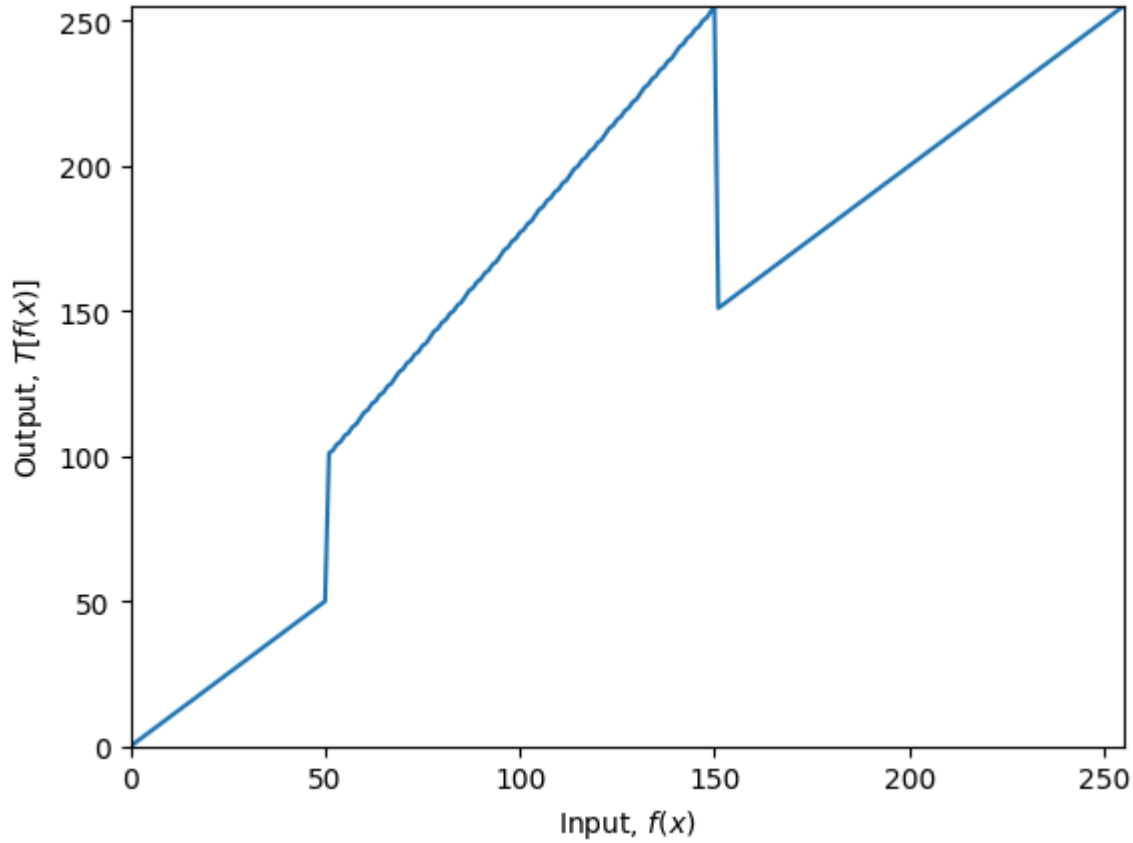
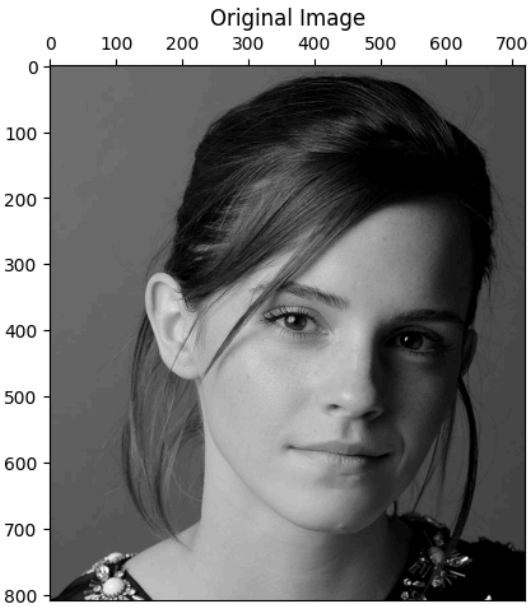
Implement the Intensity Transformation

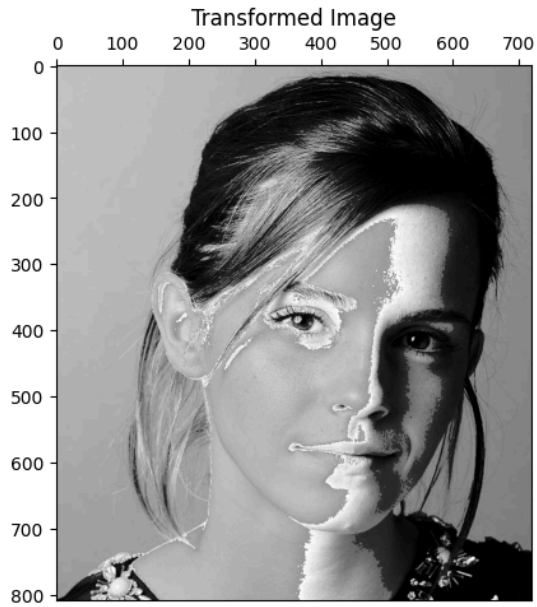
In [ ]:

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

c = np.array([(50,50),(50, 100), (150, 255), (150,150)])
t1 = np.linspace(0, c[0,1], c[0,0] + 1 - 0).astype('uint8')
print(len(t1))
t2 = np.linspace(c[0,1], c[1,1], c[1,0] - c[0,0]).astype('uint8')
print(len(t2))
t3 = np.linspace(c[1,1] + 1, c[2,1], c[2,0] - c[1,0]).astype('uint8')
print(len(t3))
t4 = np.linspace(c[2,1], c[3,1], c[3,0] - c[2,0]).astype('uint8')
print(len(t3))
t5 = np.linspace(c[3,1] + 1, 255, 255 - c[3,0]).astype('uint8')
print(len(t3))
transform = np.concatenate((t1, t2), axis=0).astype('uint8')
transform = np.concatenate((transform, t3), axis=0).astype('uint8')
transform = np.concatenate((transform, t4), axis=0).astype('uint8')
transform = np.concatenate((transform, t5), axis=0).astype('uint8')
print(len(transform))

plt.plot(transform)
plt.xlabel('Input, $f(x)$')
plt.ylabel('Output, $T[f(x)]$')
plt.xlim(0,255)
plt.ylim(0,255)
# plt.grid(color='black', linestyle='-', linewidth=1)
plt.savefig('transform.png')
plt.show()

img_orig = cv.imread(r"C:\Users\samko\Downloads\a1images\emma.jpg", cv.IMREAD_GR
image_transformed = cv.LUT(img_orig, transform)

plt.figure(figsize=(10, 5))


plt.subplot(1, 2, 1)
ax = plt.gca()
ax.xaxis.set_ticks_position('top')

plt.title("Original Image")
plt.imshow(img_orig, cmap='gray')


plt.subplot(1, 2, 2)
ax = plt.gca()
ax.xaxis.set_ticks_position('top')

plt.title("Transformed Image")
plt.imshow(image_transformed, cmap='gray')


plt.tight_layout()
plt.savefig('EMMA.png')
plt.show()
```

```
51
0
100
100
100
256
```





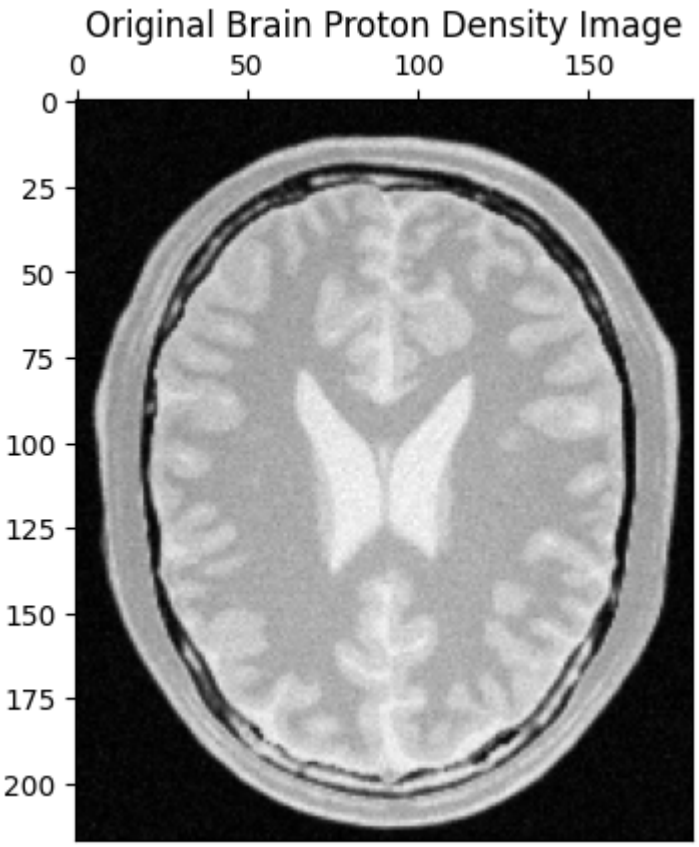


Intensity Transformation of a Brain Proton Density Slice

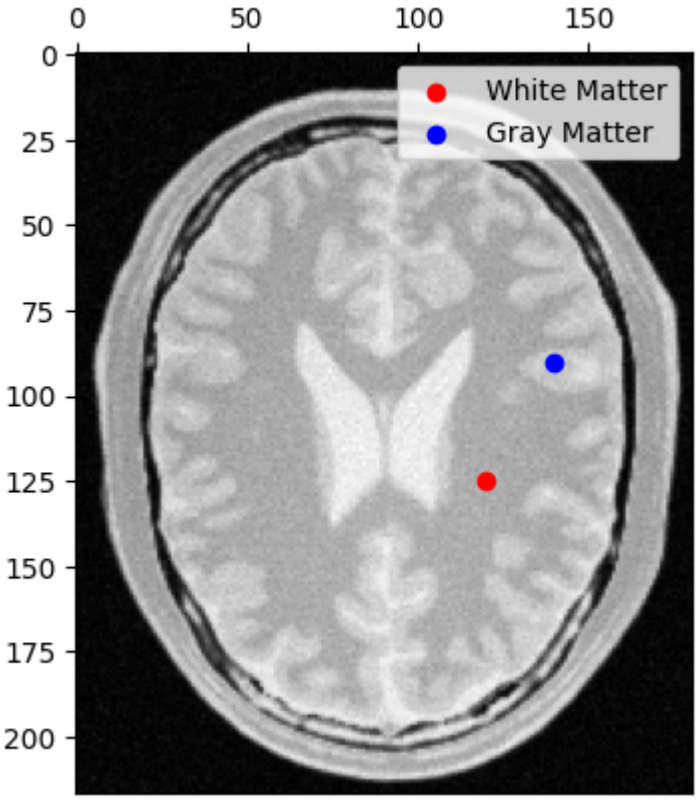
In [1]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Load the brain proton density image in grayscale
brain_image = cv2.imread(r"C:\Users\samko\Downloads\a1images\brain_proton_densit
```

```
# Display the original brain image
plt.imshow(brain_image, cmap='gray')
plt.title("Original Brain Proton Density Image")
plt.savefig("Original Brain Proton Density Image")
ax = plt.gca()
ax.xaxis.set_ticks_position('top')
plt.show()

# Output the shape of the image
print(brain_image.shape)

# Coordinates for sampling intensities
white_matter_coords = (120, 125)
gray_matter_coords = (140, 90)

# Show locations of white and gray matter in the image
plt.imshow(brain_image, cmap="gray")
plt.scatter(white_matter_coords[0], white_matter_coords[1], color='red', label='
plt.scatter(gray_matter_coords[0], gray_matter_coords[1], color='blue', label='G
plt.legend()
ax = plt.gca()
ax.xaxis.set_ticks_position('top')
plt.show()

# Retrieve pixel intensities at specified points
white_matter_intensity = brain_image[white_matter_coords]
gray_matter_intensity = brain_image[gray_matter_coords]

# Output pixel intensities for white and gray matter
print(f"White Matter Intensity: {white_matter_intensity}")
print(f"Gray Matter Intensity: {gray_matter_intensity}")

# Function to apply intensity transformations for highlighting white and gray ma
def transform_intensities(image):
    # Copy the input image for transformation
    transformed_img = np.copy(image)

    # Apply transformation for gray matter intensities (186 to 250)
    gray_matter_mask = (image >= 186) & (image <= 250)
    transformed_img[gray_matter_mask] = 1.75 * image[gray_matter_mask] + 30

    # Apply transformation for white matter intensities (150 to 185)
    white_matter_mask = (image >= 150) & (image <= 185)
    transformed_img[white_matter_mask] = 1.55 * image[white_matter_mask] + 22.5

    return transformed_img, white_matter_mask, gray_matter_mask

# Apply the transformation to the brain image
transformed_img, white_matter_mask, gray_matter_mask = transform_intensities(bra

# Display the transformed image with enhanced white and gray matter
plt.imshow(transformed_img, cmap='gray')
ax = plt.gca()
ax.xaxis.set_ticks_position('top')
plt.title("Transformed Brain Image: White and Gray Matter Enhanced")
plt.show()

# Display the white matter mask
plt.imshow(white_matter_mask, cmap='gray')
```

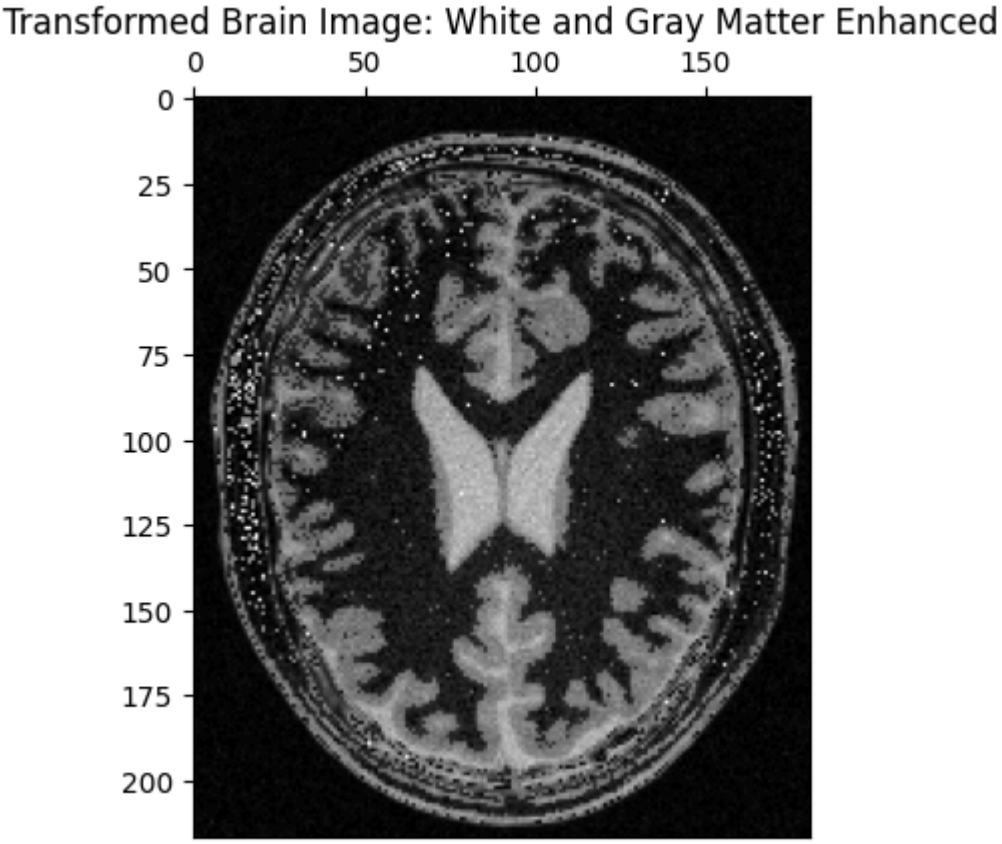
```
plt.title("White Matter")
ax = plt.gca()
ax.xaxis.set_ticks_position('top')
plt.savefig("White Matter Transformation.png")
plt.show()

# Display the gray matter mask
plt.imshow(gray_matter_mask, cmap='gray')
plt.title("Gray Matter")
ax = plt.gca()
ax.xaxis.set_ticks_position('top')
plt.savefig("Gray Matter Transformation.png")
plt.show()

# Create intensity transformation curves for both white and gray matter
x_vals = np.arange(0, 256)  # Intensity range (0-255)
gray_transformed = np.array([1.75 * x + 30 if 186 <= x <= 250 else x for x in x_
white_transformed = np.array([1.55 * x + 22.5 if 150 <= x <= 185 else x for x in

# Plot the transformation curves for both white and gray matter
plt.figure(figsize=(10, 6))
plt.plot(x_vals, white_transformed, label='White Matter Transformation', color='
plt.plot(x_vals, gray_transformed, label='Gray Matter Transformation', color='bl
plt.plot(x_vals, x_vals, label='Original Intensity', linestyle='--', color='yell
plt.title('Intensity Transformation Curves for White and Gray Matter')
plt.xlabel('Original Intensity')
plt.ylabel('Transformed Intensity')
plt.legend()
plt.grid(True)
plt.savefig("Intensity Transformation Curves for White and Gray Matter.png")
plt.show()
```

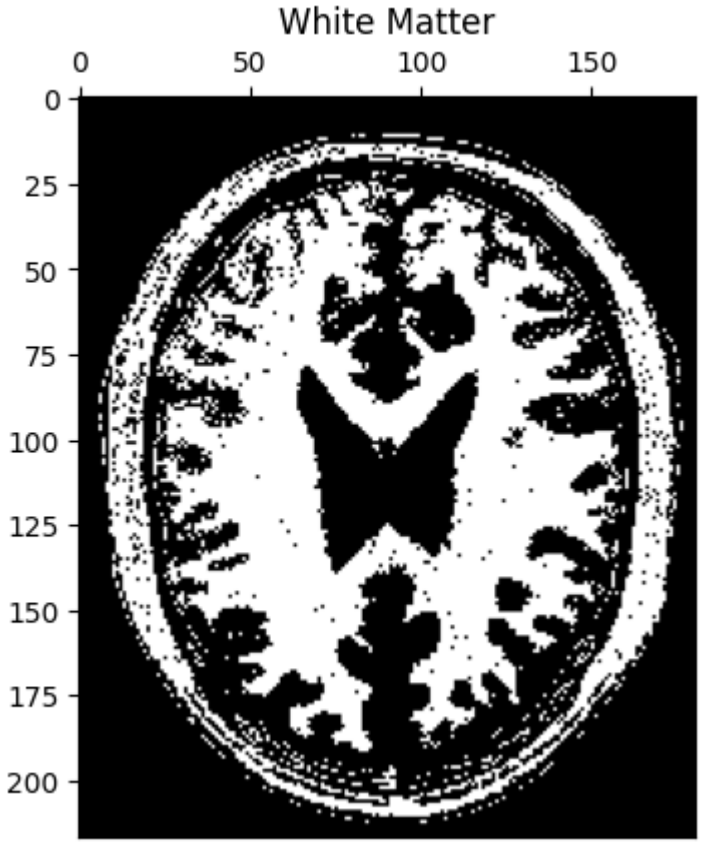


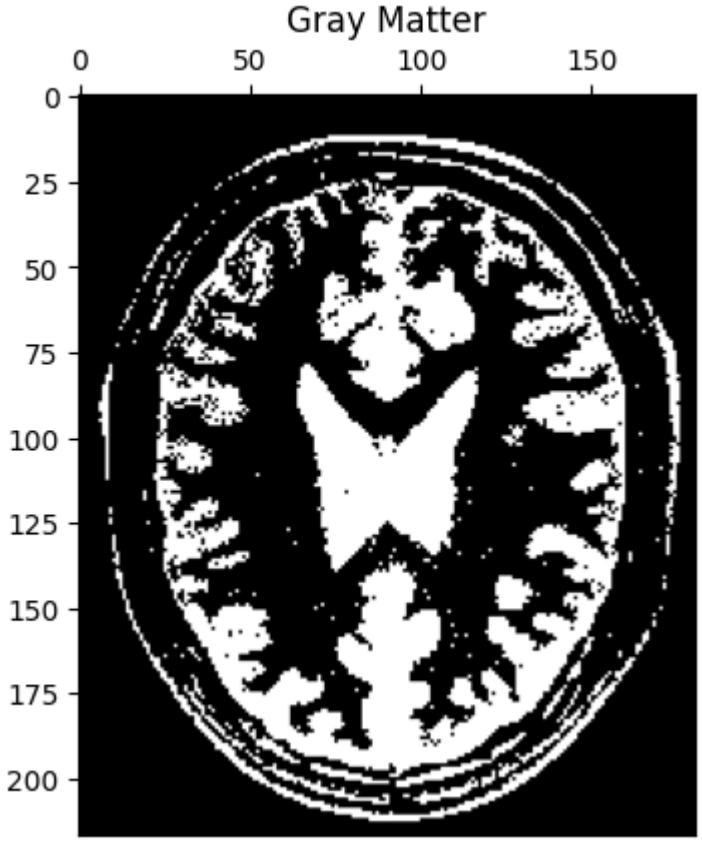

```
(217, 181)
```

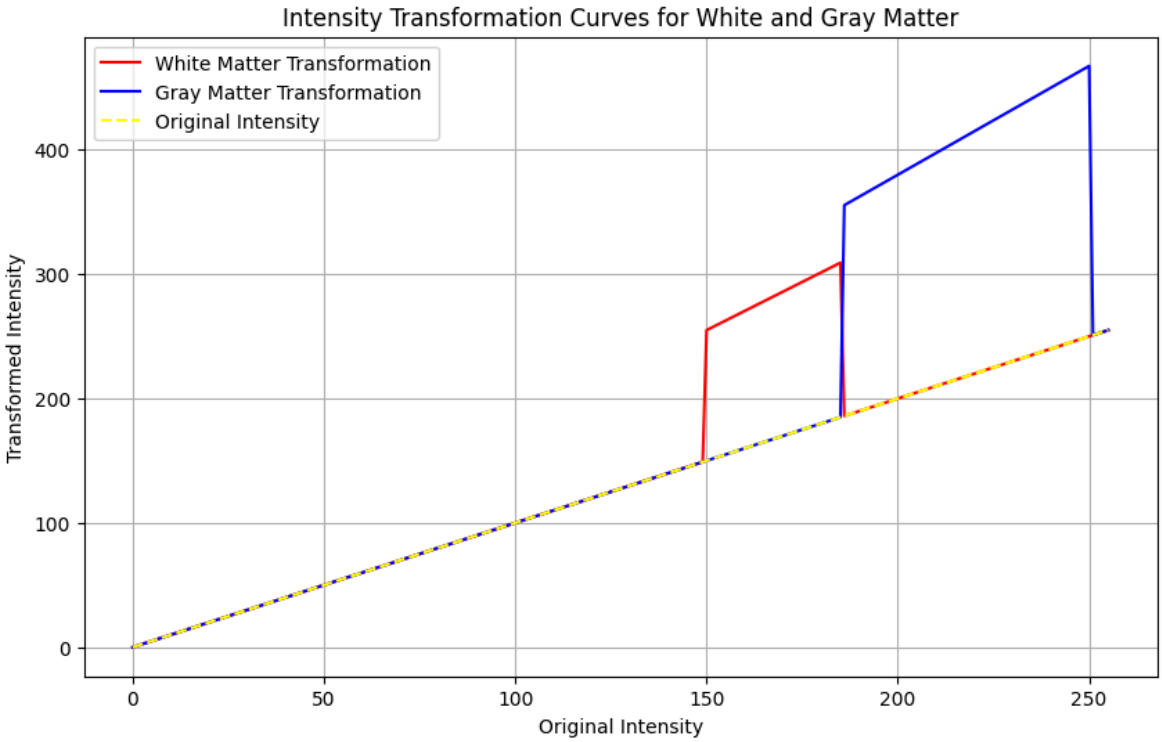



```
White Matter Intensity: 164
Gray Matter Intensity: 197
```




White Matter
0
50
100
150
0
25
50
75
100
125
150
175
200

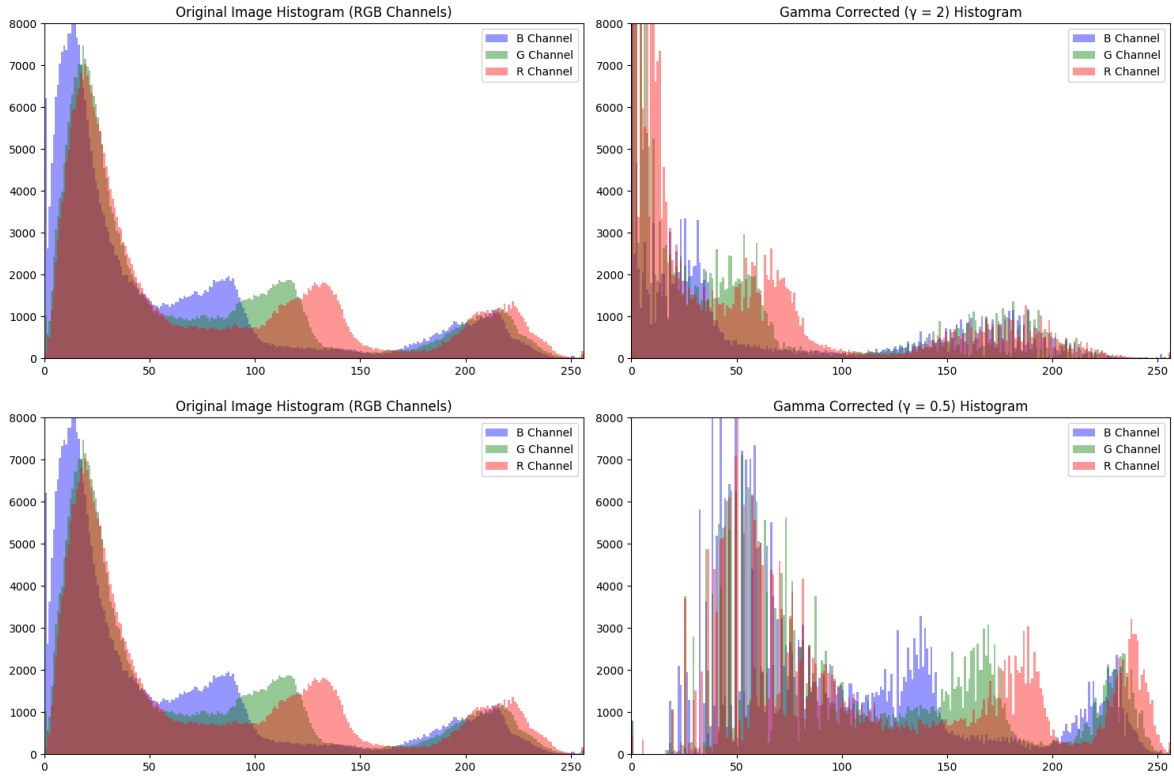

Gray Matter
0
50
100
150
0
25
50
75
100
125
150
175
200



Gamma Correction

In [16]:
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the input image for gamma correction
bgr_image = cv2.imread(r"C:\Users\samko\Downloads\a1images\highlights_and_shadow

# Convert the image from BGR to LAB color space
lab_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2LAB)

# Split the LAB image into L, A, and B channels
L_channel, A_channel, B_channel = cv2.split(lab_image)

# Normalize the L channel to the range [0, 1] for gamma correction
L_normalized = L_channel / 255.0

def apply_gamma_correction(L_norm, gamma):
    L_gamma = np.power(L_norm, gamma)
    return np.uint8(L_gamma * 255)

# Gamma values to apply
gammas = [0.5, 2]

# Store gamma corrected images
gamma_corrected_images = []

for g in gammas:
    L_gamma = apply_gamma_correction(L_normalized, g)
    lab_gamma_corrected = cv2.merge((L_gamma, A_channel, B_channel))
    gamma_corrected_bgr = cv2.cvtColor(lab_gamma_corrected, cv2.COLOR_LAB2BGR)
    gamma_corrected_images.append(gamma_corrected_bgr)

# Prepare images and titles for display
images = [
```

```
    (gamma_corrected_images[0], "Gamma Corrected (\u03B3 = 0.5)"),
    (bgr_image, "Original Image"),
    (gamma_corrected_images[1], "Gamma Corrected (\u03B3 = 2)")
]

# Display images side-by-side
fig, axes = plt.subplots(1, 3, figsize=(12, 6))

for ax, (img, title) in zip(axes, images):
    ax.xaxis.set_ticks_position("top")
    ax.set_title(title)
    ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))


plt.show()

# Histogram plotting preparation
hist_data = [
    ("Original Image Histogram (RGB Channels)", bgr_image),
    ("Gamma Corrected (\u03B3 = 2) Histogram", gamma_corrected_images[1]),
    ("Gamma Corrected (\u03B3 = 0.5) Histogram", gamma_corrected_images[0])
]

# Plot histograms in pairs: (Original + Gamma=2), (Original + Gamma=0.5)
pairs = [
    [hist_data[0], hist_data[1]],
    [hist_data[0], hist_data[2]]
]

for pair in pairs:
    fig, axs = plt.subplots(1, 2, figsize=(15, 5))
    for ax, (title, img) in zip(axs, pair):
        ax.set_title(title)
        colors = ['b', 'g', 'r']
        for i, color in enumerate(colors):
            ax.hist(img[:, :, i].flatten(), bins=256, range=[0, 256], color=colo
        ax.set_xlim([0, 256])
        ax.set_ylim([0, 8000])
        ax.legend()

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



Increasing the Vibrance of a Photograph

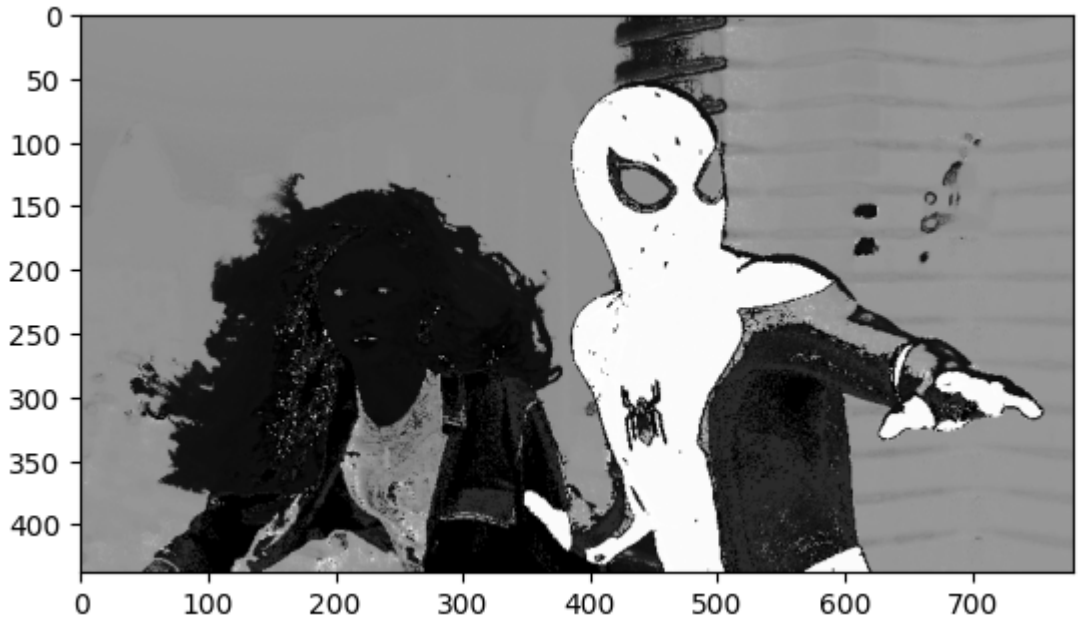
In [22]:

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

# === Load the image ===
spiderman_image = cv2.imread(r"C:\Users\samko\Downloads\a1images\spider.png")

# Convert the image to HSV color space
hsv_spiderman_image = cv2.cvtColor(spiderman_image, cv2.COLOR_BGR2HSV)

# Display shape of the image in HSV space
print(hsv_spiderman_image.shape)

# Display the hue channel
plt.imshow(hsv_spiderman_image[:, :, 0], cmap="gray")
plt.show()

# Extract the individual H, S, and V planes
hue_channel = hsv_spiderman_image[:, :, 0]
saturation_channel = hsv_spiderman_image[:, :, 1]
value_channel = hsv_spiderman_image[:, :, 2]

# Display the min and max values of each HSV plane
print(f"Min and Max of Hue Channel: {np.min(hue_channel)}, {np.max(hue_channel)}
print(f"Min and Max of Saturation Channel: {np.min(saturation_channel)}, {np.max
print(f"Min and Max of Value Channel: {np.min(value_channel)}, {np.max(value_cha

# === Define the vibrancy transformation function ===
def vibrancy_transformation(input_value: int, alpha: float, sigma: int = 70) ->
    x = input_value
    return min(
        x + alpha * 128 * math.exp(-((x - 128) ** 2) / (2 * sigma ** 2)),
```

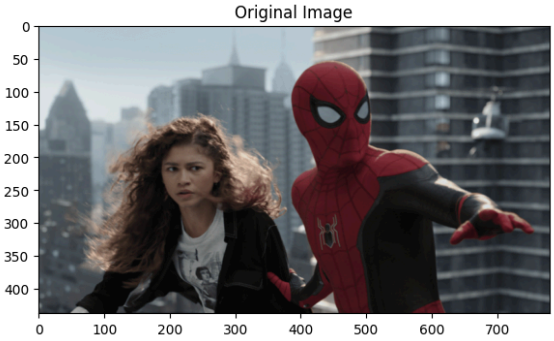
```
        255
    )

# Dictionary to store transformed images for different 'a' values
vibrancy_results = {0: None, 0.25: None, 0.5: None, 0.75: None, 1: None}

# === Apply vibrancy transformation for each alpha value ===
for alpha in vibrancy_results.keys():
    new_saturation = np.zeros(saturation_channel.shape, dtype=np.uint8)

    # Transform each pixel in the saturation plane
    for i in range(saturation_channel.shape[0]):
        for j in range(saturation_channel.shape[1]):
            new_saturation[i, j] = vibrancy_transformation(saturation_channel[i,

    # Combine H, new S, and V
    new_hsv_image = cv2.merge((hue_channel, new_saturation, value_channel))
    new_spiderman_image = cv2.cvtColor(new_hsv_image, cv2.COLOR_HSV2BGR)

    vibrancy_results[alpha] = new_spiderman_image

    # Show original and transformed images
    fig, axs = plt.subplots(1, 2, figsize=(15, 10))
    axs[0].set_title("Original Image")
    axs[0].imshow(cv2.cvtColor(spiderman_image, cv2.COLOR_BGR2RGB))
    axs[1].set_title(f"Vibrancy Transformed Image (a={alpha})")
    axs[1].imshow(cv2.cvtColor(new_spiderman_image, cv2.COLOR_BGR2RGB))
    plt.show()

# === Applying vibrancy transformation with alpha = 0.45 ===
alpha = 0.45
new_saturation_plane = np.zeros(saturation_channel.shape, dtype=np.uint8)

for i in range(saturation_channel.shape[0]):
    for j in range(saturation_channel.shape[1]):
        new_saturation_plane[i, j] = vibrancy_transformation(saturation_channel[

new_hsv_image = cv2.merge((hue_channel, new_saturation_plane, value_channel))
final_spiderman_image = cv2.cvtColor(new_hsv_image, cv2.COLOR_HSV2BGR)

# Show original vs final enhanced image
fig, axs = plt.subplots(1, 2, figsize=(15, 10))
axs[0].set_title("Original Image")
axs[0].imshow(cv2.cvtColor(spiderman_image, cv2.COLOR_BGR2RGB))
axs[1].set_title(f"Vibrancy Enhanced Image (a={alpha})")
axs[1].imshow(cv2.cvtColor(final_spiderman_image, cv2.COLOR_BGR2RGB))
plt.show()

# Display min and max saturation values
print(f"Min and Max Saturation: {saturation_channel.min()}, {saturation_channel.

# === Generate and plot the vibrancy transformation curve ===
alpha_value = 0.45
sigma_value = 70
input_pixel_values = np.arange(0, 256)
output_pixel_values = [
    vibrancy_transformation(x, alpha_value, sigma_value) for x in input_pixel_va
]

plt.figure(figsize=(8, 6))
```

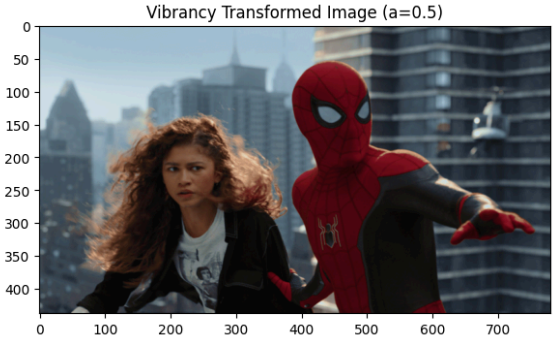
```
plt.plot(
    input_pixel_values, output_pixel_values,
    label=f'Vibrancy Transformation (a={alpha_value}, σ={sigma_value})',
    color='red'
)
plt.xlabel('Input Pixel Value')
plt.ylabel('Output Pixel Value')
plt.title('Vibrancy Transformation Curve for Saturation Plane')
plt.legend()
plt.grid(True)
plt.show()
```

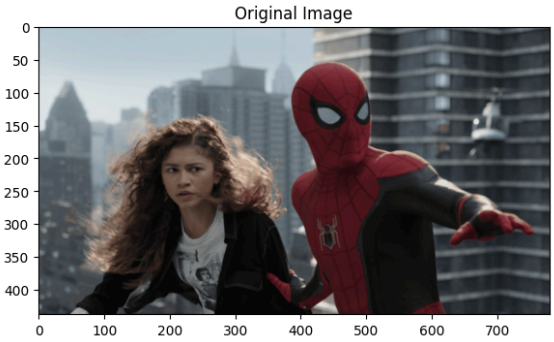
```
(438, 780, 3)
```

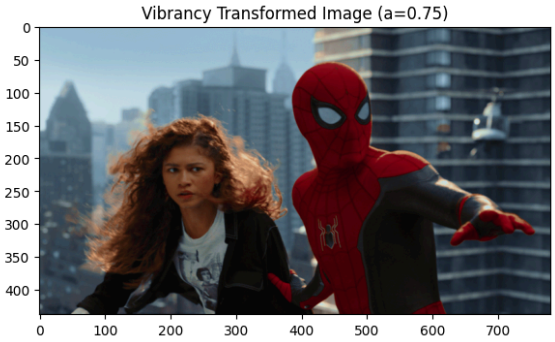


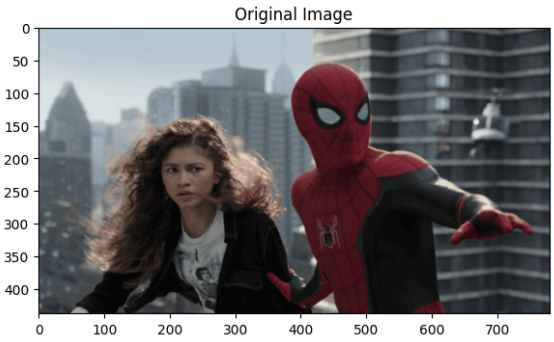

```
Min and Max of Hue Channel: 0, 179
Min and Max of Saturation Channel: 0, 185
Min and Max of Value Channel: 8, 231
```

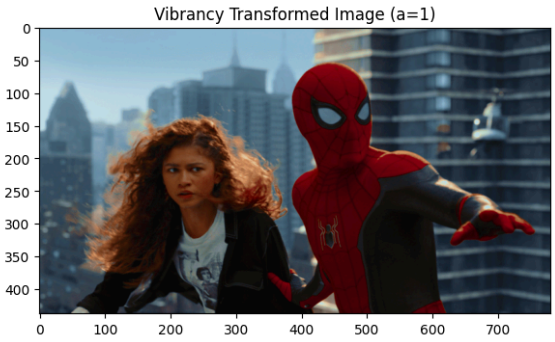


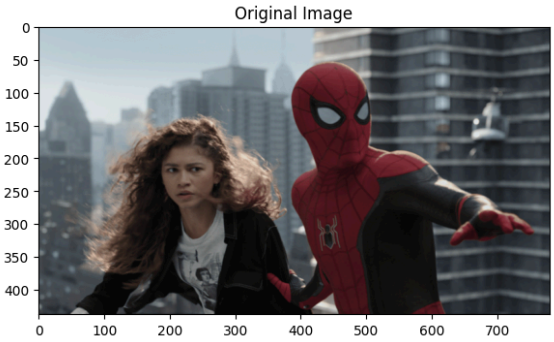


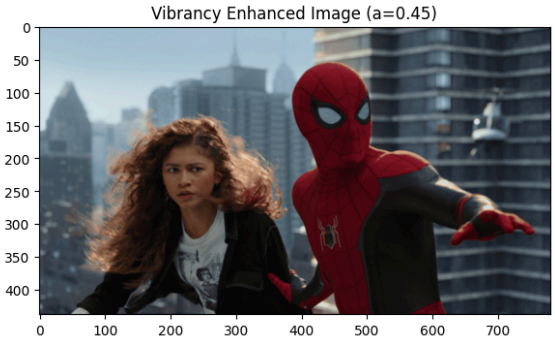




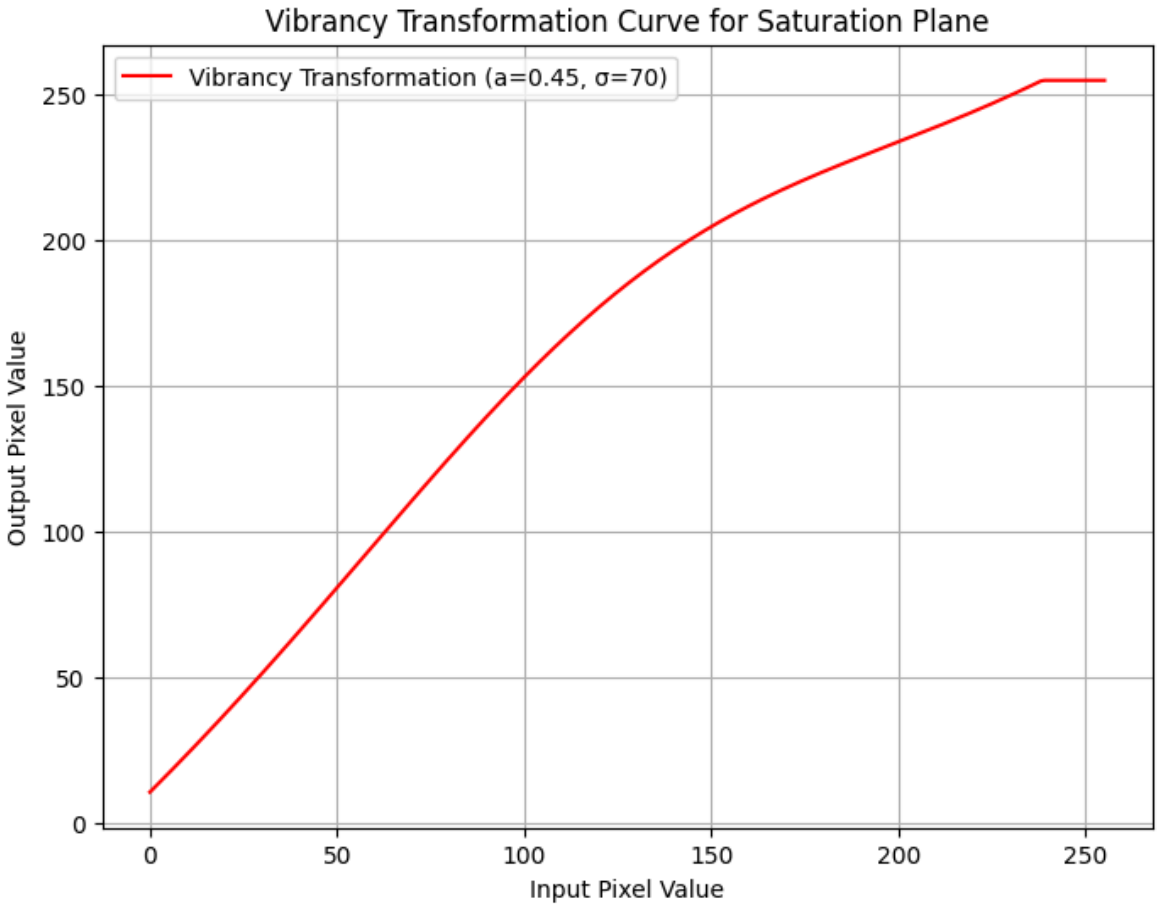








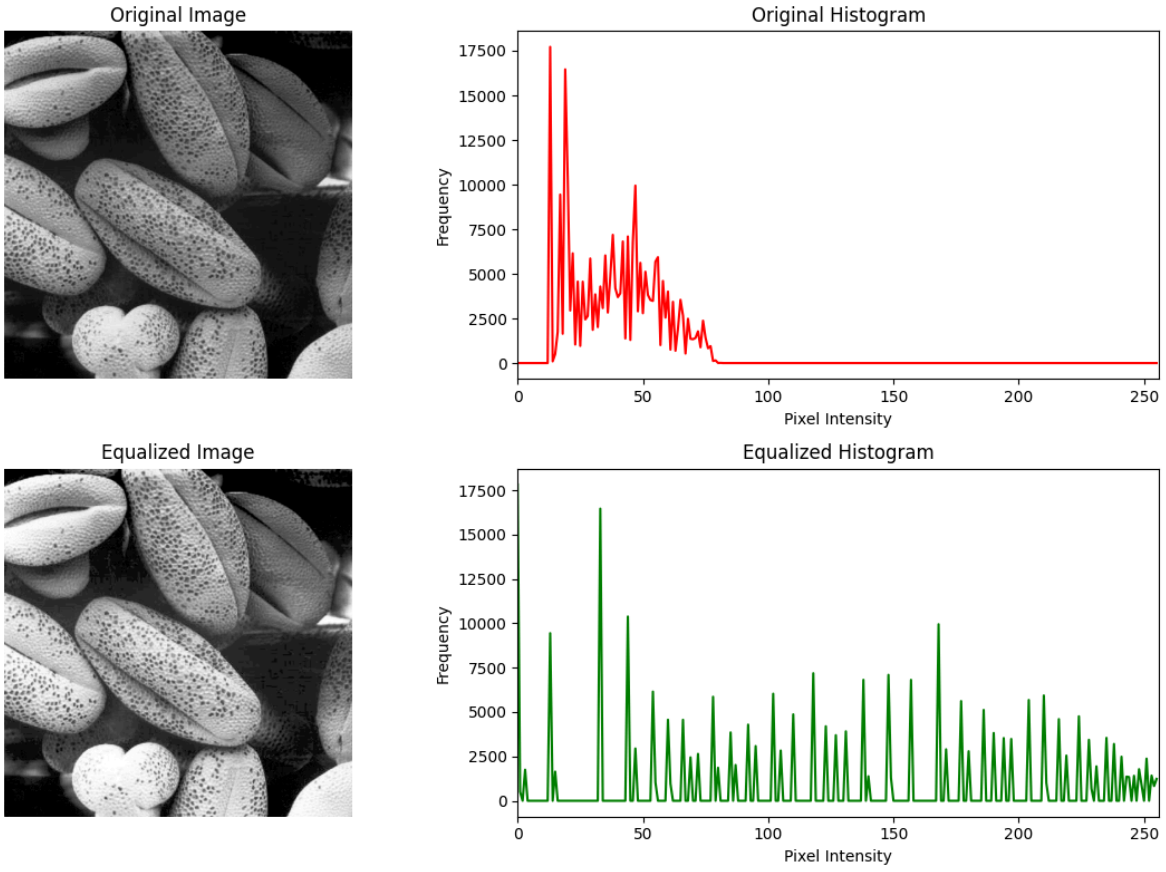
Min and Max Saturation: 0, 185



Histogram Equalization

In [28]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt

def apply_histogram_equalization(image):
    """
    Perform histogram equalization on a grayscale image using OpenCV's optimized
    """
    return cv2.equalizeHist(image)

def process_histogram_equalization(image_path):
    """
    Read image, convert to grayscale, and apply histogram equalization.
    Returns original grayscale image, equalized image, and original histogram.
    """
    original = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    equalized = apply_histogram_equalization(original)
    hist = cv2.calcHist([original], [0], None, [256], [0, 256]).flatten()
    return original, equalized, hist

def display_results(original, equalized, original_hist):
    """
    Display original and equalized images with their histograms.
    """
    equalized_hist = cv2.calcHist([equalized], [0], None, [256], [0, 256]).flatt

    plt.figure(figsize=(12, 8))

    plt.subplot(221)
```

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

    plt.subplot(222)
    plt.plot(original_hist, color='red')
    plt.title('Original Histogram')
    plt.xlim([0, 256])
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')

    plt.subplot(223)
    plt.imshow(equalized, cmap='gray')
    plt.title('Equalized Image')
    plt.axis('off')

    plt.subplot(224)
    plt.plot(equalized_hist, color='green')
    plt.title('Equalized Histogram')
    plt.xlim([0, 256])
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    image_path = r"C:\Users\samko\Downloads\a1images\shells.tif"
    original_img, equalized_img, orig_hist = process_histogram_equalization(imag
    display_results(original_img, equalized_img, orig_hist)
```



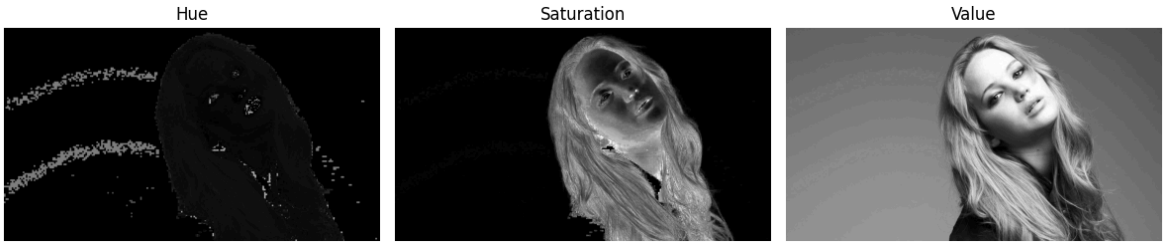
Histogram Equalized Foreground

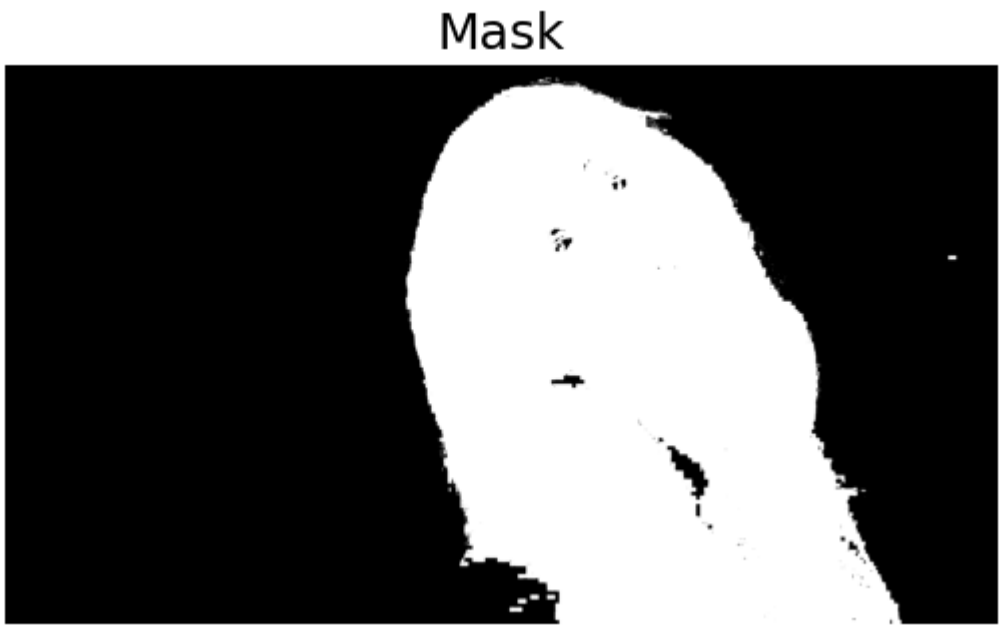
In [42]:

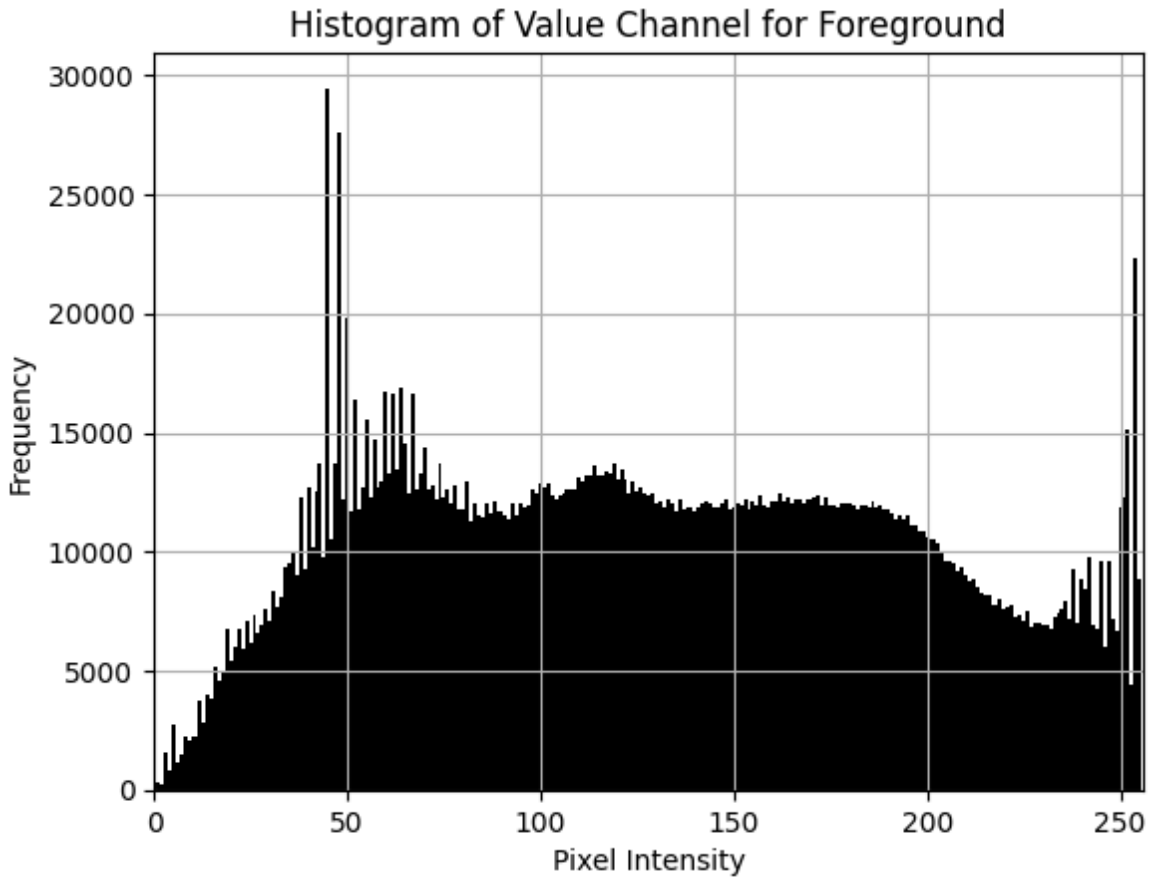
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

# --- Load and Convert Image ---
jennifer = cv.imread(r"C:\Users\samko\Downloads\a1images\jeniffer.jpg")
assert jennifer is not None
jennifer_hsv = cv.cvtColor(jennifer, cv.COLOR_BGR2HSV)
jennifer_rgb = cv.cvtColor(jennifer, cv.COLOR_BGR2RGB)

# --- Split into HSV Planes and Display ---
H, S, V = cv.split(jennifer_hsv)
fig, ax = plt.subplots(1, 3, figsize=(12, 8))
ax[0].imshow(H, cmap='gray', vmin=0, vmax=255)
ax[0].set_title('Hue')
ax[0].axis('off')
ax[1].imshow(S, cmap='gray', vmin=0, vmax=255)
ax[1].set_title('Saturation')
ax[1].axis('off')
ax[2].imshow(V, cmap='gray', vmin=0, vmax=255)
ax[2].set_title('Value')
ax[2].axis('off')
plt.tight_layout()
plt.show()

# --- Threshold Saturation for Foreground Mask and Display ---
_, mask = cv.threshold(S, 12, 255, cv.THRESH_BINARY)
plt.imshow(mask, cmap='gray')
plt.title('Mask', fontsize=18)
plt.axis('off')
plt.show()

# --- Extract Foreground and Display ---
foreground = cv.bitwise_and(jennifer, jennifer, mask=mask)
plt.figure(figsize=(10, 5))
plt.imshow(cv.cvtColor(foreground, cv.COLOR_BGR2RGB))
plt.title('Extracted Foreground', fontsize=18)
plt.axis('off')
plt.show()

# --- Compute and Display Histogram of Foreground Value Channel ---
foreground_hsv = cv.cvtColor(foreground, cv.COLOR_BGR2HSV)
H_fg, S_fg, V_fg = cv.split(foreground_hsv)
hist = cv.calcHist([V_fg], [0], mask, [256], [0, 256])
x_positions = np.arange(len(hist))
plt.figure()
plt.bar(x_positions, hist.flatten(), color='black', width=1)
plt.title('Histogram of Value Channel for Foreground')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.xlim([0, 256])
plt.grid(True)
plt.show()

# --- Compute and Display Cumulative Sum ---
cdf = hist.cumsum()
plt.figure()
plt.plot(cdf, color='black')
plt.title('Cumulative Sum of the Histogram')
```

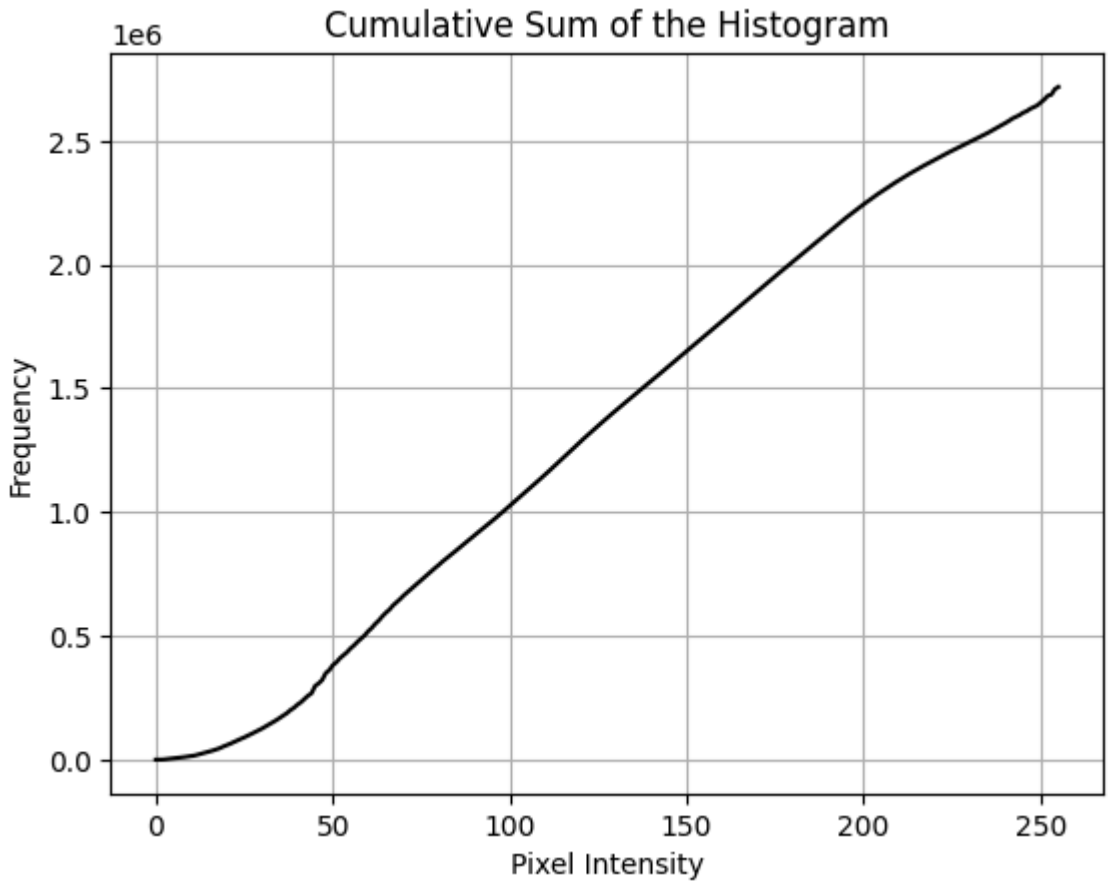
```
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# --- Histogram Equalization for Foreground Value Channel ---
pixels = cdf[-1]
t = np.array([(255) / pixels * cdf[k] for k in range(256)]).astype('uint8')
V_eq = t[V_fg]

# --- Compute and Display Histogram of Equalized Value Channel ---
hist_eq = cv.calcHist([V_eq], [0], mask, [256], [0, 256])
plt.figure()
plt.bar(np.arange(len(hist_eq)), hist_eq.flatten(), color='black', width=1)
plt.title('Equalized Histogram of Value Channel for Foreground')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.xlim([0, 256])
plt.grid(True)
plt.show()

# --- Merge and Convert Foreground to RGB ---
merged = cv.merge([H_fg, S_fg, V_eq])
foreground_modified = cv.cvtColor(merged, cv.COLOR_HSV2RGB)

# --- Extract and Convert Background to RGB ---
background = cv.bitwise_and(jennifer, jennifer, mask=cv.bitwise_not(mask))
background_rgb = cv.cvtColor(background, cv.COLOR_BGR2RGB)

# --- Prepare Equalized Foreground for Display (black background elsewhere) ---
eq_fg_display = np.zeros_like(background_rgb)
eq_fg_display[mask == 255] = foreground_modified[mask == 255]

# --- Prepare Final Result Image ---
final_result = background_rgb.copy()
final_result[mask == 255] = foreground_modified[mask == 255]

# --- DISPLAY FINAL IMAGES ONLY ---
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Equalized Foreground
axes[0].imshow(eq_fg_display)
axes[0].set_title("Equalized Foreground")
axes[0].set_xlabel("Pixel Position (X)")
axes[0].set_ylabel("Pixel Position (Y)")
axes[0].xaxis.set_ticks_position("top")
axes[0].tick_params(bottom=True, labelbottom=True, left=True, labelleft=True)

# Final Result
axes[1].imshow(final_result)
axes[1].set_title("Final Result")
axes[1].set_xlabel("Pixel Position (X)")
axes[1].set_ylabel("Pixel Position (Y)")
axes[1].xaxis.set_ticks_position("top")
axes[1].tick_params(bottom=True, labelbottom=True, left=True, labelleft=True)

# Set custom ticks
for ax in axes:
    ax.set_xticks(np.linspace(0, final_result.shape[1], 8, dtype=int))
    ax.set_yticks(np.linspace(0, final_result.shape[0], 8, dtype=int))
```

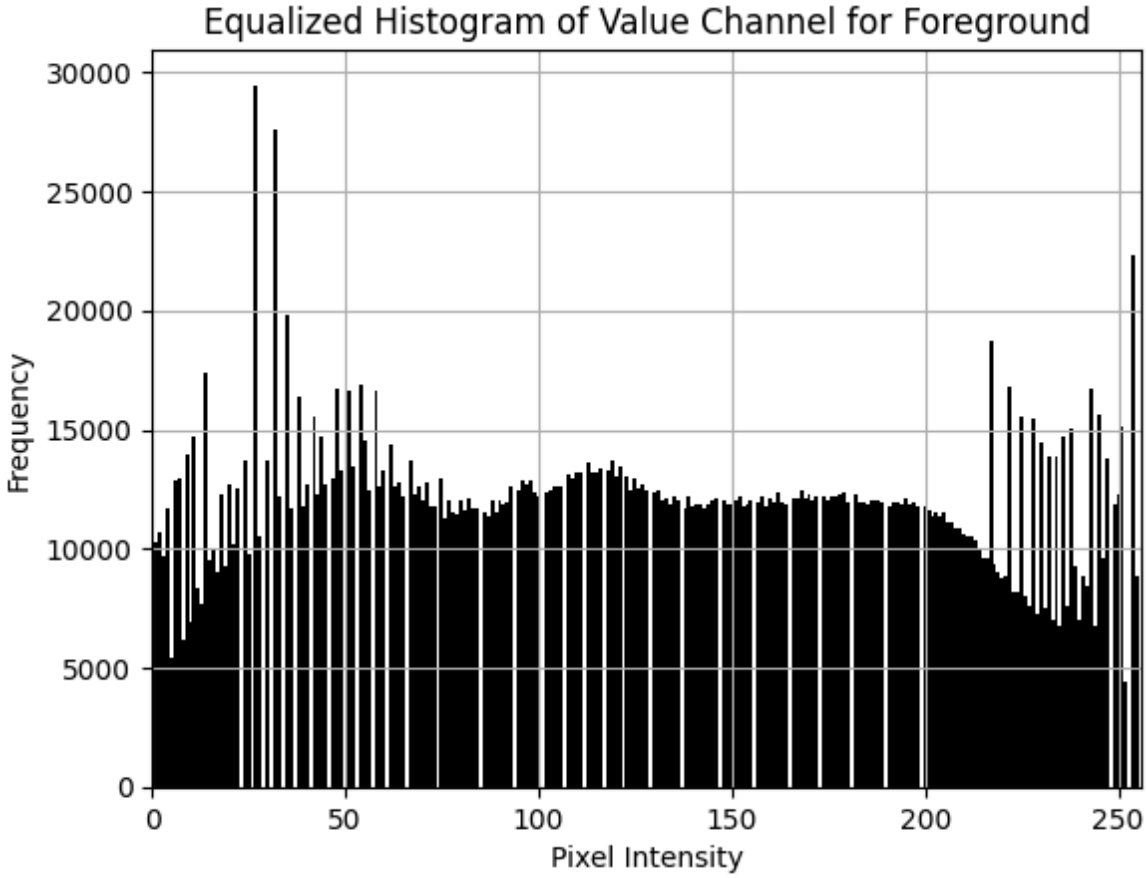
```
plt.tight_layout()
plt.show()
```

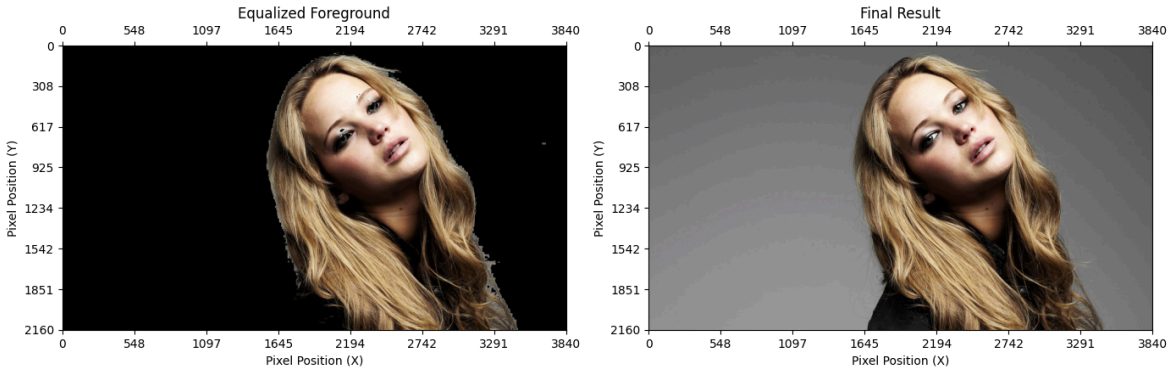





Histogram of Value Channel for Foreground
Frequency
Pixel Intensity
0
5000
10000
15000
20000
25000
30000
0
50
100
150
200
250


Cumulative Sum of the Histogram
1e6
Frequency
Pixel Intensity
0.0
0.5
1.0
1.5
2.0
2.5
0
50
100
150
200
250



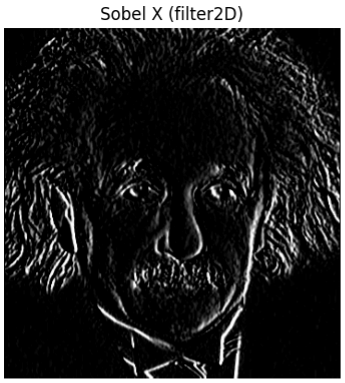


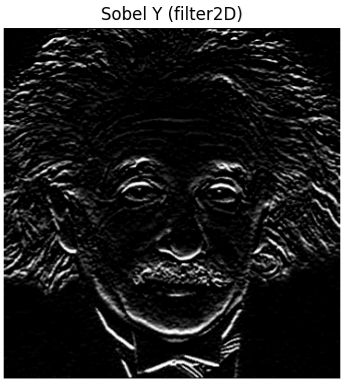
In [46]:

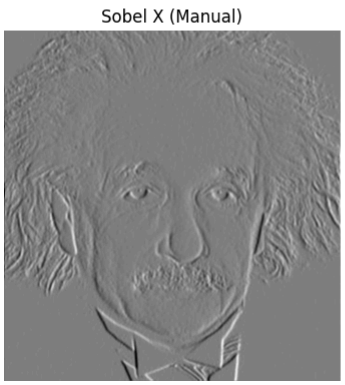
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# Load the image in grayscale
image_path = r"C:\Users\samko\Downloads\a1images\einstein.png"  # Replace with y
image = Image.open(image_path).convert('L')  # Convert image to grayscale
image_np = np.array(image)

# (a) Using OpenCV filter2D function for Sobel filtering
# Define Sobel kernels for x and y direction
sobel_x = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

# Apply Sobel filters using cv2.filter2D
sobel_x_filtered = cv2.filter2D(image_np, -1, sobel_x)
sobel_y_filtered = cv2.filter2D(image_np, -1, sobel_y)
```

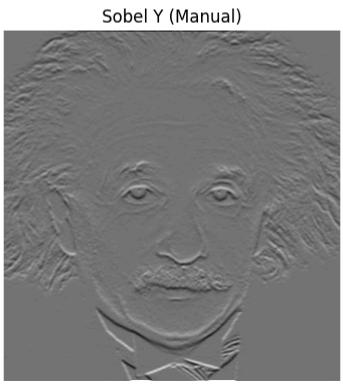
```python
# Compute the gradient magnitude
sobel_magnitude = np.sqrt(sobel_x_filtered**2 + sobel_y_filtered**2)

# Normalize the result
sobel_magnitude = np.uint8(sobel_magnitude / np.max(sobel_magnitude) * 255)

# (b) Write your own code to perform Sobel filtering (Manual convolution)
def sobel_filter_manual(image, kernel):
    # Get the dimensions of the image and kernel
    kernel_height, kernel_width = kernel.shape
    image_height, image_width = image.shape

    # Output image initialized to zeros
    output = np.zeros((image_height, image_width), dtype=np.float32)

    # Pad the image to handle the border effects
    padded_image = np.pad(image, ((1, 1), (1, 1)), mode='constant')

    # Convolution operation
    for i in range(image_height):
        for j in range(image_width):
            output[i, j] = np.sum(kernel * padded_image[i:i + kernel_height, j:j

    return output

# Apply manual Sobel filters
sobel_x_manual = sobel_filter_manual(image_np, sobel_x)
sobel_y_manual = sobel_filter_manual(image_np, sobel_y)

# Compute the gradient magnitude for manual method
sobel_magnitude_manual = np.sqrt(sobel_x_manual**2 + sobel_y_manual**2)

# Normalize the result
sobel_magnitude_manual = np.uint8(sobel_magnitude_manual / np.max(sobel_magnitud

# (c) Using the property of Sobel filter decomposition
sobel_1 = np.array([[1], [2], [1]])  # Vertical part
sobel_2 = np.array([[1, 0, -1]])     # Horizontal part

# Perform separable filtering
sobel_x_separable = cv2.sepFilter2D(image_np, -1, sobel_2, sobel_1)
sobel_y_separable = cv2.sepFilter2D(image_np, -1, sobel_1.T, sobel_2.T)

# Compute the gradient magnitude
sobel_magnitude_separable = np.sqrt(sobel_x_separable**2 + sobel_y_separable**2)
sobel_magnitude_separable = np.uint8(sobel_magnitude_separable / np.max(sobel_ma

# Display the results
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

axs[0, 0].imshow(sobel_x_filtered, cmap='gray')
axs[0, 0].set_title('Sobel X (filter2D)')
axs[0, 0].axis('off')

axs[0, 1].imshow(sobel_y_filtered, cmap='gray')
axs[0, 1].set_title('Sobel Y (filter2D)')
axs[0, 1].axis('off')

axs[0, 2].imshow(sobel_magnitude, cmap='gray')
axs[0, 2].set_title('Gradient Magnitude (filter2D)')
```

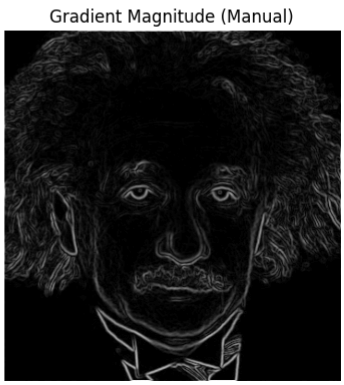
```
axs[0, 2].axis('off')

axs[1, 0].imshow(sobel_x_manual, cmap='gray')
axs[1, 0].set_title('Sobel X (Manual)')
axs[1, 0].axis('off')

axs[1, 1].imshow(sobel_y_manual, cmap='gray')
axs[1, 1].set_title('Sobel Y (Manual)')
axs[1, 1].axis('off')

axs[1, 2].imshow(sobel_magnitude_manual, cmap='gray')
axs[1, 2].set_title('Gradient Magnitude (Manual)')
axs[1, 2].axis('off')

plt.show()

# Display the separable Sobel results
fig, ax = plt.subplots(1, 3, figsize=(15, 5))

ax[0].imshow(sobel_x_separable, cmap='gray')
ax[0].set_title('Sobel X (Separable)')
ax[0].axis('off')

ax[1].imshow(sobel_y_separable, cmap='gray')
ax[1].set_title('Sobel Y (Separable)')
ax[1].axis('off')

ax[2].imshow(sobel_magnitude_separable, cmap='gray')
ax[2].set_title('Gradient Magnitude (Separable)')
ax[2].axis('off')

plt.show()
```

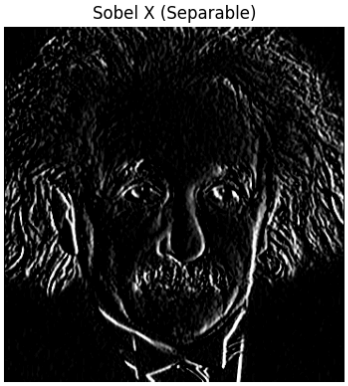


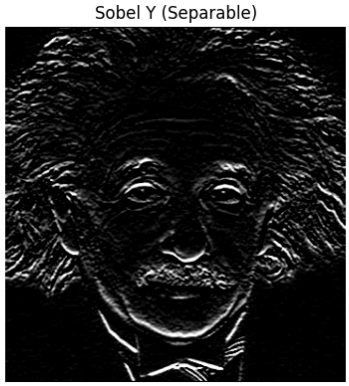












Zoom Images

In [69]:

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Function to zoom the image
def zoom_image(image, scale, interpolation):
    height, width = image.shape[:2]
    new_size = (int(width * scale), int(height * scale))
    return cv2.resize(image, new_size, interpolation=interpolation)

# Function to compute normalized SSD
def compute_normalized_ssd(img1, img2, bypass_size_error=True):
    if not bypass_size_error:
        assert img1.shape == img2.shape, "Images must be the same shape for SSD
    else:
        min_height = min(img1.shape[0], img2.shape[0])
        min_width = min(img1.shape[1], img2.shape[1])
        img1 = img1[:min_height, :min_width]
        img2 = img2[:min_height, :min_width]
    ssd = np.sum((img1.astype("float32") - img2.astype("float32")) ** 2)
    norm_ssd = ssd / np.prod(img1.shape)
    return norm_ssd

# Function to display images
def display_images(original, nearest, bilinear, titles):
    plt.figure(figsize=(15, 15))
    for idx, img in enumerate([original, nearest, bilinear]):
        plt.subplot(1, 3, idx+1)
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.title(titles[idx])
        plt.axis('off')
    plt.show()

# Main function
def get_zoom_and_original_img(small_img, big_img, scale_factor=4, bypass_size_er
    zoomed_nn = zoom_image(small_img, scale_factor, cv2.INTER_NEAREST)
    zoomed_bilinear = zoom_image(small_img, scale_factor, cv2.INTER_LINEAR)
    ssd_nn = compute_normalized_ssd(big_img, zoomed_nn, bypass_size_error=bypass
    ssd_bilinear = compute_normalized_ssd(big_img, zoomed_bilinear, bypass_size_
    print(f"Normalized SSD (Nearest Neighbor): {ssd_nn}")
    print(f"Normalized SSD (Bilinear): {ssd_bilinear}")
    display_images(big_img, zoomed_nn, zoomed_bilinear, ["Original Image", "Near

# ---- Example usage ----
# Small and big image paths
```

```
small_path_1 = r"C:\Users\samko\Downloads\a1images\a1q5images\im01small.png"
big_path_1   = r"C:\Users\samko\Downloads\a1images\a1q5images\im01.png"

small_path_2 = r"C:\Users\samko\Downloads\a1images\a1q5images\im02small.png"
big_path_2=r"C:\Users\samko\Downloads\a1images\a1q5images\im02.png"

small_img_1 = cv2.imread(small_path_1)
big_img_1  = cv2.imread(big_path_1)

small_img_2 = cv2.imread(small_path_2)
big_img_2   = cv2.imread(big_path_2)


get_zoom_and_original_img(small_img_1, big_img_1, scale_factor=4, bypass_size_er
get_zoom_and_original_img(small_img_2, big_img_2, scale_factor=4, bypass_size_er
```

```
Normalized SSD (Nearest Neighbor): 136.2690534979424
Normalized SSD (Bilinear): 115.09185185185186
```

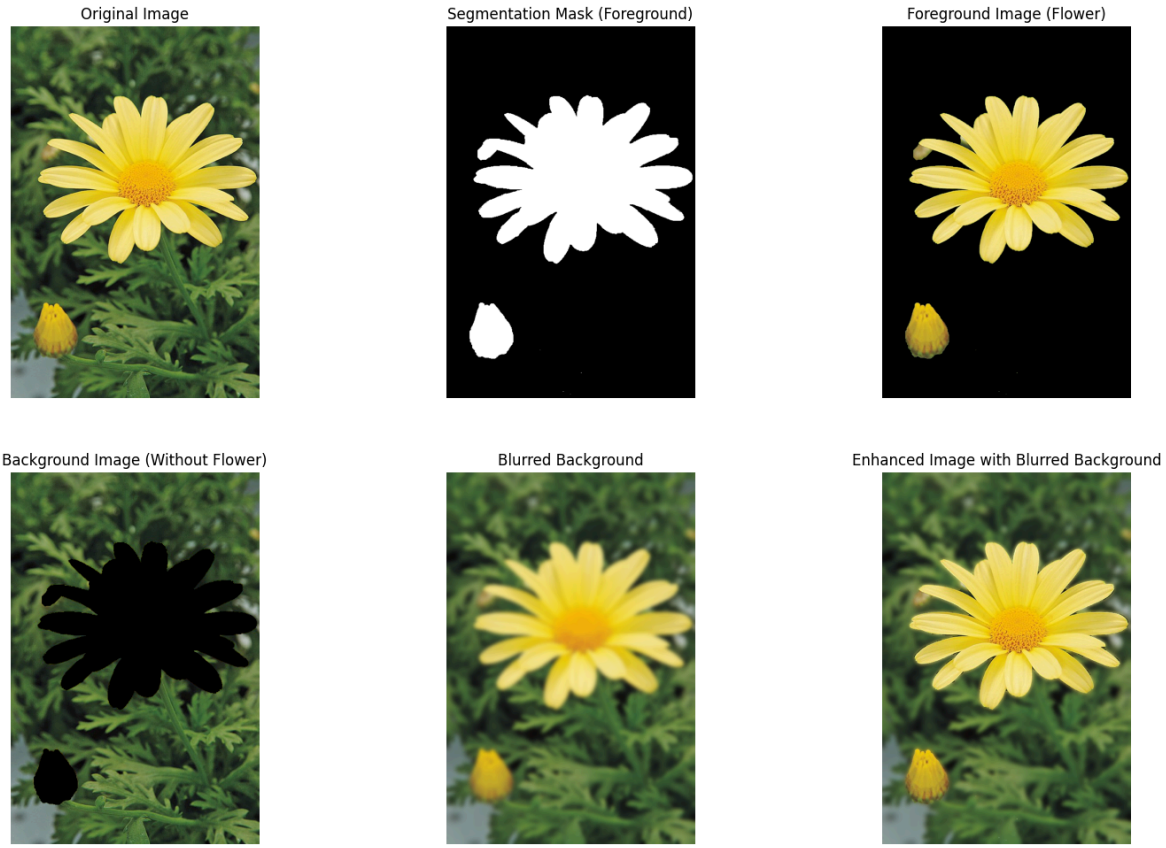

```
Normalized SSD (Nearest Neighbor): 26.446071759259258
Normalized SSD (Bilinear): 18.34591087962963
```

Segment the Image and Enhance it

In [76]:

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

# Load the image
image_path = r'C:\Users\samko\Downloads\a1images\daisy.jpg'
image = cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2RGB)  # Load as RGB

# (a) GrabCut Segmentation
mask = np.zeros(image.shape[:2], np.uint8)
bgd_model = np.zeros((1, 65), np.float64)
fgd_model = np.zeros((1, 65), np.float64)

rect = (50, 50, image.shape[1] - 50, image.shape[0] - 50)
cv2.grabCut(image, mask, rect, bgd_model, fgd_model, 5, cv2.GC_INIT_WITH_RECT)

# Vectorized mask creation
mask2 = np.where((mask == 1) | (mask == 3), 1, 0).astype('uint8')

# Vectorized foreground & background
foreground = image * mask2[:, :, np.newaxis]
background = image * (1 - mask2)[:, :, np.newaxis]
```

```python
# (b) Blurred background
blurred_background = cv2.GaussianBlur(image, (25, 25), 0)

# Vectorized enhanced image
enhanced_image = blurred_background.copy()
enhanced_image[mask2 == 1] = image[mask2 == 1]

# Display results
fig, ax = plt.subplots(2, 3, figsize=(18, 12))
titles = ["Original Image", "Segmentation Mask (Foreground)", "Foreground Image
          "Background Image (Without Flower)", "Blurred Background", "Enhanced I
images = [image, mask2, foreground, background, blurred_background, enhanced_ima

for i, (img, title) in enumerate(zip(images, titles)):
    ax[i // 3, i % 3].imshow(img if img.ndim == 3 else img, cmap=None if img.ndi
    ax[i // 3, i % 3].set_title(title)
    ax[i // 3, i % 3].axis("off")

plt.show()

# (c) Explanation
print("Explanation: The darker edges appear because the blur mixes the darker ba
```



Explanation: The darker edges appear because the blur mixes the darker background with lighter flower pixels near the border, creating a shadow-like effect.