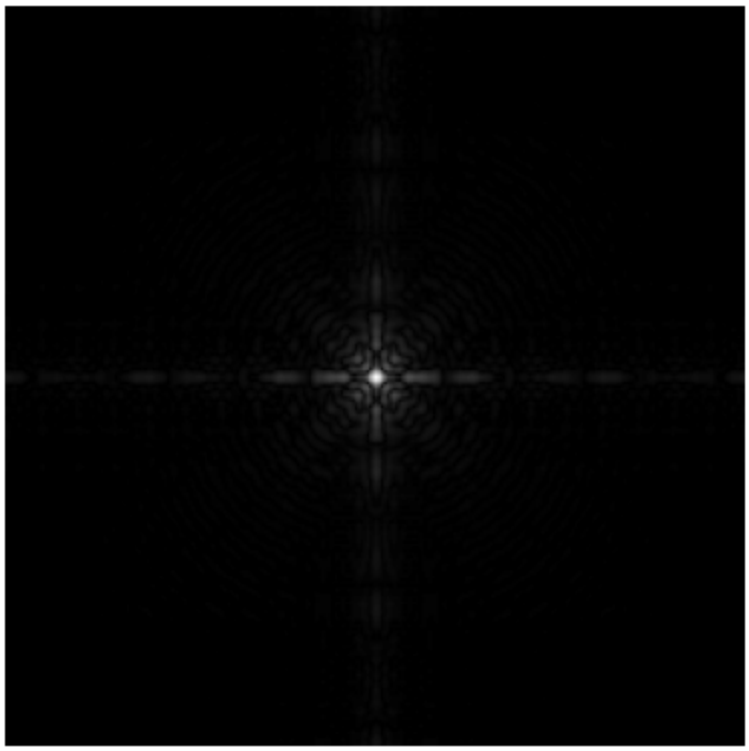
As you can see from the image after transformation, the second image has significantly more contrast than the original image. You can also visualize the contrast difference in the before and after histograms. With the first histogram you can see how all the intensity levels are close together while in the second histogram all the intensity levels are spread out. You can see though that the intensity levels change but the frequency levels do stay the same. This is because the levels are just being spread out. You can also visualize this data with the mean and standard deviation. In the original histogram the mean is around 108 and the standard deviation is around 14. Then after the histogram equalization the mean is around 130 and the standard deviation is around 74. So as you can see the mean has shifted by around 22 which shows that the main intensity levels didn't shift too much. But the standard deviation changed by around 70. That is a major change which shows you how much more spread out all the intensity labels are.

3. Discussion

From this assignment I learned simple image enhancement using power law transformation, log transformation, and histogram equalization. I was familiar with these concepts but had no idea how to implement them. It was very interesting to learn about the transfer images that were used in the histogram equalization as I would have never guessed to implement an algorithm like that. I was aware of individual pixel alterations with power law and log transformations but I was not aware about the different transformation functions, and it was cool to visualize the different transformation functions that can be made with the power law.

4. CODE

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = mpimg.imread('fourierspectrum.pgm')
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.show()
```

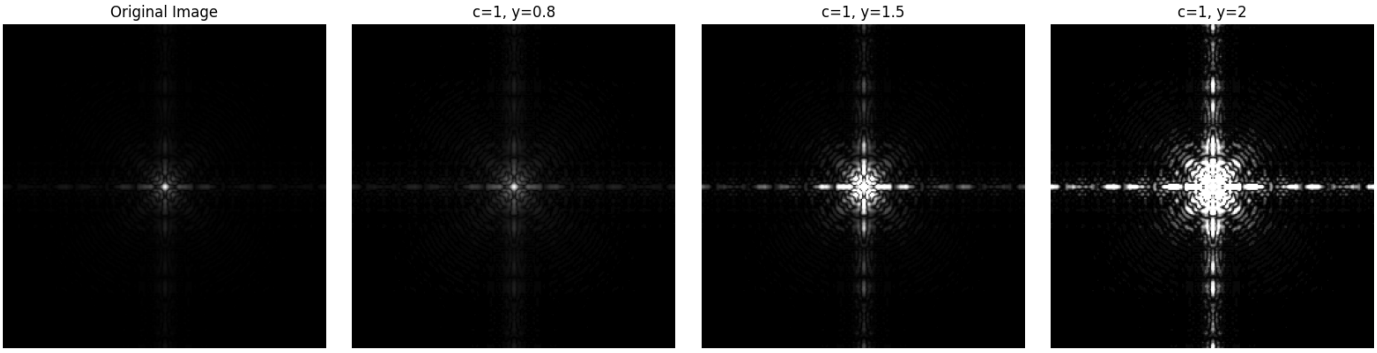


Power Law Transformation

In [12]:
```
def power_law_transformation (c, y, img):
  img_shape = img.shape
  length = img_shape[0]
  height = img_shape[1]

  # create new img to return
  new_img = np.zeros((length, height))

  # loop through every pixel
  for row in range (length):
    for col in range(height):
      new_img[row][col] = c * (img[row][col] ** y)

  # Normalize the image to [0, 255] range and cast to uint8
  new_img = np.clip(new_img, 0, 255)

  return new_img.astype(np.uint8)
```

In [13]:
```
c_values = [1, 1, 1]
y_values = [0.8, 1.5, 2]

plt.figure(figsize=(16, 4))

# Original image
plt.subplot(1, 4, 1)
```

```
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.title('Original Image')

# 3 different variations of power law transformation
for i in range(3):
    c = c_values[i]
    y = y_values[i]
    new_img = power_law_transformation(c, y, img)

    plt.subplot(1, 4, i+2)
    plt.imshow(new_img, cmap='gray')
    plt.axis('off')
    plt.title(f'c={c}, y={y}')

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

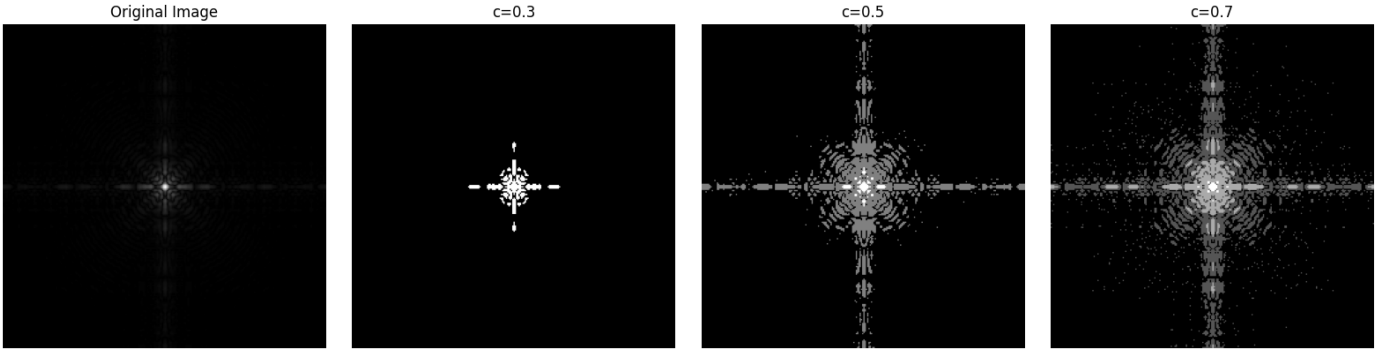


Log Transformation

In [14]:
```
def log_transformation(c, img):
  img_shape = img.shape
  length = img_shape[0]
  height = img_shape[1]

  # create new img to return
  new_img = np.zeros((length, height))

  # loop through every pixel
  for row in range (length):
    for col in range(height):
      new_img[row][col] = c * (np.log(1 + img[row][col]))

  # Normalize the image to [0, 255] range and cast to uint8
  new_img = np.clip(new_img, 0, 255)

  return new_img.astype(np.uint8)
```

In [15]:
```
c_values = [0.3, 0.5, 0.7]

plt.figure(figsize=(16, 4))

# Original image
plt.subplot(1, 4, 1)
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.title('Original Image')

# three diferent variations of log transformation
```

```
for i in range(3):
    c = c_values[i]
    new_img = log_transformation(c, img)

    plt.subplot(1, 4, i+2)
    plt.imshow(new_img, cmap='gray')
    plt.axis('off')
    plt.title(f'c={c}')

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

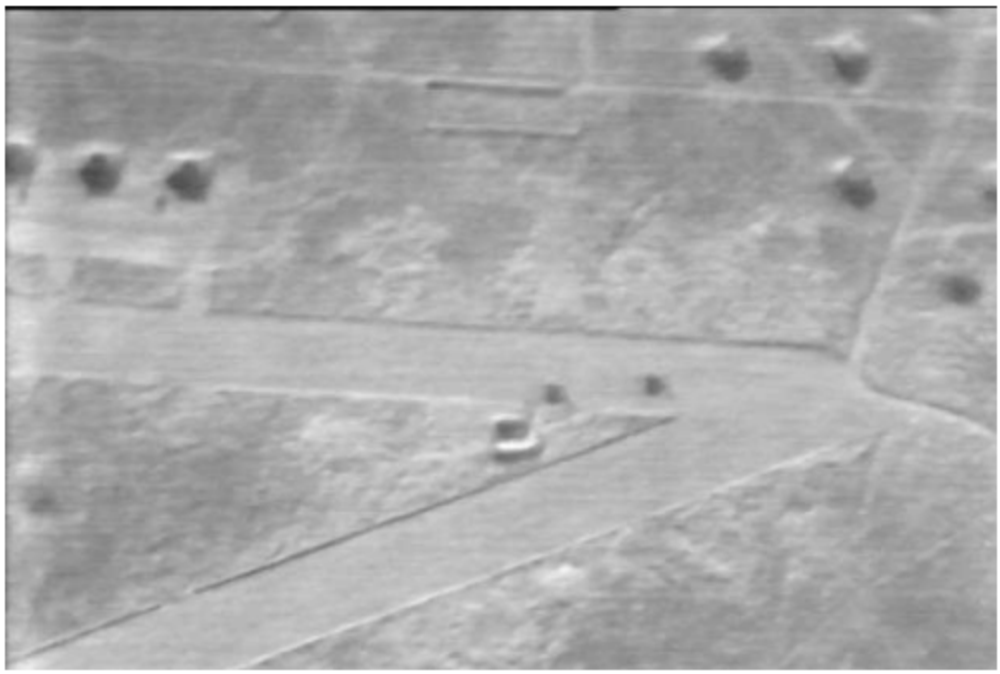


Histogram Equilization

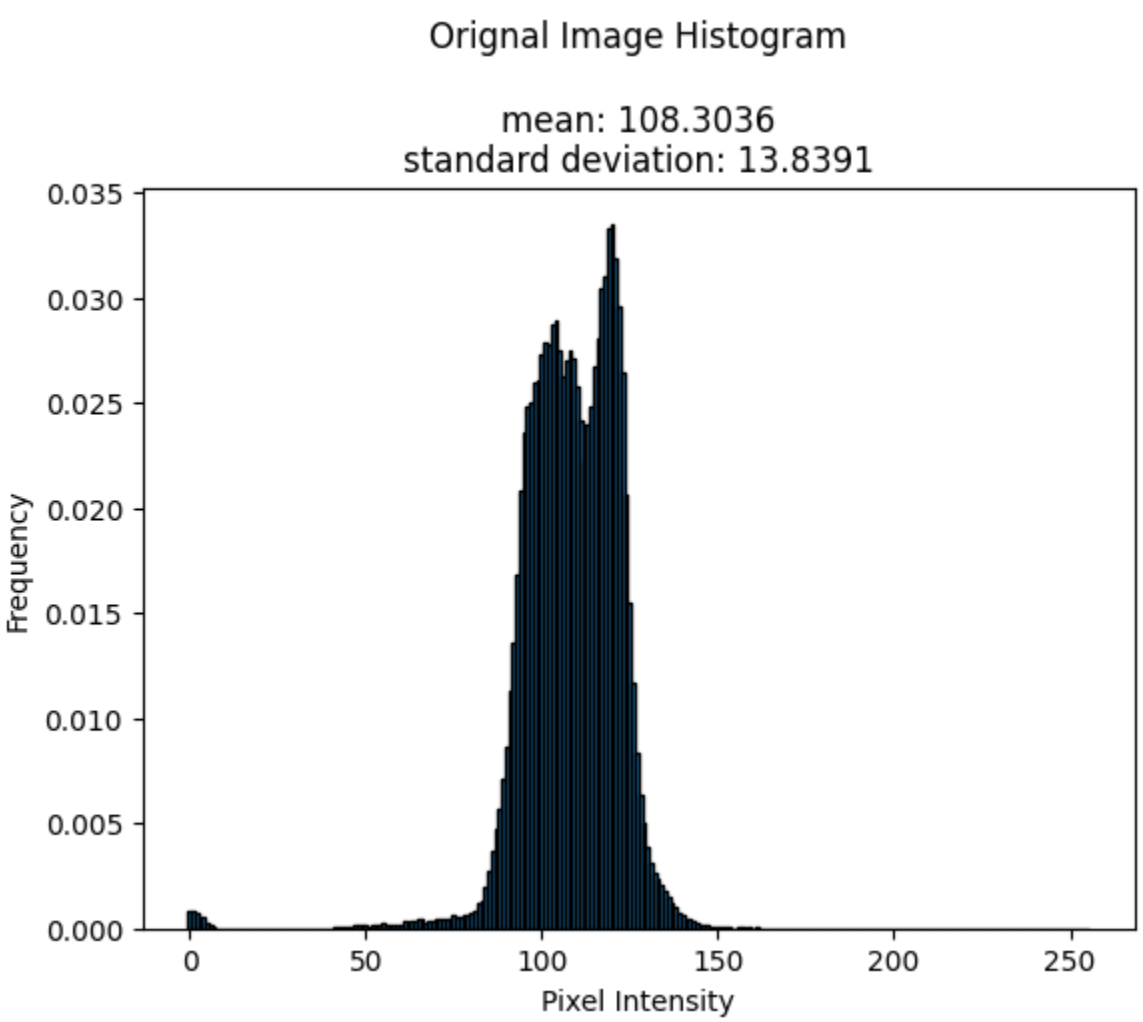
In [16]:
```
# create a normalized histogram
def normalized_hist(img):
  # ensure that all images are unint8 values
  img = np.uint8(img)

  m, n = img.shape
  hist = np.zeros(256)

  # loop through every pixel and count how many pixel intensities exist
  for row in range(m):
    for col in range(n):
      hist[img[row,col]] += 1

  return np.array(hist/(m*n))

# histogram equalization function
def hist_equalization(img):
  hist = normalized_hist(img) # normalize the histogram
  cdf = np.cumsum(hist) # calculate the cumalitive sum
  transfer_img = np.uint8(255 * cdf)
  # create the transfer image be multiplying the cumalitive sum by (L-1)
  # L = 256 since we are working with 8 bit gray level images

  m,n  = img.shape
  new_img = np.zeros((m,n))

  # apply the tranfer image to each pixel in the original image
  for row in range(m):
    for col in range(n):
      new_img[row,col] = transfer_img[img[row,col]]

  return new_img
```

Initial Image

In [17]:
```
img = mpimg.imread('banker.jpeg')
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.show()
```



In [18]:
```
hist = normalized_hist(img)
intensity_levels = np.arange(len(hist))
mean = np.round(np.sum(hist * intensity_levels), 4)
std_dev = np.round(np.sqrt(np.sum(hist * (intensity_levels - mean) ** 2)), 4)

plt.bar(range(256), hist, width=1.0, edgecolor='black')
plt.title(f'Orignal Image Histogram\n\nmean: {mean}\nstandard deviation: {std_dev}')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.show()
```



In [19]:
```
hist_equalization_img = hist_equalization(img)
plt.imshow(hist_equalization_img, cmap='gray')
plt.axis('off')
plt.title('Image After Histogram Equalization')
plt.show()
```

In [20]:
```
hist = normalized_hist(hist_equalization_img)
```

```
intensity_levels = np.arange(len(hist))
mean = np.round(np.sum(hist * intensity_levels), 4)
std_dev = np.round(np.sqrt(np.sum(hist * (intensity_levels - mean) ** 2)), 4)

plt.bar(range(256), hist, width=1.0, edgecolor='black')
plt.title(f'Image After Equalization Histogram\n\nmean: {mean}\nstandard deviation: {std
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.show()
```