P2: Demonstrate the concept of sampling and quantization.
Objective: To explore the effects of sampling and quantization on image quality using students' own photographs. This exercise includes visual comparisons and basic quantitative analysis to understand the trade-offs involved.

Scenario: You are a digital imaging specialist working on optimizing high-resolution images for various applications. Your task is to evaluate how different sampling rates and quantization levels affect image quality using your own high-resolution photographs.

Tasks:

1. Image Preprocessing:
   ○ Use your own high-resolution photograph (minimum 1024x1024 pixels) and convert the picture to grayscale to simplify the analysis.

2. Image Sampling:
   ○ Downsample the grayscale image to various resolutions (e.g., 512x512,256x256, and 128x128 pixels) using nearest-neighbor interpolation.
   ○ Observe and interpret the details of each downsampled image's visual changes and potential loss.

3. Image Quantization:
   ○ Quantize the grayscale image to different levels of intensity (e.g., 256 levels,64 levels, and 4 levels).
   ○ Observe and interpret how the reduction in quantization levels affects theimage quality.

4. Visual and Quantitative Analysis:
   ○ Compare the original, downsampled, and quantized images by plotting histograms of pixel values
   ○ Visualize the effect of quantization levels.

5. Error Metrics:
   ○ Calculate the Mean Squared Error (MSE) between the original image and each downsampled and quantized image.
   ○ Interpret how these error metrics correlate with visual observations.

IMAGE PROCESSING

In [9]:
```
import cv2
import matplotlib.pyplot as plt

image_path = 'myphoto.JPG'
```

```
image = cv2.imread(image_path)

# Convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Display the original grayscale image
plt.imshow(gray_image, cmap='gray')
plt.title("Grayscale Image")
plt.axis('off')
plt.show()
```

Grayscale Image

IMAGE SAMPLING

In [10]:

```
# Downsample image using nearest-neighbor interpolation
resolutions = [512, 256, 128]
downsampled_images = []

for res in resolutions:
    downsampled = cv2.resize(gray_image, (res, res), interpolation=cv2.INTER
    downsampled_images.append(downsampled)

    # Display downsampled images
    plt.imshow(downsampled, cmap='gray')
    plt.title(f"Downsampled to {res}x{res}")
    plt.axis('off')
    plt.show()
```

Downsampled to 512x512

Downsampled to 256x256

Downsampled to 128x128

IMAGE QUANTIZATION

In [11]:
```
import numpy as np

# Function to quantize image
def quantize_image(image, levels):
    quantized = np.floor(image / (256 / levels)) * (256 / levels)
    return quantized.astype(np.uint8)

quantization_levels = [256, 64, 4]
quantized_images = []

for levels in quantization_levels:
    quantized = quantize_image(gray_image, levels)
    quantized_images.append(quantized)

    # Display quantized images
    plt.imshow(quantized, cmap='gray')
    plt.title(f"Quantized to {levels} levels")
    plt.axis('off')
    plt.show()
```

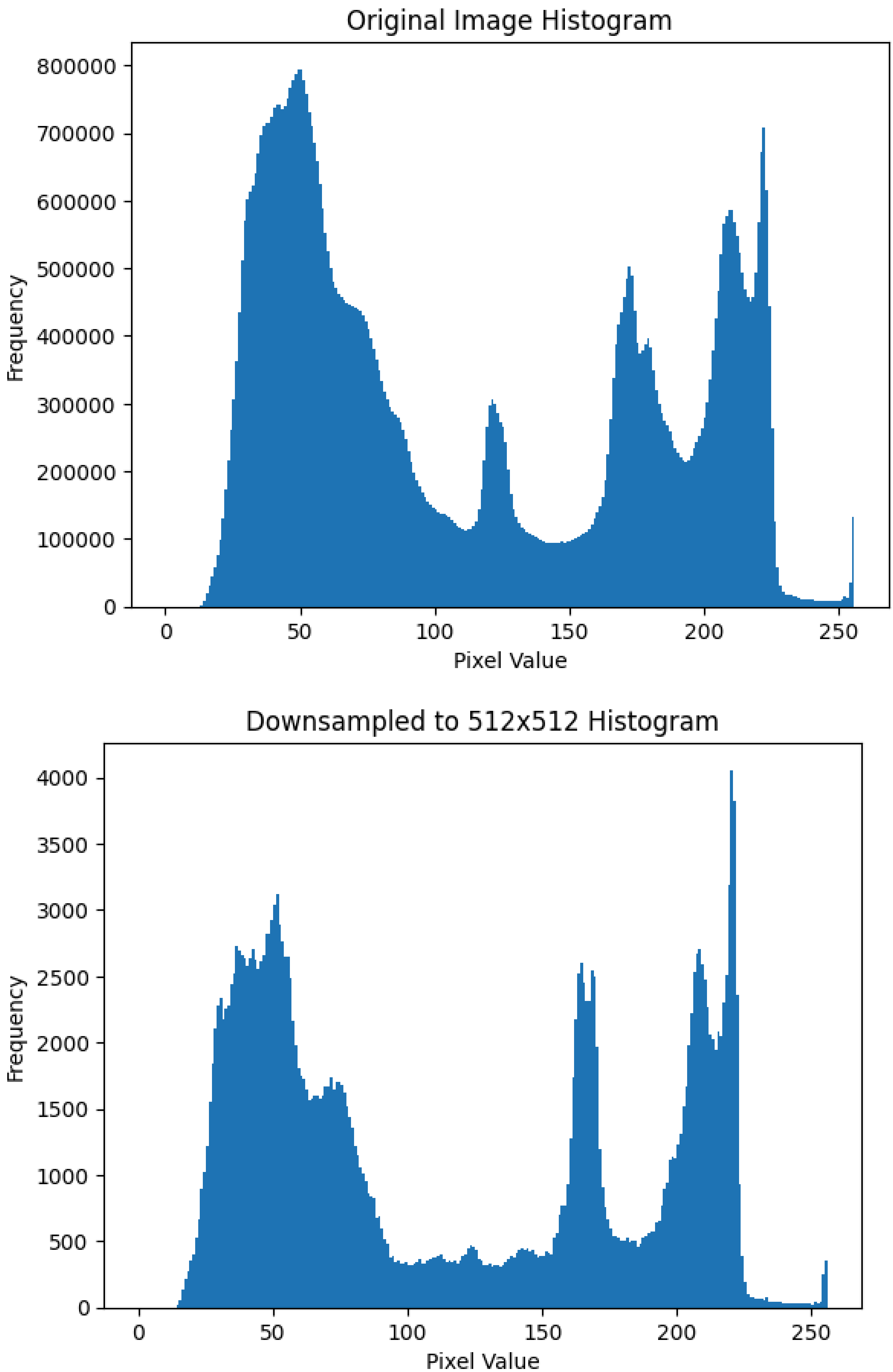
Quantized to 256 levels

Quantized to 64 levels

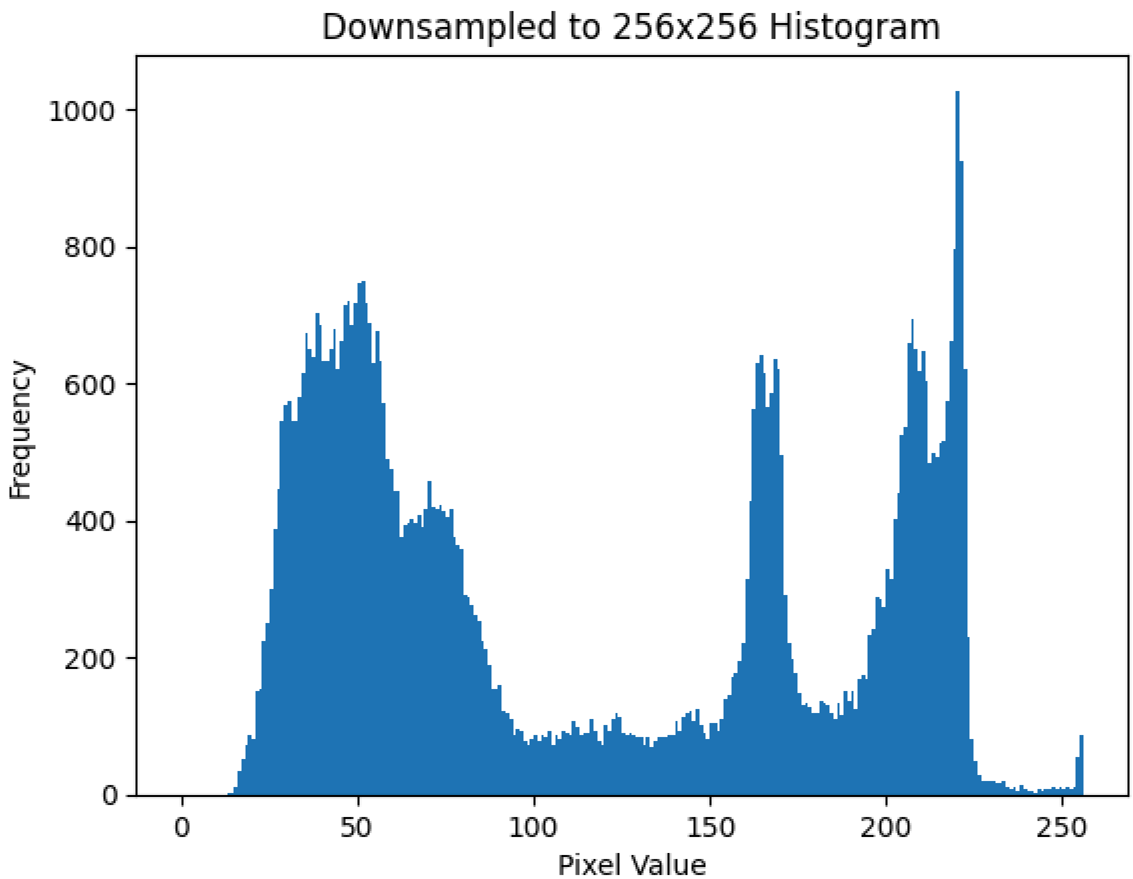
Quantized to 4 levels

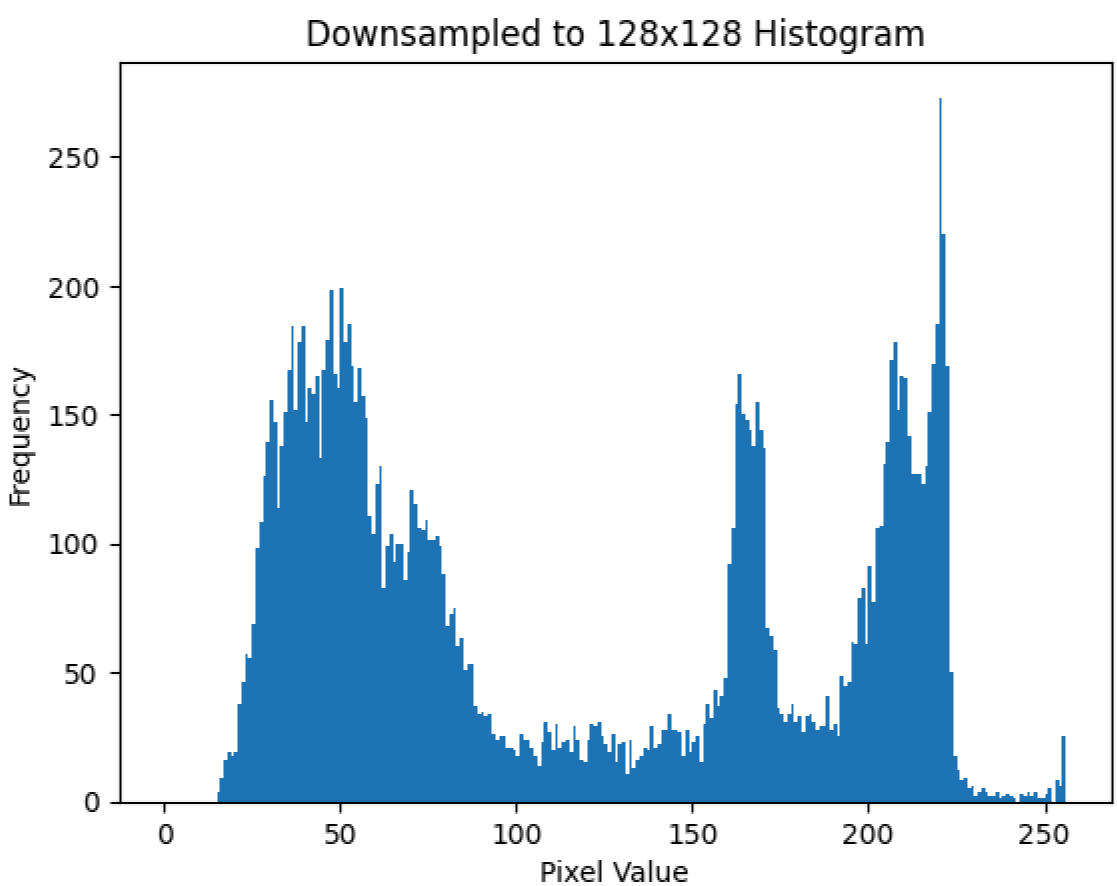
VISUAL AND QUANTITATIVE ANALYSIS

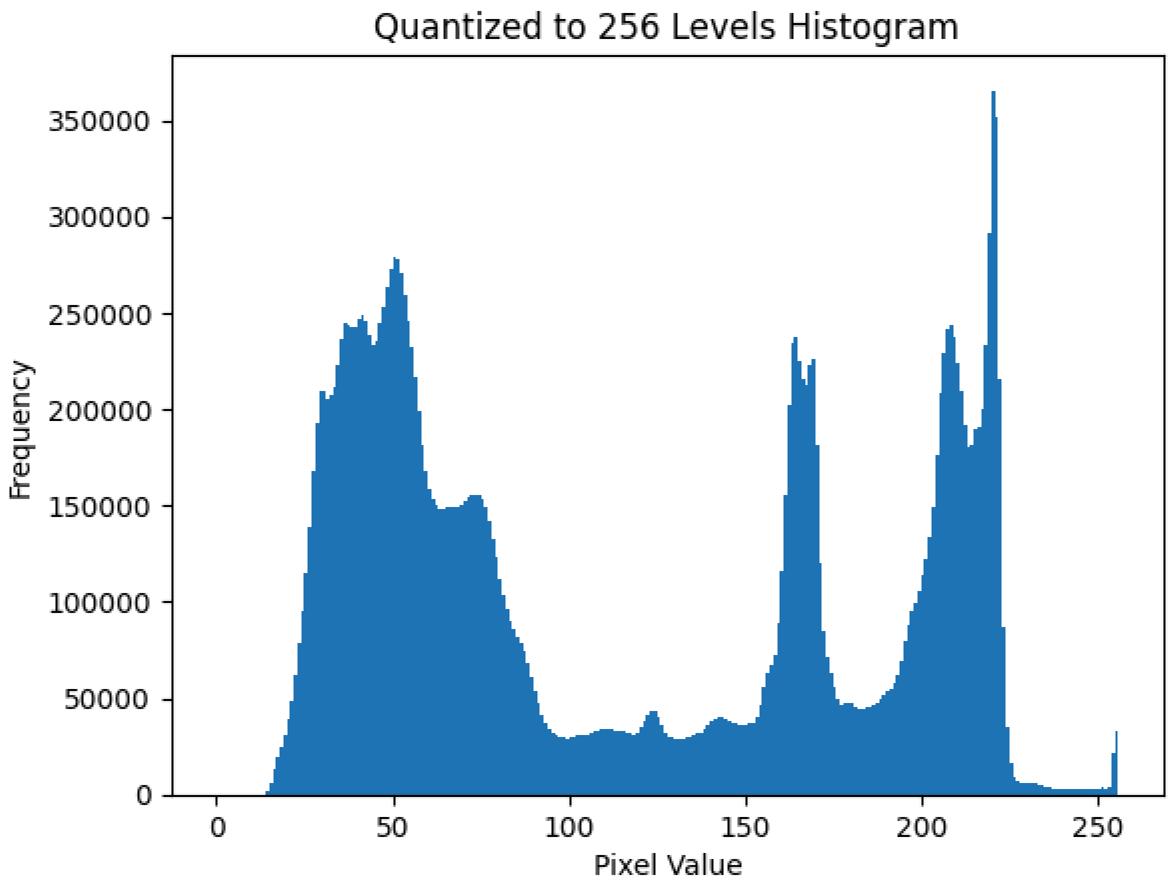
In [12]:
```
def plot_histogram(image, title):
    plt.hist(image.ravel(), bins=256, range=[0, 256])
    plt.title(title)
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')
    plt.show()

# Plot histogram for original grayscale image
plot_histogram(image, "Original Image Histogram")

# Plot histograms for downsampled images
for i, res in enumerate(resolutions):
    plot_histogram(downsampled_images[i], f"Downsampled to {res}x{res} Histo

# Plot histograms for quantized images
for i, levels in enumerate(quantization_levels):
    plot_histogram(quantized_images[i], f"Quantized to {levels} Levels Histo
```

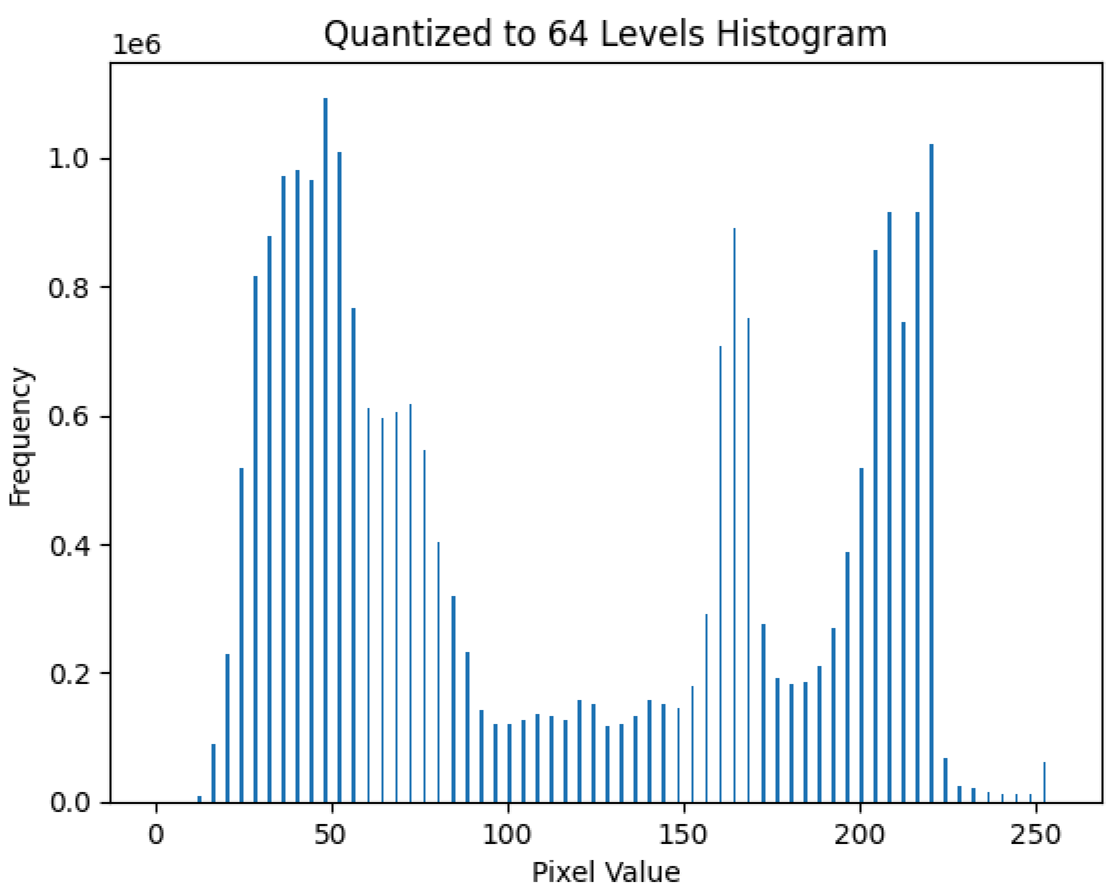


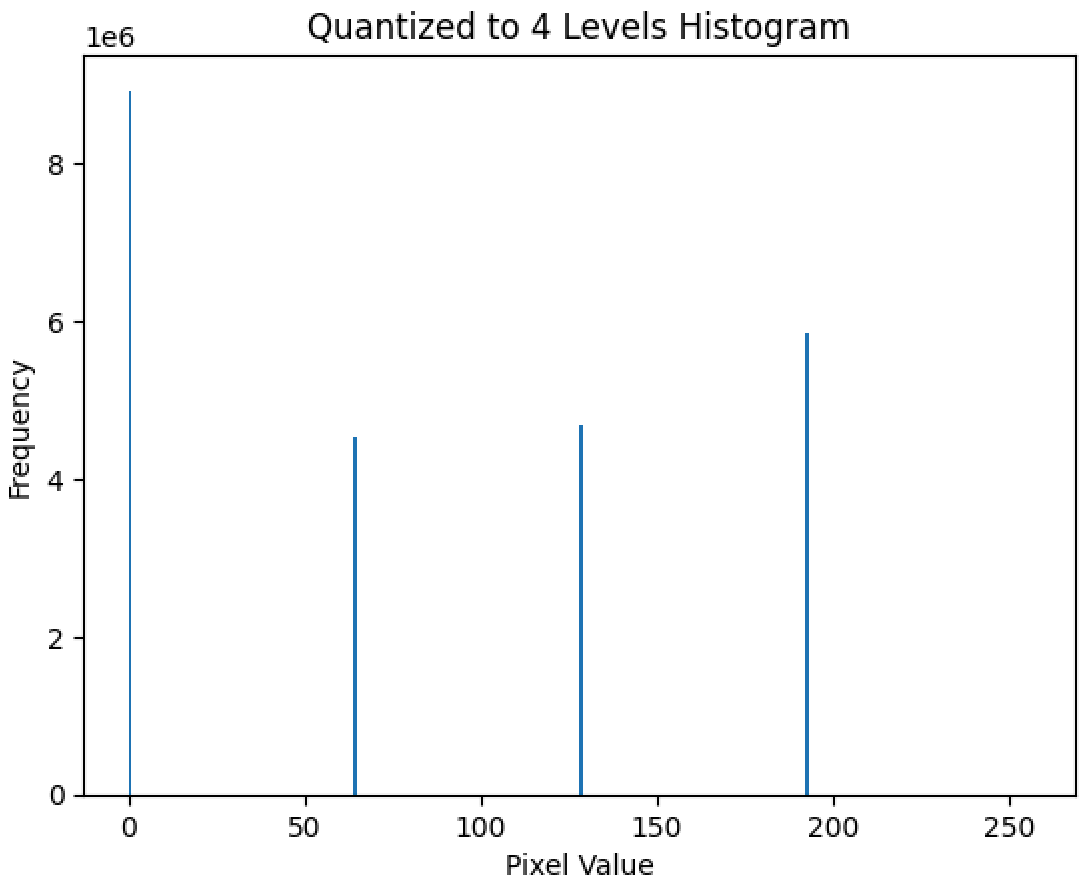
Original Image Histogram
Frequency
Pixel Value
0
100000
200000
300000
400000
500000
600000
700000
800000
0
50
100
150
200
250
Downsampled to 512x512 Histogram
Frequency
Pixel Value
0
500
1000
1500
2000
2500
3000
3500
4000
0
50
100
150
200
250


Downsampled to 256x256 Histogram
Frequency
Pixel Value
0
200
400
600
800
1000
0
50
100
150
200
250


Downsampled to 128x128 Histogram
Frequency
Pixel Value
0
50
100
150
200
250
0
50
100
150
200
250


Quantized to 256 Levels Histogram
Frequency
Pixel Value
0
50000
100000
150000
200000
250000
300000
350000
0
50
100
150
200
250


Quantized to 64 Levels Histogram
1e6
Frequency
Pixel Value
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250



ERROR METRICS

In [13]:
```
from sklearn.metrics import mean_squared_error

# Function to calculate MSE between two images
def calculate_mse(original, modified):
    return mean_squared_error(original.flatten(), modified.flatten())

# Calculate and print MSE for downsampled images
for i, res in enumerate(resolutions):
    mse_downsampled = calculate_mse(gray_image, cv2.resize(downsampled_image
    print(f"MSE between original and downsampled to {res}x{res}: {mse_downsa

# Calculate and print MSE for quantized images
for i, levels in enumerate(quantization_levels):
    mse_quantized = calculate_mse(gray_image, quantized_images[i])
    print(f"MSE between original and quantized to {levels} levels: {mse_quar
```

```
MSE between original and downsampled to 512x512: 19.656534375
MSE between original and downsampled to 256x256: 24.385822166666667
MSE between original and downsampled to 128x128: 31.01861975
MSE between original and quantized to 256 levels: 0.0
MSE between original and quantized to 64 levels: 3.48504725
MSE between original and quantized to 4 levels: 110.10873191666667
```

MSE will increase as image quality decreases (either due to downsampling or quantization).

Larger errors correspond to more visible artifacts and reduced image fidelity.

Comparing the MSE values will help you quantify the trade-offs in image quality when reducing resolution or quantization levels.