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

# Step 1: Read the image
image = cv2.imread('/content/cvlab1img.jpg')  # Replace with your image filename

# Convert BGR to RGB (Matplotlib uses RGB format)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 2: Display the original image using Matplotlib
plt.figure(figsize=(8, 6))
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis("off")
plt.show()

# Step 3: Extract Image Size
height, width, channels = image.shape
print(f"Image Size: Width={width}, Height={height}, Channels={channels}")

# Step 4: Calculate Total Pixels
total_pixels = width * height
print(f"Total Pixels: {total_pixels}")

# Step 5: Convert RGB to Grayscale
gray_image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2GRAY)
cv2.imwrite('image_gray.jpg', gray_image)  # Save grayscale image

# Display Grayscale Image
plt.figure(figsize=(8, 6))
plt.imshow(gray_image, cmap='gray')
plt.title("Grayscale Image")
plt.axis("off")
plt.show()

# Step 6: Convert Grayscale to Binary using a threshold
threshold_value = 127
_, binary_image = cv2.threshold(gray_image, threshold_value, 255, cv2.THRESH_BINARY)
cv2.imwrite('image_binary.jpg', binary_image)  # Save binary image

# Display Binary Image
plt.figure(figsize=(8, 6))
plt.imshow(binary_image, cmap='gray')
plt.title("Binary Image")
plt.axis("off")
plt.show()

# Count Black Pixels (0 value pixels)
black_pixel_count = np.sum(binary_image == 0)
print(f"Black Pixel Count: {black_pixel_count}")
```

Original Image

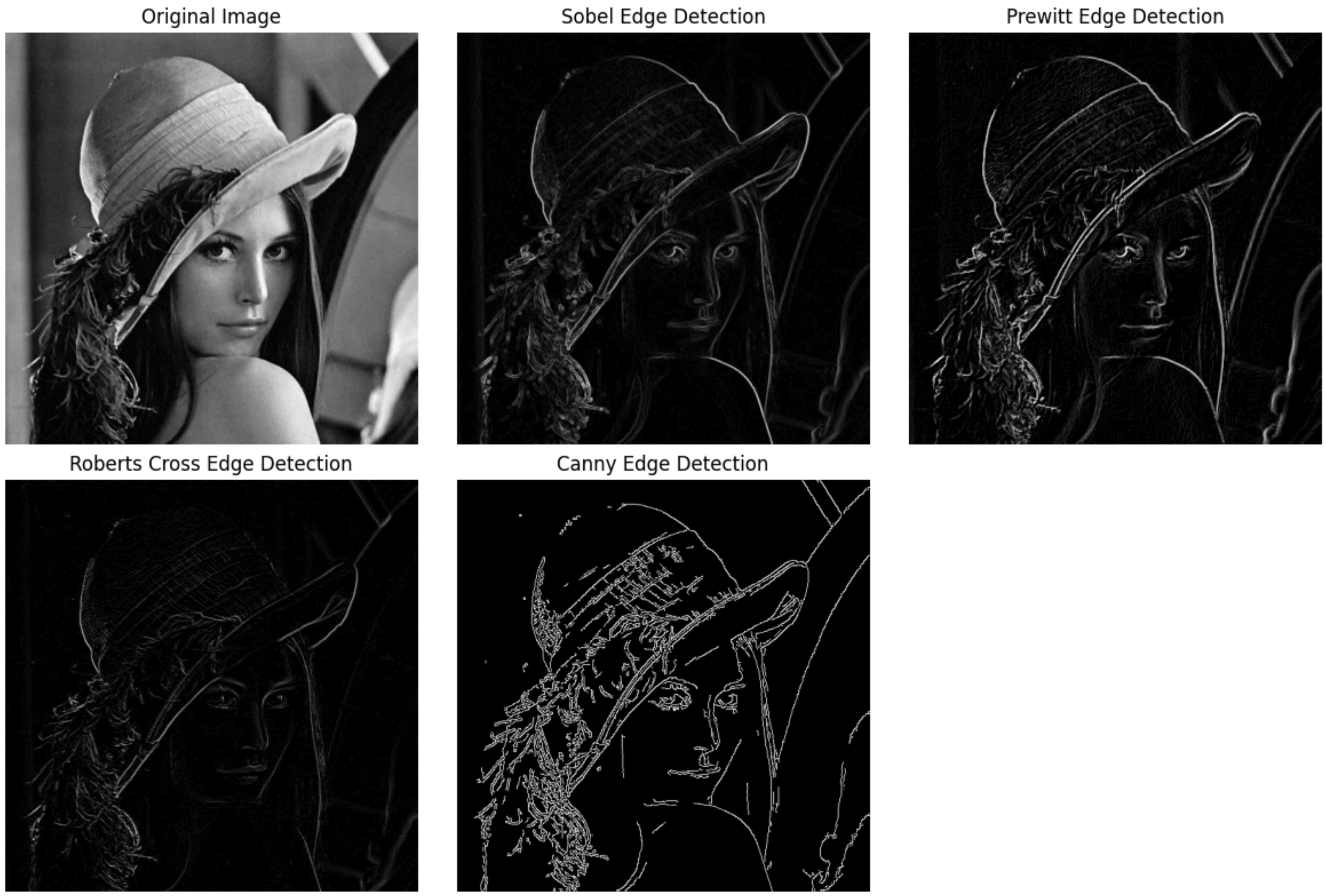
```
Image Size: Width=512, Height=512, Channels=3
Total Pixels: 262144
```

Grayscale Image

Binary Image

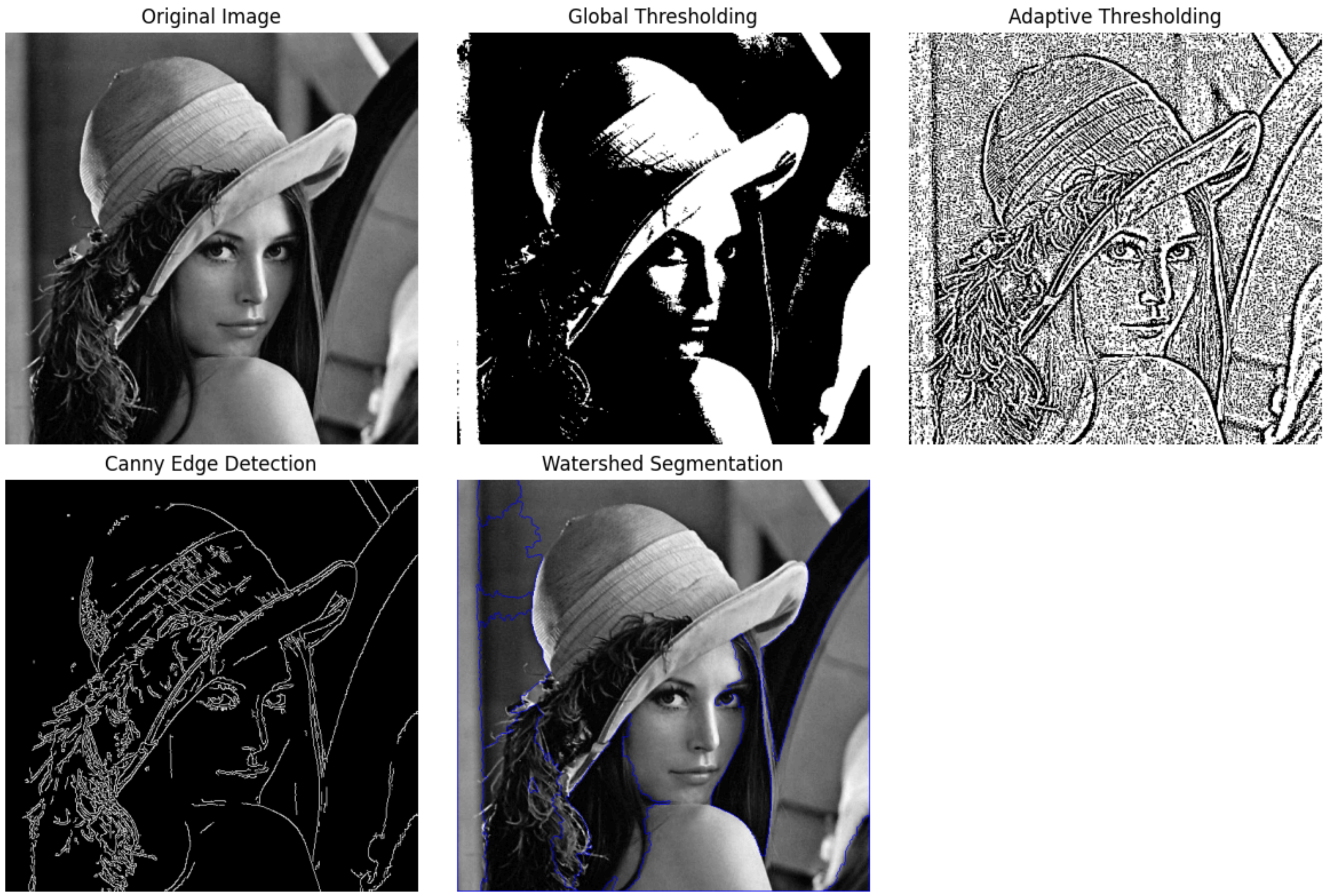
```
Black Pixel Count: 199287
```

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

# Load the image in grayscale
image = cv2.imread('/content/cvlab1img.jpg', cv2.IMREAD_GRAYSCALE)  # Replace with your image filename

# Apply Sobel Operator (X and Y gradients)
sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)  # Sobel in X direction
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)  # Sobel in Y direction
sobel_combined = cv2.magnitude(sobel_x, sobel_y)  # Compute gradient magnitude

# Apply Prewitt Operator
prewitt_x = cv2.filter2D(image, -1, np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]]))  # Prewitt X
prewitt_y = cv2.filter2D(image, -1, np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]]))  # Prewitt Y
prewitt_combined = cv2.magnitude(prewitt_x.astype(np.float32), prewitt_y.astype(np.float32))  # Gradient magnitude

# Apply Roberts Cross Operator
roberts_x = cv2.filter2D(image, -1, np.array([[1, 0], [0, -1]]))  # Roberts X
roberts_y = cv2.filter2D(image, -1, np.array([[0, 1], [-1, 0]]))  # Roberts Y
roberts_combined = cv2.magnitude(roberts_x.astype(np.float32), roberts_y.astype(np.float32))  # Gradient magnitude

# Apply Canny Edge Detection
canny_edges = cv2.Canny(image, 100, 200)  # Thresholds: 100 (lower) and 200 (upper)

# Display all edge-detected images using Matplotlib
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

axes[0, 0].imshow(image, cmap='gray')
axes[0, 0].set_title("Original Image")
axes[0, 0].axis("off")

axes[0, 1].imshow(sobel_combined, cmap='gray')
axes[0, 1].set_title("Sobel Edge Detection")
axes[0, 1].axis("off")

axes[0, 2].imshow(prewitt_combined, cmap='gray')
axes[0, 2].set_title("Prewitt Edge Detection")
axes[0, 2].axis("off")

axes[1, 0].imshow(roberts_combined, cmap='gray')
axes[1, 0].set_title("Roberts Cross Edge Detection")
axes[1, 0].axis("off")

axes[1, 1].imshow(canny_edges, cmap='gray')
axes[1, 1].set_title("Canny Edge Detection")
axes[1, 1].axis("off")

# Hide last empty subplot
axes[1, 2].axis("off")

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



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

# Load the image in grayscale
image_path = '/content/cvlab1img.jpg'  # Replace with your image path
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image is loaded properly
#if image is None:
 #   print(f"Error: Unable to load image at {image_path}. Check the file path and format.")
  #  exit()

### 1. Global Thresholding ###
ret, global_thresh = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

### 2. Adaptive Thresholding ###
adaptive_thresh = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                        cv2.THRESH_BINARY, 11, 2)

### 3. Edge Detection for Segmentation (Canny Edge) ###
edges = cv2.Canny(image, 100, 200)

### 4. Region-Based Segmentation (Watershed Algorithm) ###
# Convert to BGR for watershed algorithm
```

```
image_color = cv2.imread(image_path)

# Apply Otsu's Thresholding to create a binary image
ret, binary = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

# Noise removal using morphological operations
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel, iterations=2)

# Sure background area
sure_bg = cv2.dilate(opening, kernel, iterations=3)

# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
ret, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)

# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)

# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)

# Add 1 to all labels to ensure background is not 0
markers = markers + 1

# Mark the unknown region as 0
markers[unknown == 255] = 0

# Apply watershed algorithm
cv2.watershed(image_color, markers)
image_color[markers == -1] = [255, 0, 0]  # Mark boundaries in red

### Plot all results ###
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

axes[0, 0].imshow(image, cmap='gray')
axes[0, 0].set_title("Original Image")
axes[0, 0].axis("off")

axes[0, 1].imshow(global_thresh, cmap='gray')
axes[0, 1].set_title("Global Thresholding")
axes[0, 1].axis("off")

axes[0, 2].imshow(adaptive_thresh, cmap='gray')
axes[0, 2].set_title("Adaptive Thresholding")
axes[0, 2].axis("off")

axes[1, 0].imshow(edges, cmap='gray')
axes[1, 0].set_title("Canny Edge Detection")
axes[1, 0].axis("off")

axes[1, 1].imshow(cv2.cvtColor(image_color, cv2.COLOR_BGR2RGB))
axes[1, 1].set_title("Watershed Segmentation")
axes[1, 1].axis("off")

# Hide last empty subplot
axes[1, 2].axis("off")

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


Original Image
Global Thresholding
Adaptive Thresholding
Canny Edge Detection
Watershed Segmentation