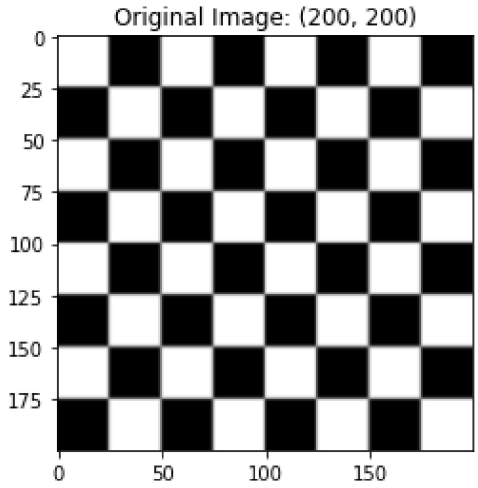
In [ ]:
```
import matplotlib.pyplot as plt
import cv2
import numpy as np
import skimage.data as data
```

6.1

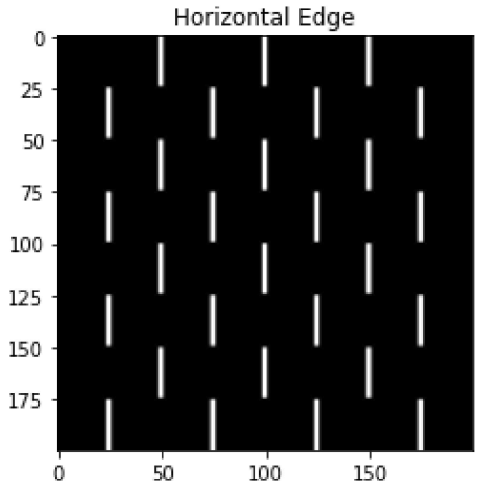
In [ ]:
```
img_org = data.checkerboard()
plt.imshow(img_org, cmap='gray')
plt.title(f'Original Image: {img_org.shape}')
```

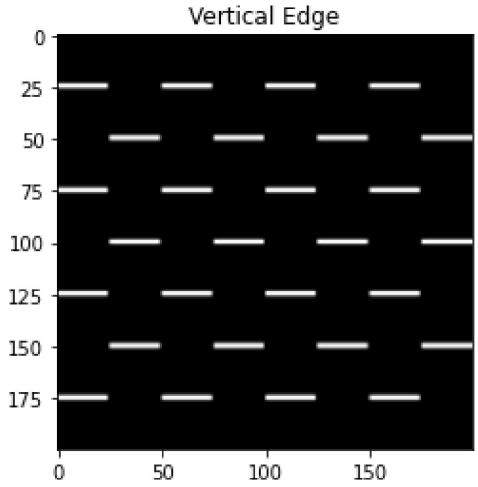
Out[ ]:
```
Text(0.5, 1.0, 'Original Image: (200, 200)')
```

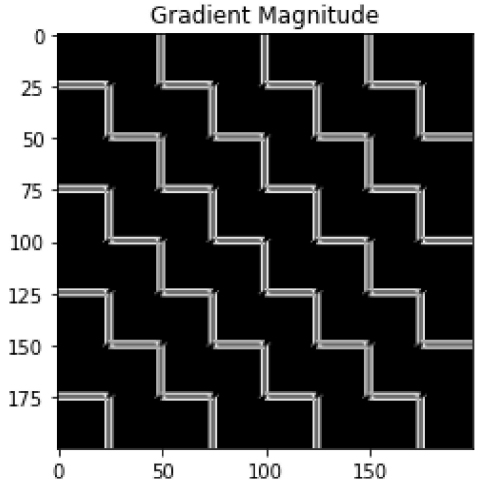


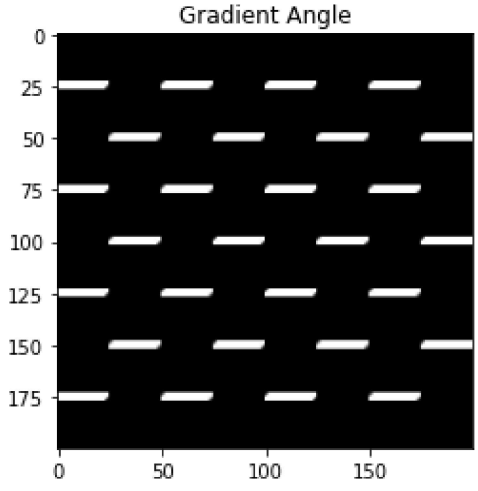
In [ ]:
```
def sobel_edge_detection(image, filter, verbose=False):
    G_dx = cv2.filter2D(image, -1, filter)
    if verbose:
        plt.imshow(G_dx, cmap='gray')
        plt.title("Horizontal Edge")
        plt.show()

    G_dy = cv2.filter2D(image,-1, np.flip(filter.T, axis=0))

    if verbose:
        plt.imshow(G_dy, cmap='gray')
        plt.title("Vertical Edge")
        plt.show()

    gradient_magnitude = np.sqrt(np.square(G_dx) + np.square(G_dy))

    gradient_magnitude *= 255.0 / gradient_magnitude.max()
    gradient_angle = np.arctan2(G_dy, G_dx)

    if verbose:
        plt.imshow(gradient_magnitude, cmap='gray')
        plt.title("Gradient Magnitude")
        plt.show()

        plt.imshow(gradient_angle, cmap='gray')
        plt.title("Gradient Angle")
        plt.show()

    return gradient_magnitude, gradient_angle



filter = np.array([[0, 0, 0],
                   [-1, 0, 1],
                   [0, 0, 0]
```

```
                ])

gradient_magnitude, gradient_angle = sobel_edge_detection(img_org, filter=filter, verbose=True)
```





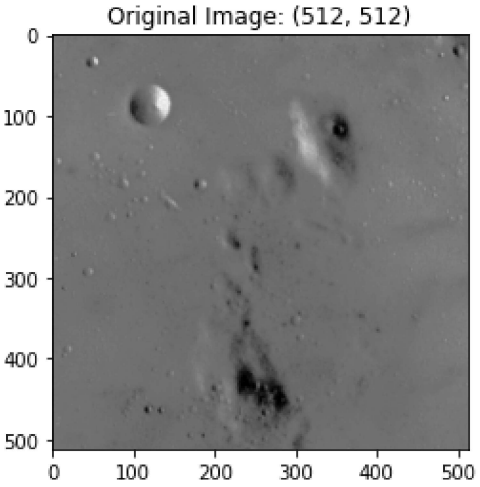




6.2

In [ ]:
```
img_org = data.moon()
plt.imshow(img_org, cmap='gray')
plt.title(f'Original Image: {img_org.shape}')
```

Out[ ]:
```
Text(0.5, 1.0, 'Original Image: (512, 512)')
```

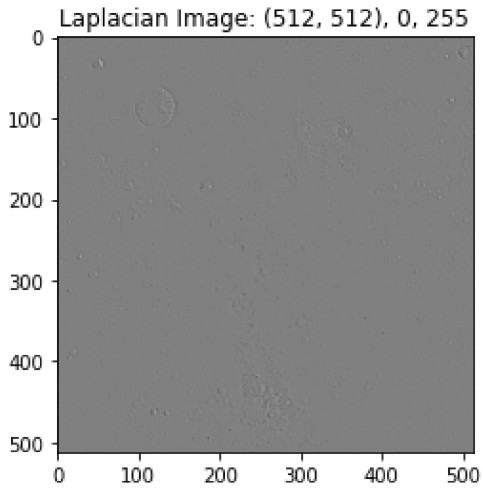


In [ ]:
```
def normalize(img):
    return (img - img.min())/(img.max() - img.min())
```

Laplacian

In [ ]:
```
def laplacian(img, diagonal=True, ddepth=cv2.CV_64F, default=False, Normalize=True):
    # https://aishack.in/tutorials/sobel-laplacian-edge-detectors/

    if diagonal:
        filter = np.array((
                           [-1, -1, -1],
                           [-1, 8, -1],
                           [-1, -1, -1]))
    else:
        filter = np.array((
                           [0, -1, 0],
                           [-1, 4, -1],
                           [0, -1, 0]))

    if default:
        lap =  cv2.Laplacian(img, ddepth=ddepth)
    else:
        lap = cv2.filter2D(img, ddepth=ddepth, kernel=filter)
    # Normalize
    if Normalize:
        #lap = cv2.convertScaleAbs(lap)
        lap = normalize(lap)
        lap = (lap * 255).astype(np.uint8)

    return lap


lap = laplacian(img_org)
print(lap.dtype)
plt.imshow(lap, cmap='gray')
plt.title(f'Laplacian Image: {lap.shape}, {lap.min()}, {lap.max()}')
```

Out[ ]:
```
uint8
Text(0.5, 1.0, 'Laplacian Image: (512, 512), 0, 255')
```



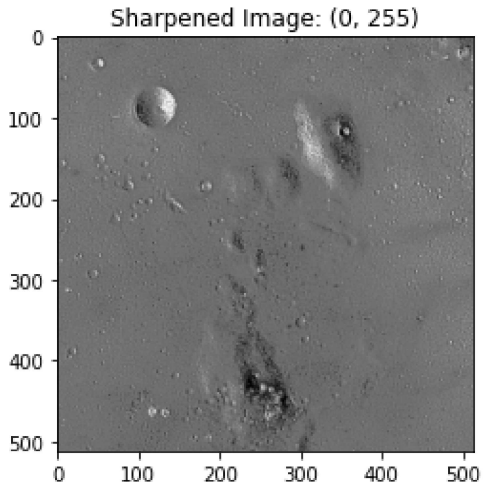
Sharpening

In [ ]:

```
def sharpen(img):
    lap_img = laplacian(img, Normalize=False)
    out = img + lap_img
    out = cv2.convertScaleAbs(out)
    out = out.astype(np.uint8)

    return out

sharpened_image = sharpen(img_org)

plt.imshow(sharpened_image, cmap='gray')
plt.title(f'Sharpened Image: {sharpened_image.min(), sharpened_image.max()}')
```

Out[ ]:

```
Text(0.5, 1.0, 'Sharpened Image: (0, 255)')
```

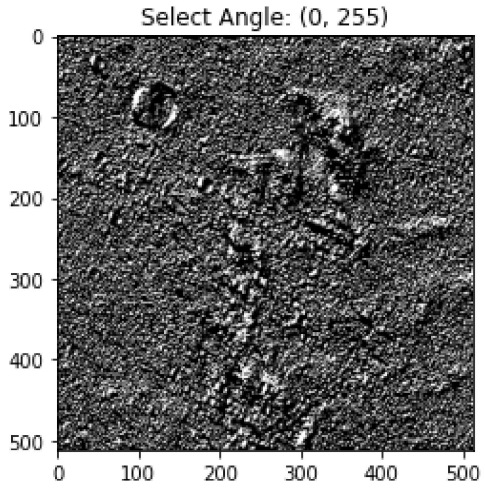


45 degree sharpening

In [ ]:

```
def select_polar_angle(angle_im, threshold):
    return np.where(np.abs(angle_im) < threshold, 0, angle_im)

filter = np.array([[0, 0, 0],
                   [-1, 0, 1],
                   [0, 0, 0]
                ])

gradient_magnitude, gradient_angle = sobel_edge_detection(img_org, filter=filter, verbose=False)

select_angle = select_polar_angle(gradient_angle, np.pi/4)
select_angle = (normalize(select_angle) * 255).astype(np.uint8)

sharpened_image = img_org + select_angle
```

```
plt.imshow(select_angle, cmap='gray')
plt.title(f'Select Angle: {select_angle.min(), select_angle.max()}')
plt.show()

plt.imshow(sharpened_image, cmap='gray')
plt.title(f'Sharpened Image: {sharpened_image.min(), sharpened_image.max()}')
```



Out[ ]: Text(0.5, 1.0, 'Sharpened Image: (0, 255)')