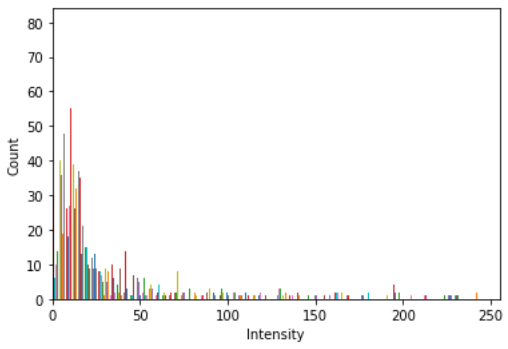
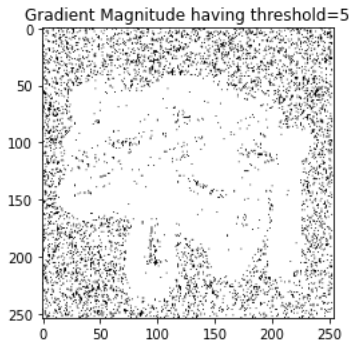
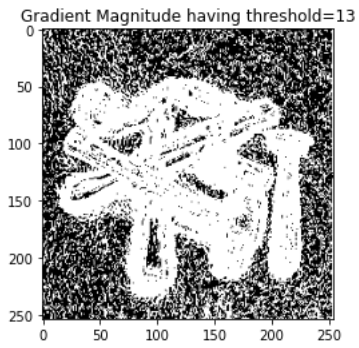
```
import numpy as np
from numpy.fft import fft2, ifft2, fftshift
from scipy.signal import convolve2d, correlate2d
import matplotlib.pyplot as plt

from Functions import *
from gaussfft import gaussfft
#import cv2
import time


tools = np.load("/content/few256.npy")
house = np.load("/content/godthem256.npy")


# Approximate the first order partial derivatives
# Sobel filter smoothens the image when convolved

delta_x = [[-1, 0, 1],  # Horizontal Edges
           [-2, 0, 2],
           [-1, 0, 1]]

delta_y = [[-1, -2, -1],   # Vertical Edges
           [0, 0, 0],
           [1, 2, 1]]

# Discrete Derivation Approximations
# 'Valid' -> Return convolution parts without zero paded. Hence,
# size of dxtools & dytools reduces by 2.
dxtools = convolve2d(tools, delta_x, 'valid')  # Convolution
dytools = convolve2d(tools, delta_y, 'valid')

print("Size of dxmask", np.shape(delta_x))  # (5,5)
print("Size of dymask", np.shape(delta_y))  # (5,5)
print("Size of Image", np.shape(tools))  # (256,256)
print("Size of dxtools", np.shape(dxtools))  # (254,254)
print("Size of dytools", np.shape(dytools))  # (254,254)

# plotting
plt.close("all")
plt.figure()

ax = plt.subplot(2, 3, 1)
ax.axis("off")
ax.imshow(tools, cmap='gray')
ax.set_title("image")

ax = plt.subplot(2, 3, 2)
ax.axis("off")
ax.imshow(dxtools, cmap='gray')
ax.set_title("Gradient along x")

ax = plt.subplot(2, 3, 3)
ax.axis("off")
ax.imshow(dytools, cmap='gray')
ax.set_title("Gradient along y")
plt.show()
```

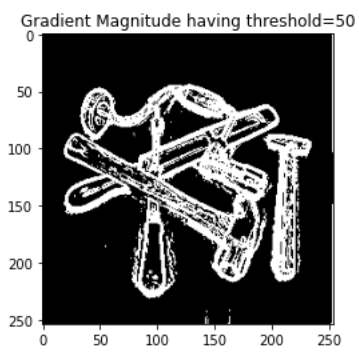
```
Size of dxmask (3, 3)
Size of dymask (3, 3)
Size of Image (256, 256)
Size of dxtools (254, 254)
Size of dytools (254, 254)
```

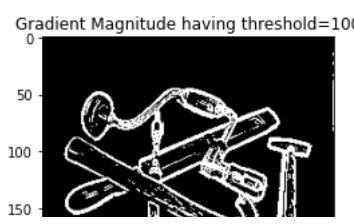

```
# 2. Point-wise thresholding of gradient magnitudes

# If magnitude at a pixel exceeds a threshold,report a possible edge point
gradmagntools = np.sqrt(dxtools**2+dytools**2)
plt.hist(gradmagntools,255)
plt.ylabel("Count")
plt.xlabel("Intensity")
plt.xlim([0,255])
plt.show()

thresholds=[5,13,50,100,150,200]
for i in thresholds: #range(1,len(thresholds)):
    plt.title('Gradient Magnitude having threshold=%i' %i)
    plt.imshow(((gradmagntools-i)>0).astype(int),cmap='gray')
    plt.show()
```



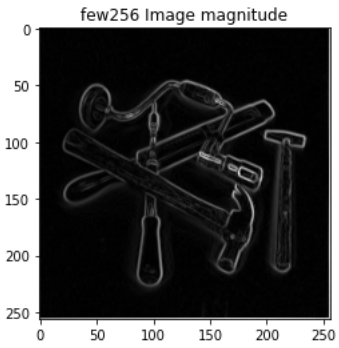


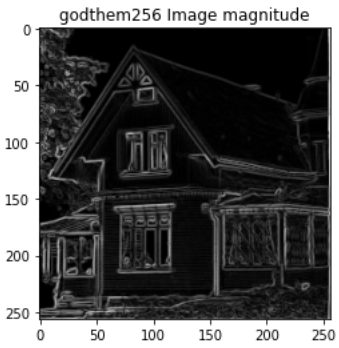






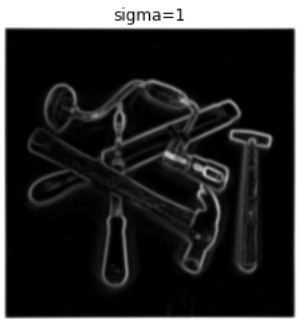

```
sigma_x = [[-1, 0, 1],
           [-2, 0, 2],
           [-1, 0, 1]]

sigma_y = [[-1, -2, -1],
           [0, 0, 0],
           [1, 2, 1]]

shape = 'same'


def Lv(inpic, shape):
    Lx = convolve2d(inpic, sigma_x, shape)
    Ly = convolve2d(inpic, sigma_y, shape)
    return np.sqrt(Lx**2+Ly**2)

resultt = Lv(tools, shape)
plt.imshow(resultt, cmap='gray')
plt.title("few256 Image magnitude")
plt.show()

resultt2 = Lv(house, shape)
plt.imshow(resultt2, cmap='gray')
plt.title("godthem256 Image magnitude")
plt.show()


plt.title(label="sigma=1")
showgrey((discgaussfft(resultt, 1)))  # Kernel variance=0.1
plt.show()

plt.title(label="sigma=16")
showgrey((discgaussfft(resultt, 16)))  # Kernel variance=16
plt.show()

plt.title(label="sigma=1")
showgrey((discgaussfft(resultt2, 1)))  # Kernel variance=0.1
plt.show()

plt.title(label="sigma=16")
showgrey((discgaussfft(resultt2, 16)))  # Kernel variance=16
plt.show()
```





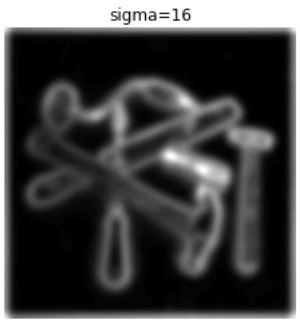

```
(256, 256) (256, 256) (256, 256)
```

sigma=1



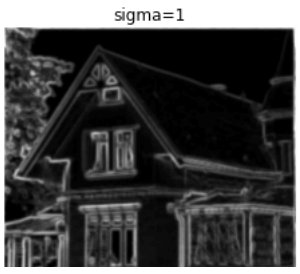
```
(256, 256) (256, 256) (256, 256)
```

sigma=16



```
(256, 256) (256, 256) (256, 256)
```

sigma=1



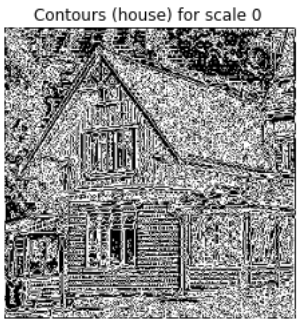
4. Computing differential geometry descriptors

```
# Extracting thin edges by considering points for which gradient magnitude
# reaches local maxima in gradient direction.

# Lvv= 2nd order derivative of smoothened intensity function L in v direction.


sigmaa_x = [[0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0],
            [0, -0.5, 0, 0.5, 0],
            [0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0]]

sigmaa_y = [[0, 0, 0, 0, 0],
            [0, 0, -0.5, 0, 0],
            [0, 0, 0, 0, 0],
            [0, 0, 0.5, 0, 0],
            [0, 0, 0, 0, 0]]

sigmaa_xx = [[0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0],
             [0, 1, -2, 1, 0],
             [0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0]]

sigmaa_yy = [[0, 0, 0, 0, 0],
             [0, 0, 1, 0, 0],
             [0, 0, -2, 0, 0],
             [0, 0, 1, 0, 0],
             [0, 0, 0, 0, 0]]

shape = 'same'
sigmaa_xy = convolve2d(sigmaa_x, sigmaa_y, shape)
sigmaa_xxx = convolve2d(sigmaa_x, sigmaa_xx, shape)
sigmaa_yyy = convolve2d(sigmaa_y, sigmaa_yy, shape)
sigmaa_xxy = convolve2d(sigmaa_x, sigmaa_xy, shape)
sigmaa_xyy = convolve2d(sigmaa_x, sigmaa_yy, shape)

def Lvvtilde(pic, shape):
    Lx = convolve2d(pic, sigmaa_x, shape)
    Ly = convolve2d(pic, sigmaa_y, shape)
    Lxx = convolve2d(pic, sigmaa_xx, shape)
    Lxy = convolve2d(pic, sigmaa_xy, shape)
    Lyy = convolve2d(pic, sigmaa_yy, shape)

    first_term = (Lx**2)*(Lxx)
    second_term = 2*(Lx)*(Ly)*(Lxy)
    third_term = (Ly**2)*Lyy
    Lvv = first_term+second_term+third_term
    #print("Lvv Sign:")
    #print(np.sign(Lvv))
```

```
        return Lvv

    def Lvvvtilde(pic, shape='same'):
        Lx = convolve2d(pic, sigmaa_x, shape)
        Ly = convolve2d(pic, sigmaa_y, shape)
        Lxx = convolve2d(pic, sigmaa_xx, shape)
        Lxy = convolve2d(pic, sigmaa_xy, shape)
        Lyy = convolve2d(pic, sigmaa_yy, shape)

        Lxxx = convolve2d(pic, sigmaa_xxx, shape)
        Lyyy = convolve2d(pic, sigmaa_yyy, shape)
        Lxxy = convolve2d(pic, sigmaa_xxy, shape)
        Lxyy = convolve2d(pic, sigmaa_xyy, shape)

        Lvvv = ((Lx**3)*Lxxx) + 3*(Lx**2)*Ly*Lxxy + 3*Lx*(Ly**2)*Lxyy + (Ly**3)*Lyyy
        #print("Lvvv Sign:")
        #print(np.sign(Lvvv))
        return Lvvv

    scale = [0.0001, 1.0, 4.0, 16.0, 64.0]
    for i in scale:
        plt.title("Contours (house) for scale %i " % i)
        # Computing zero-crossings of Lvv on scale
        showgrey(contour(Lvvtilde(discgaussfft(house, i), shape)))

    for i in scale:
        plt.title("3rd order derivative (few256) for scale %i " % i)
        res=discgaussfft(tools, i)
        showgrey((Lvvvtilde(res, shape)<0).astype(int))


    [x, y] = np.meshgrid(range(-5, 6), range(-5, 6))
    #print(convolve2d(x**3, sigmaa_xxx, 'valid'))
    #print(convolve2d(x**3, sigmaa_xx, 'valid'))
    #print(convolve2d(x**2*y, sigmaa_xxy, 'valid'))
```

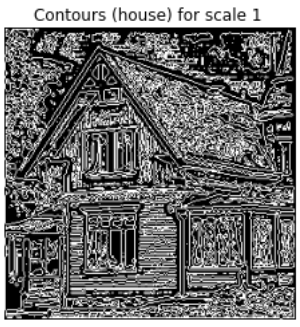
```
(256, 256) (256, 256) (256, 256)
```

Contours (house) for scale 0



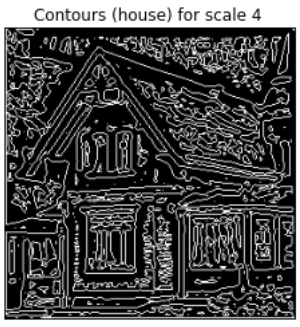
```
(256, 256) (256, 256) (256, 256)
```

Contours (house) for scale 1



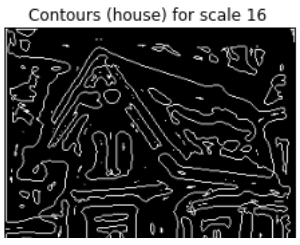
```
(256, 256) (256, 256) (256, 256)
```

Contours (house) for scale 4



```
(256, 256) (256, 256) (256, 256)
```

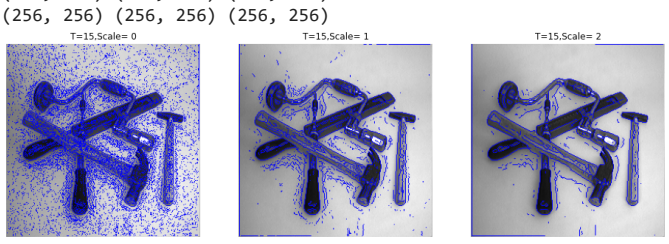
Contours (house) for scale 16

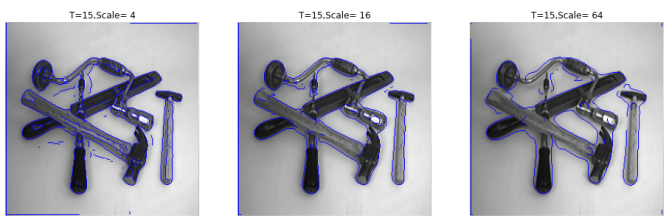


```
### 5. Extraction of edge segments

def extractedge(inpic, scale, threshold, shape):
    zeropic = Lvvtilde(discgaussfft(inpic, scale), shape)
    Lvvvtil = Lvvvtilde(discgaussfft(inpic, scale), shape)
    Lv1 = Lv(discgaussfft(inpic, scale), shape)

    maskpic1 = np.array((Lvvvtil < 0))
    maskpic2 = np.array(Lv1 > threshold)
    # Extract level curves in image & rejects points based on sign of 2nd input arguement
    curves = zerocrosscurves(zeropic, maskpic1)
    # Thresholds these curves wrt sign of another image
    curves = thresholdcurves(curves, maskpic2)
    return curves


# Threshold for the gradient magnitude
# Find best scale for each image & adjest threshold accordingly
scale = [0.0001,1, 2, 4, 16, 64]
#threshold = 0.7
thresholds=[1,5,10,15,17,20]

fig = plt.figure(figsize=(16, 16))
for i in range(len(thresholds)):
    edgecurves = extractedge(tools,scale[4],thresholds[i], shape)
    ax=fig.add_subplot(2,3,i+1)
    ax.plot()
    plt.title("Scale=16, threshold %i" % thresholds[i])
    overlaycurves(tools, edgecurves)
plt.show()

fig = plt.figure(figsize=(16, 16))
for i in range(len(scale)):
    edgecurves = extractedge(tools,scale[i], thresholds[3], shape)
    ax=fig.add_subplot(2,3,i+1)
    ax.plot()
    plt.title("T=15,Scale= %i" % scale[i])
    overlaycurves(tools, edgecurves)
plt.show()

fig = plt.figure(figsize=(16, 16))
for i in range(len(scale)):
    edgecurves = extractedge(house,scale[i], thresholds[3], shape)
    ax=fig.add_subplot(2,3,i+1)
    ax.plot()
    plt.title("T=15,Scale= %i" % scale[i])
    overlaycurves(house, edgecurves)
plt.show()

fig = plt.figure(figsize=(16, 16))
for i in range(len(thresholds)):
    edgecurves = extractedge(house,scale[4],thresholds[i], shape)
    ax=fig.add_subplot(2,3,i+1)
    ax.plot()
    plt.title("Scale=16, threshold %i" % thresholds[i])
    overlaycurves(house, edgecurves)
plt.show()
```

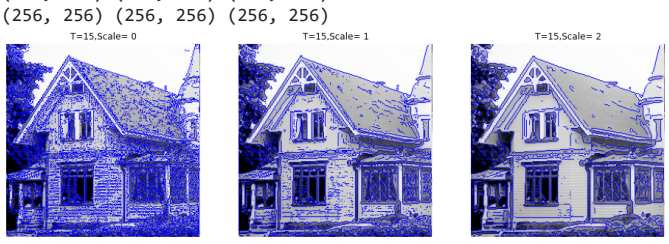
```
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
```


```
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
```






```
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
```

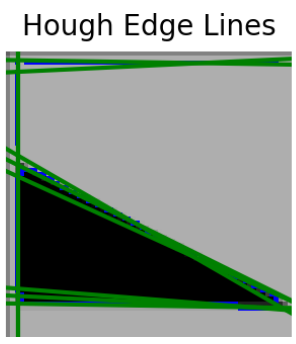



```
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
```

```
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
(256, 256) (256, 256) (256, 256)
```


```
# 6. Hough Transform

def houghline(pic, curves, magnitude, nrho, ntheta, threshold, nlines=10, verbose=False, scale = 0):
    accumulator = np.zeros((nrho, ntheta))
    x, y = magnitude.shape
    r = np.sqrt(np.square(x) + np.square(y))
    rho = np.linspace(-r, r, nrho)
    theta = np.linspace(-np.pi/2, np.pi/2, ntheta)

    for i in range(len(curves[0])):
        x = curves[0][i]
        y = curves[1][i]
        curve_magnitude = magnitude[x][y]
        if curve_magnitude > threshold:
            for j in range(ntheta):
                rho_Val = (x*np.cos(theta[j])) + (y*np.sin(theta[j]))
                rho_Index = np.argmin(abs(rho - rho_Val))
                accumulator[rho_Index][j] += 1

    linepar = []

    # Detecting local maxima in accumulator by locmax8()
    pos, value, _ = locmax8(accumulator)
    indexvector = np.argsort(value)[-nlines:]  # Index
    pos = pos[indexvector]

    f = plt.figure(figsize=(4, 4), dpi=200)
    f.subplots_adjust(wspace=0.2, hspace=0.4)
```

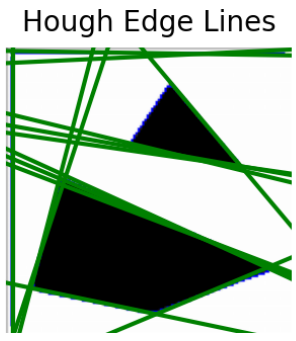
```
        xx = f.add_subplot(1, 2, 1)
        showgrey(accumulator, False)
        xx.title.set_text("Accumulator")

        xx = f.add_subplot(1, 2, 2)
        overlaycurves(pic, curves)
        xx.title.set_text("Hough Edge Lines")

        for idx in range(nlines):  # Index values correspond to nlines
            thetaidxacc = pos[idx][0]
            rhoidxacc = pos[idx][1]

            rhoMax = rho[rhoidxacc]
            thetaMax = theta[thetaidxacc]
            linepar.append([rhoMax, thetaMax])

            # Convert (rho,theta) back to cartesian coordinates (x,y)
            x0 = rhoMax * np.cos(thetaMax)
            y0 = rhoMax * np.sin(thetaMax)
            dx = r * (-np.sin(thetaMax))
            dy = r * (np.cos(thetaMax))

            plt.plot([y0 - dy, y0, y0 + dy],[x0 - dx, x0, x0 + dx],  "g-")

        plt.show()
        return linepar, accumulator


    def houghedgeline(pic, scale, thresholdd, nrho, ntheta, nlines=10, verbose=False):
        curves = extractedge(pic, scale, thresholdd, "same")
        gaussianSmooth = discgaussfft(pic, scale)  # Smoothen the histogram before detecting local maxima
        gradmagn = Lv(gaussianSmooth, "same")

        linepar, acc = houghline(pic,curves,gradmagn,nrho,ntheta,thresholdd,nlines,verbose,scale)
        return None


    start_time = time.time()
    testimage1=np.load("/content/triangle128.npy")
    smalltest1=binsubsample(testimage1)
    #scale = [0.0001,1, 2, 4, 16, 64]
    #threshold = 0.7
    #thresholds=[1,5,10,15,17,20]
    scale=4
    thresholdd= 5 # Lowest value allowed for given magnitude
    edgecurves = extractedge(smalltest1,scale, thresholdd, shape)
    ntheta = 100 # Number of Accumulator in theta direction
    nrho = 200  # Number of Accumulators in rho direction

    houghedgeline(smalltest1, scale, thresholdd, nrho, ntheta, nlines=10)

    # For large rho/theta values, lines will be of low accuracy
    # For samll rho/theta values, result may contain multiple responses for same line.

    print("--- %s seconds ---" % (time.time() - start_time))

         [[0.0625 0.125  0.0625]
          [0.125  0.25   0.125 ]
          [0.0625 0.125  0.0625]]
        (64, 64) (64, 64) (64, 64)
        (64, 64) (64, 64) (64, 64)
        (64, 64) (64, 64) (64, 64)
        (64, 64) (64, 64) (64, 64)
        (64, 64) (64, 64) (64, 64)
        (64, 64) (64, 64) (64, 64)
        (64, 64) (64, 64) (64, 64)
```




```
--- 0.5854420661926270 seconds ---
```

```
start_time = time.time()

testimage1=np.load("/content/houghtest256.npy")
smalltest1=binsubsample(testimage1)
#scale = [0.0001,1, 2, 4, 16, 64]
#threshold = 0.7
#thresholds=[1,5,10,15,17,20]
scale=4
thresholdd= 5 # Lowest value allowed for given magnitude
edgecurves = extractedge(smalltest1,scale, thresholdd, shape)
ntheta = 100 # Number of Accumulator in theta direction
nrho = 200  # Number of Accumulators in rho direction

houghedgeline(smalltest1, scale, thresholdd, nrho, ntheta, nlines=15)

# For large rho/theta values, lines will be of low accuracy
# For samll rho/theta values, result may contain multiple responses for same line.

print("--- %s seconds ---" % (time.time() - start_time))
```

```
[[0.0625 0.125  0.0625]
 [0.125  0.25   0.125 ]
 [0.0625 0.125  0.0625]]
(128, 128) (128, 128) (128, 128)
(128, 128) (128, 128) (128, 128)
(128, 128) (128, 128) (128, 128)
(128, 128) (128, 128) (128, 128)
(128, 128) (128, 128) (128, 128)
(128, 128) (128, 128) (128, 128)
(128, 128) (128, 128) (128, 128)
```




```
--- 1.261528491973877 seconds ---
```

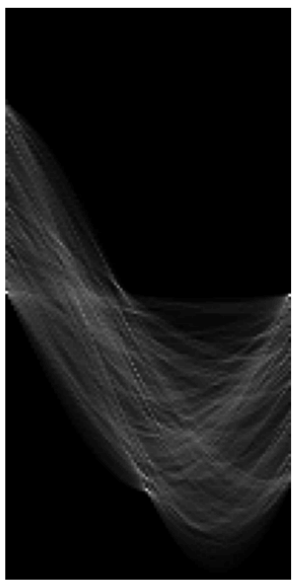
```
start_time = time.time()

testimage1=np.load("/content/few256.npy")
smalltest1=binsubsample(testimage1)
#scale = [0.0001,1, 2, 4, 16, 64]
#threshold = 0.7
#thresholds=[1,5,10,15,17,20]
scale=4
thresholdd= 5 # Lowest value allowed for given magnitude
edgecurves = extractedge(smalltest1,scale, thresholdd, shape)
ntheta = 100 # Number of Accumulator in theta direction
nrho = 200  # Number of Accumulators in rho direction

houghedgeline(smalltest1, scale, thresholdd, nrho, ntheta, nlines=15)

# For large rho/theta values, lines will be of low accuracy
# For samll rho/theta values, result may contain multiple responses for same line.

print("--- %s seconds ---" % (time.time() - start_time))
```

```
      [[0.0625 0.125  0.0625]
       [0.125  0.25   0.125 ]
       [0.0625 0.125  0.0625]]
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
```

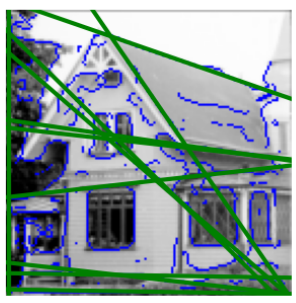
```
start_time = time.time()

testimage1=np.load("/content/godthem256.npy")
smalltest1=binsubsample(testimage1)
#scale = [0.0001,1, 2, 4, 16, 64]
#threshold = 0.7
#thresholds=[1,5,10,15,17,20]
scale=10
thresholdd= 1 # Lowest value allowed for given magnitude
edgecurves = extractedge(smalltest1,scale, thresholdd, shape)
ntheta = 100 # Number of Accumulator in theta direction
nrho = 200  # Number of Accumulators in rho direction

houghedgeline(smalltest1, scale, thresholdd, nrho, ntheta, nlines=12)

# For large rho/theta values, lines will be of low accuracy
# For samll rho/theta values, result may contain multiple responses for same line.

print("--- %s seconds ---" % (time.time() - start_time))
```

```
      [[0.0625 0.125  0.0625]
       [0.125  0.25   0.125 ]
       [0.0625 0.125  0.0625]]
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
```

Accumulator



Hough Edge Lines



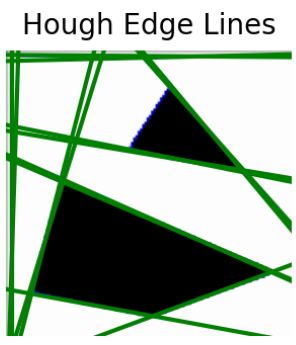
```
--- 3.19510817527771 seconds ---
```

***Q.9 How do the results and computational time depend on the number of cells in the accumulator? ***

```
start_time = time.time()

testimage1=np.load("/content/houghtest256.npy")
smalltest1=binsubsample(testimage1)

scale=4
thresholdd= 5 # Lowest value allowed for given magnitude
edgecurves = extractedge(smalltest1,scale, thresholdd, shape)

ntheta = 200 # Number of Accumulator in theta direction
nrho = 200  # Number of Accumulators in rho direction

houghedgeline(smalltest1, scale, thresholdd, nrho, ntheta, nlines=15)
print("ntheta:",ntheta," ","nrho:",nrho)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
start_time = time.time()

testimage1=np.load("/content/houghtest256.npy")
smalltest1=binsubsample(testimage1)

scale=4
thresholdd= 5 # Lowest value allowed for given magnitude
edgecurves = extractedge(smalltest1,scale, thresholdd, shape)

ntheta = 200 # Number of Accumulator in theta direction
nrho = 400  # Number of Accumulators in rho direction

houghedgeline(smalltest1, scale, thresholdd, nrho, ntheta, nlines=15)
print("ntheta:",ntheta," ","nrho:",nrho)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
     [[0.0625 0.125  0.0625]
      [0.125  0.25   0.125 ]
      [0.0625 0.125  0.0625]]
     (128, 128) (128, 128) (128, 128)
     (128, 128) (128, 128) (128, 128)
     (128, 128) (128, 128) (128, 128)
     (128, 128) (128, 128) (128, 128)
     (128, 128) (128, 128) (128, 128)
     (128, 128) (128, 128) (128, 128)
     (128, 128) (128, 128) (128, 128)
```



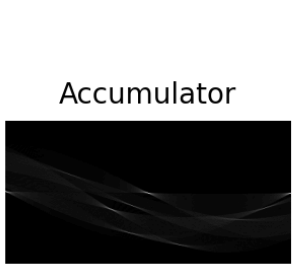

```
ntheta: 200   nrho: 400
--- 2.206779241561889 seconds ---
```

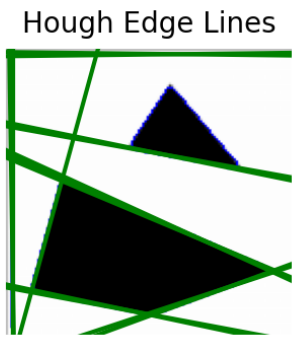
```
start_time = time.time()

testimage1=np.load("/content/houghtest256.npy")
smalltest1=binsubsample(testimage1)

scale=4
thresholdd= 5 # Lowest value allowed for given magnitude
edgecurves = extractedge(smalltest1,scale, thresholdd, shape)

ntheta = 200 # Number of Accumulator in theta direction
nrho = 800  # Number of Accumulators in rho direction

houghedgeline(smalltest1, scale, thresholdd, nrho, ntheta, nlines=15)
print("ntheta:",ntheta," ","nrho:",nrho)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
      [[0.0625 0.125  0.0625]
       [0.125  0.25   0.125 ]
       [0.0625 0.125  0.0625]]
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
```

```
start_time = time.time()

testimage1=np.load("/content/houghtest256.npy")
smalltest1=binsubsample(testimage1)

scale=4
thresholdd= 5 # Lowest value allowed for given magnitude
edgecurves = extractedge(smalltest1,scale, thresholdd, shape)

ntheta = 400 # Number of Accumulator in theta direction
nrho = 200  # Number of Accumulators in rho direction

houghedgeline(smalltest1, scale, thresholdd, nrho, ntheta, nlines=15)
print("ntheta:",ntheta," ","nrho:",nrho)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
      [[0.0625 0.125  0.0625]
       [0.125  0.25   0.125 ]
       [0.0625 0.125  0.0625]]
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
```

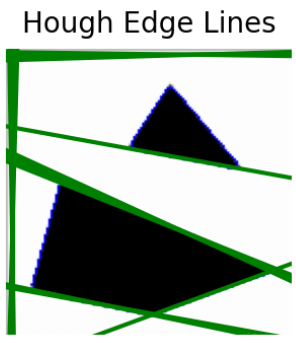





```
      ntheta: 400   nrho: 200
```

```
start_time = time.time()

testimage1=np.load("/content/houghtest256.npy")
smalltest1=binsubsample(testimage1)

scale=4
thresholdd= 5 # Lowest value allowed for given magnitude
edgecurves = extractedge(smalltest1,scale, thresholdd, shape)

ntheta = 800 # Number of Accumulator in theta direction
nrho = 200  # Number of Accumulators in rho direction

houghedgeline(smalltest1, scale, thresholdd, nrho, ntheta, nlines=15)
print("ntheta:",ntheta," ","nrho:",nrho)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
      [[0.0625 0.125  0.0625]
       [0.125  0.25   0.125 ]
       [0.0625 0.125  0.0625]]
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
      (128, 128) (128, 128) (128, 128)
```




```
ntheta: 800   nrho: 200
```