In [1]:

```python
import pickle as pkl
import numpy as np
import cv2
from PIL import Image as im
import math
```

Step 1 : Gaussian smoothing

In [2]:

```python
def loadMask(path,lable): # This function is used to Load Masks from .pkl files
    with open(path, 'rb') as f:
        mask = pkl.load(f)
        print(lable+" :\n",mask)
        return mask
```

In [3]:

```python
def load_img(path): # This function Loads image and converts 3 channel image to single channel grayscale image
    img=im.open('Images/Test_patterns.bmp')
    img=img.convert('L')
    img=np.asarray(img)
    return img
```

In [4]:

```python
def display_Img(img,lable="Img"): # function to Display an Image
    img=np.uint8(img)
    cv2.imshow(lable,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    #data = im.fromarray(img) # to save Image
    #data.save(lable+'.bmp')
```

In [5]:

```python
#This function takes Image, Gaussian Mask and returns Gaussian smoothed image
def apply_Gaussian_mask(oldImg,mask,normalize_factor=1):
    ( h , w ) = oldImg.shape
    normalize_factor=np.sum(mask) # Normalize factor is sum of the vaues of the Gaussian mask
    newImg = np.zeros( ( h , w ), dtype = np.int32 )
    P = ( mask[0].size//2 )
    for j in range( P , h - P ):
        for i in range( P , w - P ):
            value = 0
            for k in range( -P , P+1 ):
                for l in range( -P , P+1 ):
                    value += mask[k+P][l+P] * oldImg[j+k][i+l]
            newImg[j][i] = value//normalize_factor
            # here normalization by dividing pixel value with Normalize factor is performed simultaneously.
    return ( newImg , P )
```

In [6]:

```python
def Gaussian_smoothing(path):
    Gaussian_mask = loadMask('Pickle/Gaussian_mask.pkl','Gaussian Mask') # Loading Gaussian Mask from .pkl file
    Img = load_img(path) # Loading Image
    ( Img , pedding ) = apply_Gaussian_mask( Img , Gaussian_mask ) # Smoothing Image using Gaussian filter
    return (Img , pedding)
```

In [7]:

```python
( Image , pedding ) = Gaussian_smoothing('Images/House.bmp')
display_Img(Image,"Gaussian smoothed") # Displaying smoothed image
```

```
Gaussian Mask :
 [[ 1  1  2  2  2  1  1]
 [ 1  2  2  4  2  2  1]
 [ 2  2  4  8  4  2  2]
 [ 2  4  8 16  8  4  2]
 [ 2  2  4  8  4  2  2]
 [ 1  2  2  4  2  2  1]
 [ 1  1  2  2  2  1  1]]
```

Step 2: Gradient Operation

In [8]:

```python
def apply_mask(oldImg,mask,pedding=0): # This function is used to apply operators like prewitt,Robert on an Image
    ( h , w ) = oldImg.shape
    newImg = np.zeros( ( h , w ), dtype = np.int32 )
    P = ( mask[0].size//2 )
    for j in range( (P + pedding) , h - (P + pedding) ):
        for i in range( (P + pedding) , w - (P + pedding) ):
            value = 0
            for k in range( -P , P+1 ):
                for l in range( -P , P+1 ):
                    value += mask[k+P][l+P] * oldImg[j+k][i+l]
            newImg[j][i] = value
    return ( newImg , P + pedding )
```

In [9]:

```python
def Normalization(img): # In Normalization, pixel values are rescale to 0-255 range
    img=np.absolute(img)
```

```
    img=img/img.max()*255
    return img
```

In [10]:

```
def gradient_magnitude(gx,gy): # Function to calculate gradient magnitude from horizontal and vertical gradient
    ( h , w ) = gx.shape
    grad_mag = np.zeros( ( h , w ), dtype = np.uint32 )
    for j in range(h):
        for i in range(w):
            grad_mag[j][i]= abs(gx[j][i])+abs(gy[j][i])
    return grad_mag
```

In [11]:

```
def gradient_angle(gx,gy): # Function to calculate gradient angle from horizontal and vertical gradient
    ( h , w ) = gx.shape
    grad_ang = np.zeros( ( h , w ), dtype = np.float32 )
    for j in range(h):
        for i in range(w):
            if gx[j][i] != 0:
                grad_ang[j][i]= math.degrees( math.atan(gy[j][i]/gx[j][i]) )
    return grad_ang
```

In [12]:

```
def sector(grad_ang,pedding=0): # Function to calculate sector value from gradient angle
    ( h , w ) = grad_ang.shape
    sec= np.zeros( ( h , w ), dtype = np.uint8 )
    for j in range( (1 + pedding) , h - (1 + pedding) ):
        for i in range( (1 + pedding) , w - (1 + pedding) ):
            Ang = grad_ang[j][i]
            if ( (Ang >= 0 and Ang < 22.5) or (Ang >= 337.5 and Ang <= 360) or (Ang >= 157.5 and Ang < 202.5) ):
                sec[j][i]=0
            elif ((Ang >= 22.5 and Ang < 67.5) or (Ang >= 202.5 and Ang < 247.5)):
                sec[j][i]=1
            elif ((Ang >= 67.5 and Ang < 112.5) or (Ang >= 247.5 and Ang < 292.5)):
                sec[j][i]=2
            elif ((Ang >= 112.5 and Ang < 157.5) or (Ang >= 292.5 and Ang < 337.5)):
                sec[j][i]=3
    return sec
```

In [13]:

```
def Gradient_Operation( Image , pedding ):
    Prewitt_X = loadMask('Pickle/prewitt-x.pkl','Prewitt X derivative') # Loading Prewitt X derivative from .pkl file
    Prewitt_Y = loadMask('Pickle/prewitt-y.pkl','Prewitt Y derivative') # Loading Prewitt Y derivative from .pkl file

    (Gx , _ ) = apply_mask(Image,Prewitt_X,pedding) # Calculating horizontal gradient
    Gx=Normalization(Gx)  # Normalizing horizontal gradient image

    (Gy , pedding ) = apply_mask(Image,Prewitt_Y,pedding) #Calculating vertical gradient
    Gy=Normalization(Gy)  # Normalizing vertical gradient image

    Grad_Mag=gradient_magnitude(Gx,Gy) # Calculating gradient magnitude
    Grad_Mag=Normalization(Grad_Mag) # Normalizing gradient magnitude image

    Grad_Ang = gradient_angle(Gx,Gy) # Calculating gradient angle

    Sector=sector(Grad_Ang,pedding) # Calculating sector value from gradient angle

    return ( Gx , Gy , Grad_Mag , Grad_Ang , Sector , pedding )
```

In [14]:

```
( Gx , Gy , Grad_Mag , Grad_Ang , Sector , pedding ) = Gradient_Operation( Image , pedding )
display_Img(Gx,"Gx") # displaying horizontal gradient
display_Img(Gy,"Gy") # displaying vertical gradient
display_Img(Grad_Mag,"Gradient Mangitude") # displaying gradient magnitude
```

```
Prewitt X derivative :
 [[-1  0  1]
 [-1  0  1]
 [-1  0  1]]
Prewitt Y derivative :
 [[ 1  1  1]
 [ 0  0  0]
 [-1 -1 -1]]
```

Step 3: Non-maxima Suppression

In [15]:

```
def NMS(grad_mag,sec,pedding=0): # Applying NMS on Gradient Mangitude with the help of sector value
    ( h , w ) = grad_mag.shape
    nms_img = np.zeros( ( h , w ), dtype = np.uint8 )
    for j in range( (1 + pedding) , h - (1 + pedding) ):
        for i in range( (1 + pedding) , w - (1 + pedding) ):
            if( sec[j][i] == 0 ):
                m=0
                n=1
            elif( sec[j][i] == 1 ):
                m=-1
                n=1
            elif( sec[j][i] == 2 ):
                m=1
                n=0
            elif( sec[j][i] == 3 ):
```

```
                m=1
                n=1
            if(grad_mag[j][i]>grad_mag[j-m][i-n] and grad_mag[j][i]>grad_mag[j+m][i+n]):
                nms_img[j][i] = grad_mag[j][i]
    return nms_img
```

In [16]:

```
Img_NMS=NMS(Grad_Mag,Sector,pedding)
display_Img(Img_NMS,"NMS") # displaying NMS image
```

Step 4: Thresholding

In [17]:

```
# Applying binary thresolding by calculating pixel value for nth percentile as a threshold value
def thresholding(img,percentile):
    ( h , w ) = img.shape
    Output = np.zeros( ( h , w ), dtype = np.uint8 )
    A = np.empty(np.count_nonzero(img)) # counting none zero values
    k=0
    for j in range(h):
        for i in range(w):
            if img[j][i]>0:
                A[k] = img[j][i]
                k=k+1
    t=np.percentile(A, percentile)
    for j in range(h):
        for i in range(w):
            if img[j][i]>t:
                Output[j][i]=255
    return Output
```

In [18]:

```
Img_25 = thresholding(Img_NMS,25) #25th percentile value as threshold
display_Img(Img_25,"25th percentile")
```

In [19]:

```
Img_50 = thresholding(Img_NMS,50) #50th percentile value as threshold
display_Img(Img_50,"50th percentile")
```

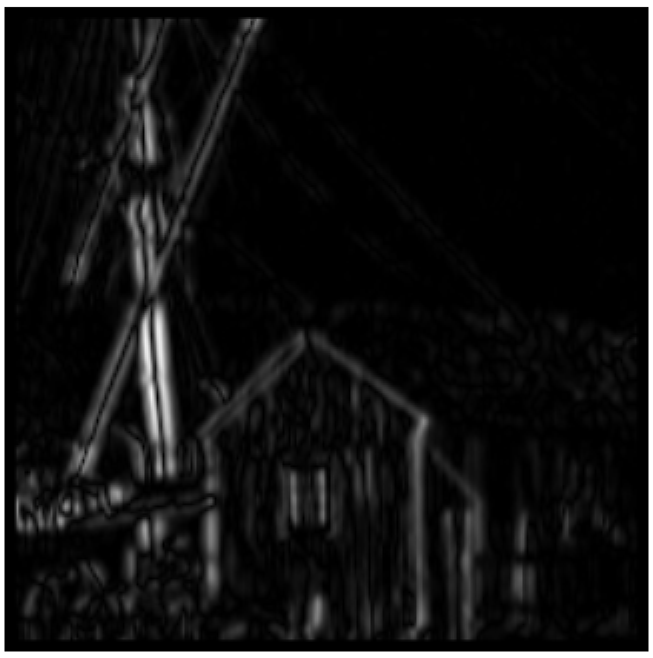
In [20]:

```
Img_75 = thresholding(Img_NMS,75) #75th percentile value as threshold
display_Img(Img_75,"75th percentile")
```

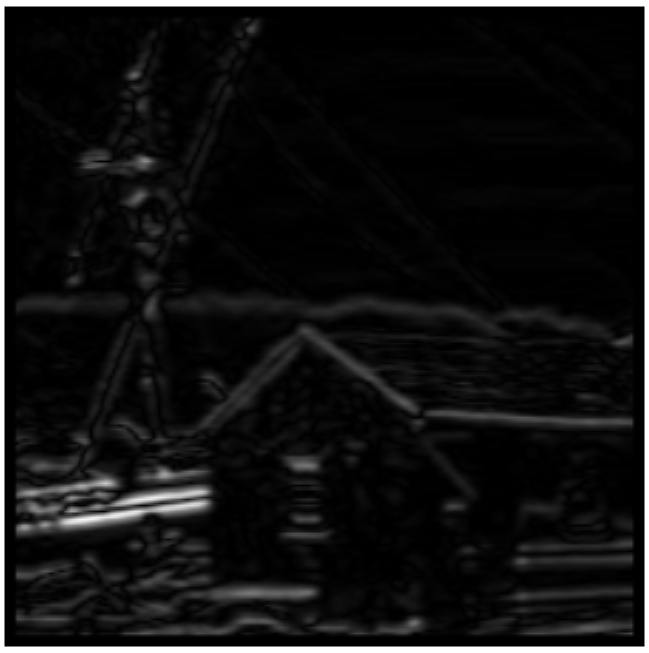
Input Image : House.bmp

1. Gaussian Smoothing

2. Horizontal Gradient Magnitude (Gx)



3. Vertical Gradient Magnitude (Gy)



4. Gradient Magnitude

5. Non-Maxima Suppression

6. Thresholding at 25th percentile

7. Thresholding at 50th percentile

8. Thresholding at 75th percentile

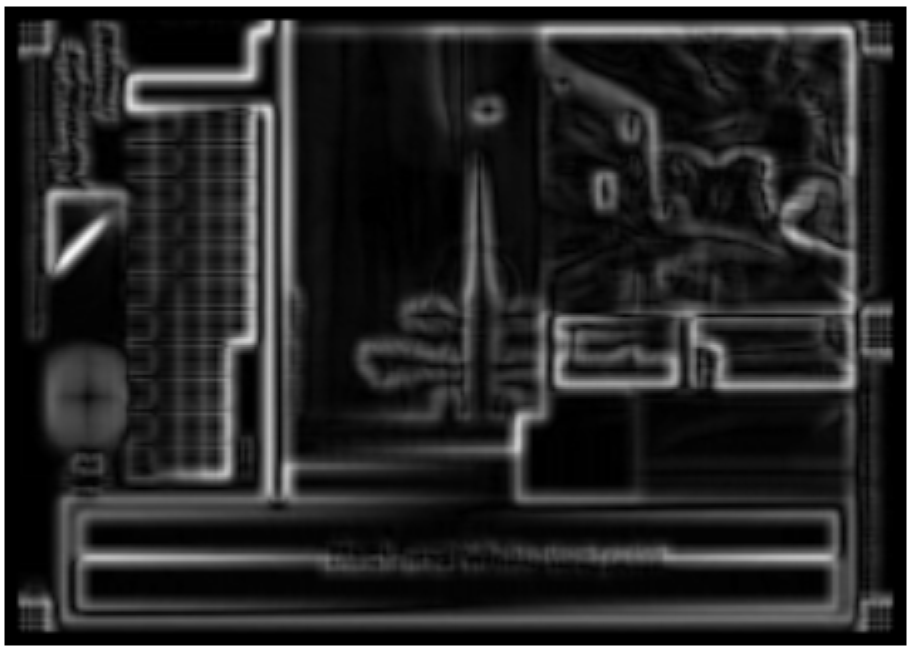
Input Image : Test patterns

1. Gaussian Smoothing

2. Horizontal Gradient Magnitude (Gx)

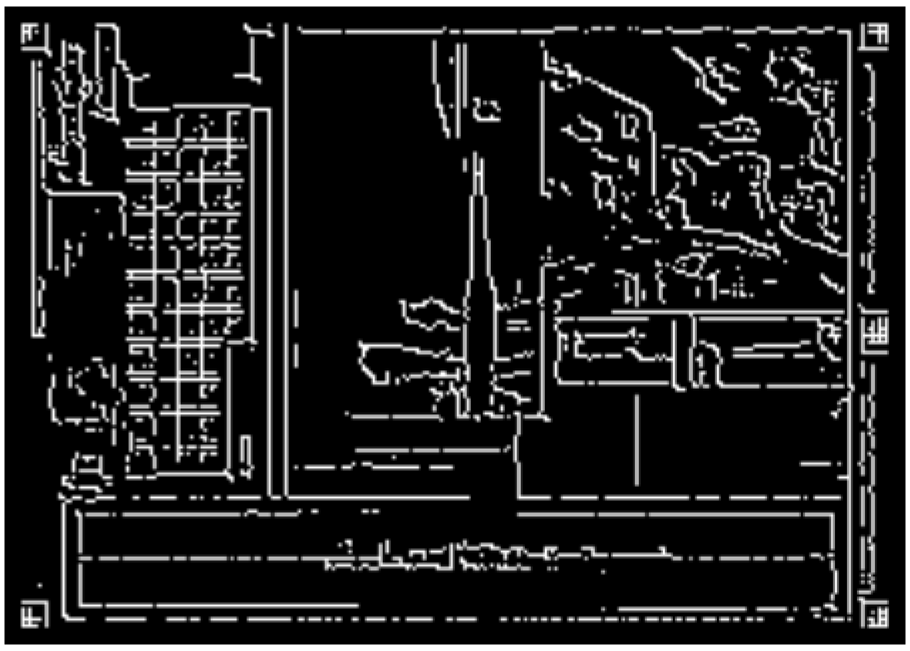
3. Vertical Gradient Magnitude (Gy)

4. Gradient Magnitude

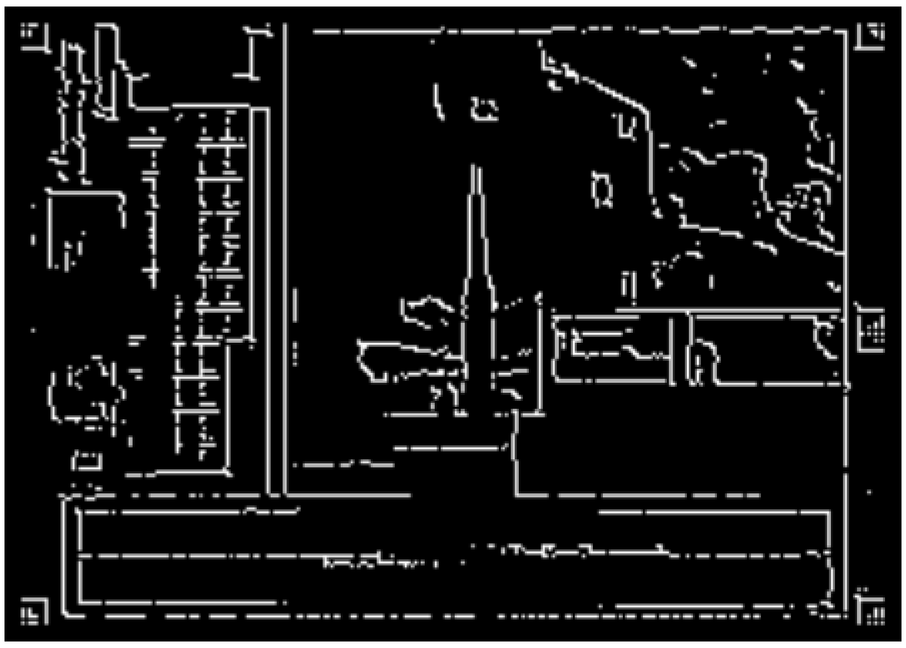


5. Non-Maxima Suppression

6. Thresholding at 25th percentile



7. Thresholding at 50th percentile



8. Thresholding at 75th percentile