

# 实验目的

掌握基本边缘检测算法

## 实验内容

1. Roberts算子边缘检测
2. Sobel算子边缘检测
3. Prewitt算子边缘检测
4. 拉普拉斯算子边缘检测（由于拉普拉斯算子检测过暗，本实验换为高斯-拉普拉斯算子）

## 算法设计

1. Roberts算子边缘检测

-1	0	0	-1
0	1	1	0

Roberts

### 算法分析：

Roberts算子的模板分为水平方向和垂直方向，如上图所示，从其模板可以看出，Roberts算子能较好的增强正负45度的图像边缘。根据上图可以得出下面算法：

图中某点经过罗伯特交叉梯度算子边缘检测后，获得响应主要依赖于对角上写的像素值  
 $f(\text{右下角}) - f(\text{左上角})$  或者是  $f(\text{左下角}) - f(\text{右上角})$ ，即左边图对负45°为0响应，右边图对正45°为0响应，两图合并实现对正负45°边缘的检测。

2. Prewitt算子边缘检测

-1	-1	-1	-1	0	1
0	0	0	-1	0	1
1	1	1	-1	0	1

Prewitt

### 算法分析：

Prewitt算子边缘检测利用两个卷积核（一个用于水平方向，一个用于垂直方向）分别对图像进行卷积，然后通过求两个方向的梯度幅值和方向来检测边缘。

Prewitt算子对边缘的响应比较敏感，但也容易受到噪声的影响。

Prewitt算子的卷积核对边缘的响应较为敏感，是因为卷积核中的权重值对边缘的梯度有较强的反应，也正是因为这个，卷积核中的权重也对图像中的噪声敏感，当图像受到噪声的影响时，Prewitt算子可能会放大噪声，导致在边缘以外的区域产生较强的响应。

### 3. Sobel算子边缘检测

-1	-2	-1	-1	0	1
0	0	0	-2	0	2
1	2	1	-1	0	1

Sobel

#### 算法分析：

Sobel算子类似于Prewitt算子，也是通过两个卷积核（一个用于水平方向，一个用于垂直方向）来计算梯度。

但是Sobel算子在距离中心像素点较近的四个点位（上下左右）采取了较其他像素点双倍权重的方法，使得靠近中心像素点的点占比更大，能有效降低较远处的噪声的影响。也就是在一定程度上能够让图片降噪，使得结果更加精确真实。

### 4. 拉普拉斯算子边缘检测以及高斯拉普拉斯算子边缘检测

拉普拉斯算子：

0	1	0
1	-4	1
0	1	0

#### 算法分析：

对于拉普拉斯算子边缘检测，如果结果  $< 0$ ，说明滤波中心是局部极大值，可能是噪声点，也可能是边缘上的点；结果  $= 0$ ，说明滤波中心很可能处于平坦区域；结果  $> 0$ ，说明滤波中心是局部极小值，可能是噪声点，也可能是边缘。

即拉普拉斯算子对含有噪声的图像极容易检测出假边缘，从而对噪点做出高响应。

于是引入高斯拉普拉斯算子边缘检测

高斯拉普拉斯算子：

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

#### 算法分析：

先对有噪声的图像高斯平滑处理，然后再使用拉普拉斯算子进行边缘检测，也可以将两个步骤集成在如上图所示的一个卷积核中处理。。

在图像中，平坦的区域的拉普拉斯响应为0，在边缘处的响应值较为剧烈，可以检测边缘的存在。

## 代码

```

from PIL import Image
from math import *

def read_image(file_path):
    # 读取图片
    image = Image.open(file_path)
    return image

def save_image(image, file_path):
    # 保存图片
    image.save(file_path)

def display_image(image):
    # 显示图片
    image.show()

def roberts_operator(image):
    # Roberts算子边缘检测
    width, height = image.size
    result_pixels = []

    for y in range(height - 1):
        for x in range(width - 1):
            pixel1 = image.getpixel((x, y))
            pixel2 = image.getpixel((x + 1, y + 1))
            pixel3 = image.getpixel((x + 1, y))
            pixel4 = image.getpixel((x, y + 1))

            # 边缘处理
            if(x+1>=width):
                pixel2 = 0
                pixel3 = 0
            if(y+1>=height):
                pixel2 = 0
                pixel4 = 0

            # 使用Roberts算子进行边缘检测
            gradient1 = abs(pixel1 - pixel2)
            gradient2 = abs(pixel3 - pixel4)
            gradient = sqrt(pow(gradient1, 2)+pow(gradient2, 2))

            result_pixels.append(gradient)

    result_image = Image.new("L", (width - 1, height - 1))
    result_image.putdata(result_pixels)
    return result_image

def sobel_operator(image):
    # Sobel算子边缘检测
    width, height = image.size
    result_pixels = []

    for y in range(1, height - 1):
        for x in range(1, width - 1):
            # Sobel算子权重
            weights1 = [

```

```

        [-1,  0,  1],
        [-2,  0,  2],
        [-1,  0,  1]
    ]
    weights2 = [
        [-1, -2, -1],
        [0,  0,  0],
        [1,  2,  1]
    ]
}

# 计算Sobel算子的梯度
gradient1 = sum(sum(weights1[i][j] * image.getpixel((x + j - 1, y +
i - 1)) for j in range(3)) for i in range(3))
gradient2 = sum(sum(weights2[i][j] * image.getpixel((x + j - 1, y +
i - 1)) for j in range(3)) for i in range(3))
gradient = sqrt(pow(gradient1, 2)+pow(gradient2, 2))

result_pixels.append(abs(gradient))

result_image = Image.new("L", (width - 2, height - 2))
result_image.putdata(result_pixels)
return result_image

def prewitt_operator(image):
    # Prewitt算子边缘检测
    width, height = image.size
    result_pixels = []

    for y in range(1, height - 1):
        for x in range(1, width - 1):
            # Prewitt算子权重
            weights1 = [
                [-1,  0,  1],
                [-1,  0,  1],
                [-1,  0,  1]
            ]
            weights2 = [
                [-1, -1, -1],
                [0,  0,  0],
                [1,  1,  1]
            ]

            # 计算Prewitt算子的梯度
            gradient1 = sum(sum(weights1[i][j] * image.getpixel((x + j - 1, y +
i - 1)) for j in range(3)) for i in range(3))
            gradient2 = sum(sum(weights2[i][j] * image.getpixel((x + j - 1, y +
i - 1)) for j in range(3)) for i in range(3))
            gradient = sqrt(pow(gradient1, 2)+pow(gradient2, 2))
            result_pixels.append(abs(gradient))

    result_image = Image.new("L", (width - 2, height - 2))
    result_image.putdata(result_pixels)
    return result_image

```

```

def laplacian_operator(image):
    # 拉普拉斯算子边缘检测
    width, height = image.size
    result_pixels = []

    for y in range(1, height - 1):
        for x in range(1, width - 1):
            # 拉普拉斯算子权重
            weights = [
                [0, 1, 0],
                [1, -4, 1],
                [0, 1, 0]
            ]
            weights2 = [
                [0, 0, -1, 0, 0],
                [0, -1, -2, -1, 0],
                [-1, -2, 17, -2, -1],
                [0, -1, -2, -1, 0],
                [0, 0, -1, 0, 0]
            ]
            ]

            # 计算拉普拉斯算子的梯度
            # gradient = sum(sum(weights[i][j] * image.getpixel((x + j - 1, y + i - 1)) for j in range(3)) for i in range(3))
            # 高斯拉普拉斯算子
            gradient = 0
            for i in range(5):
                temp = 0
                for j in range(5):
                    if(x+j-2>=width or x+j-2<0 or y+i-2>=height or y+i-2<0):
                        pixel1 = 0
                    else:
                        pixel1 = image.getpixel((x + j - 2, y + i - 2))
                        temp += weights2[i][j] * pixel1
                gradient +=temp

            # gradient = sum(sum(weights2[i][j] * image.getpixel((x + j - 2, y + i - 2)) for j in range(5)) for i in range(5))

            result_pixels.append(abs(gradient))

    result_image = Image.new("L", (width - 2, height - 2))
    result_image.putdata(result_pixels)
    return result_image


def main():
    dir_path = "./img3/"
    # 读取图片
    input_image = read_image(dir_path+"img3.tif")
    display_image(input_image)
    kernel_size = 3

```

```
# Roberts算子边缘检测
roberts_result = roberts_operator(input_image)
save_image(roberts_result, dir_path+"robert.bmp")
display_image(roberts_result)

# Sobel算子边缘检测
sobel_result = sobel_operator(input_image)
save_image(sobel_result, dir_path+"sobel.bmp")
display_image(sobel_result)

# Prewitt算子边缘检测
prewitt_result = prewitt_operator(input_image)
save_image(prewitt_result, dir_path+"prewitt.bmp")
display_image(prewitt_result)

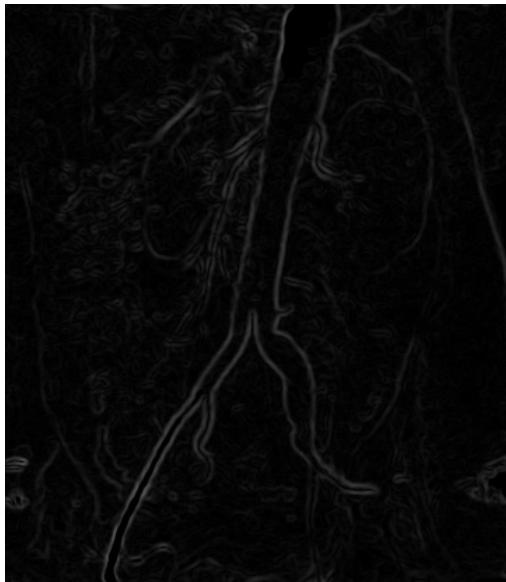
# 拉普拉斯算子边缘检测
midvalue_result = midValue(inpunt_image,kernel_size)
display_image(midValue_result)
laplacian_result = laplacian_operator(input_image)
display_image(laplacian_result)
save_image(laplacian_result, dir_path+"laplacian_gaosj_result.bmp")

if __name__ == "__main__":
    main()
```

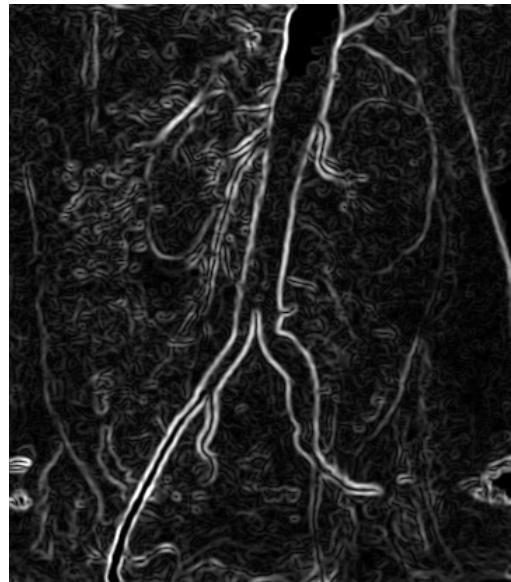
## 实验结果

---

对于图一：



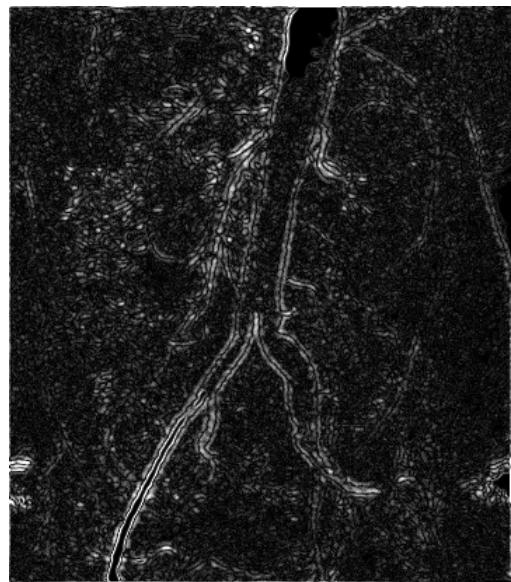
*Image 1: robert\_result*



*Image 2: prewitt\_result*

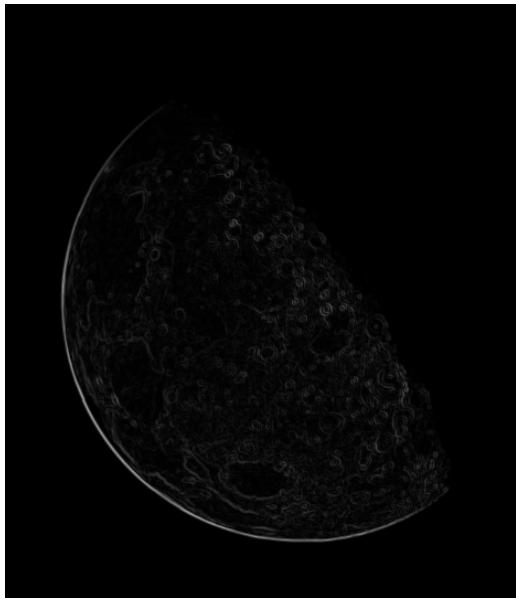


*Image 3: sobel\_result*

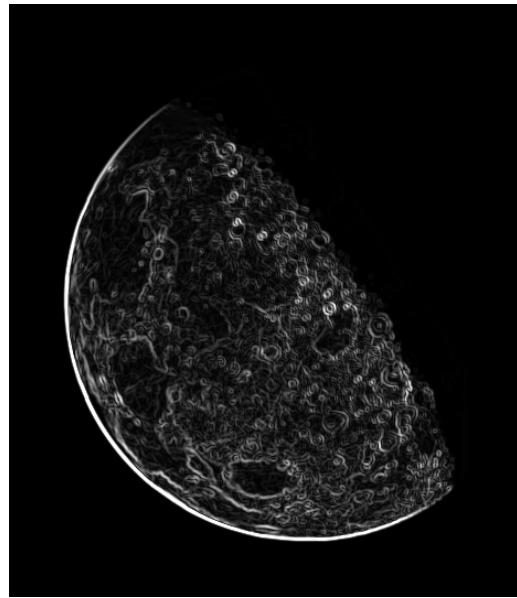


*Image 4:LoG\_result*

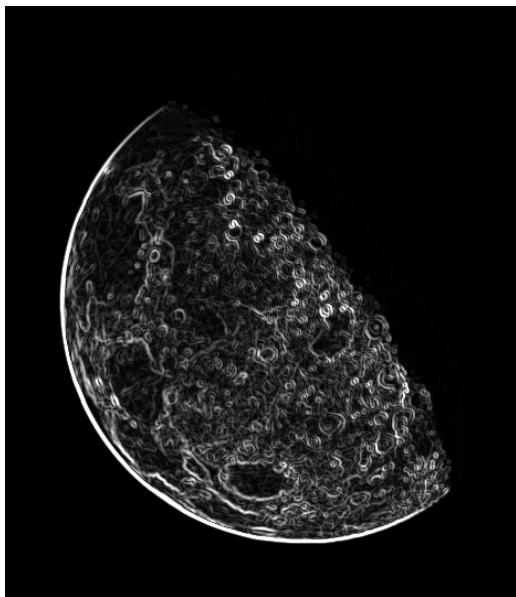
对于图二：



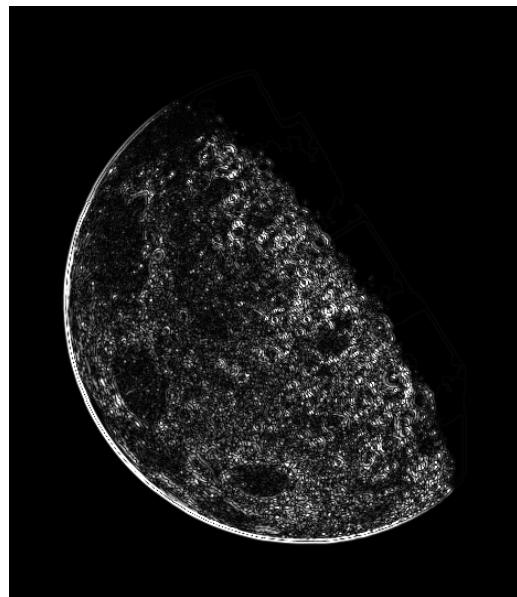
*Image 5: robert\_result*



*Image 6: prewitt\_result*

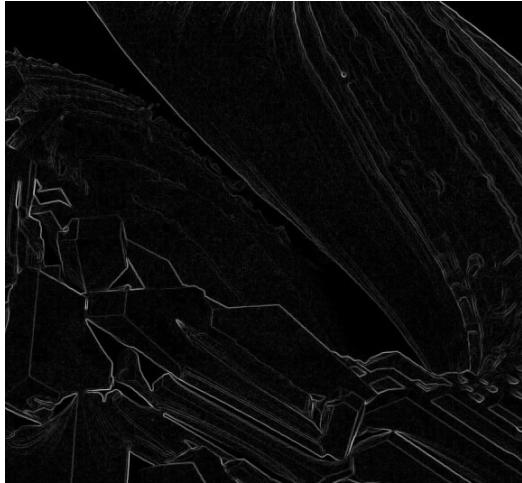


*Image 7: sobel\_result*



*Image 8:LoG\_result*

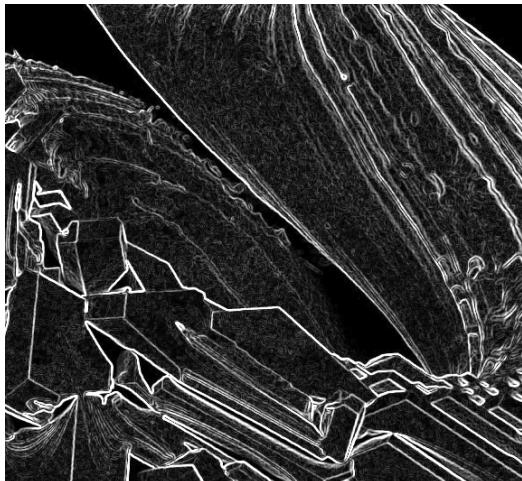
对于图三：



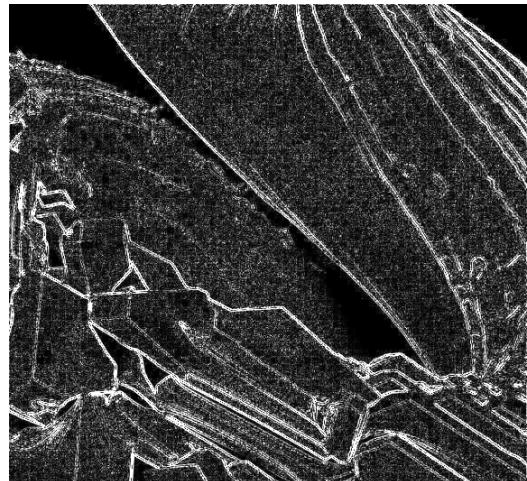
*Image 9: robert\_result*



*Image 10: prewitt\_result*



*Image 11: sobel\_result*



*Image 12:LoG\_result*

对于图四：



Image 13: robert\_result



Image 14: prewitt\_result



Image 15: sobel\_result

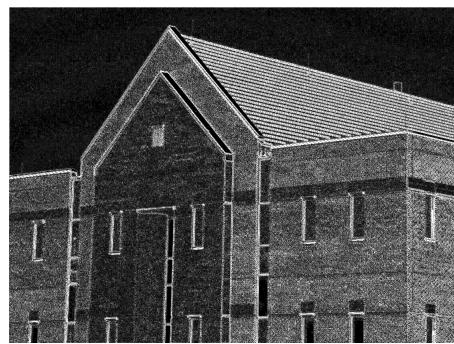


Image 16:LoG\_result

## 实验结论

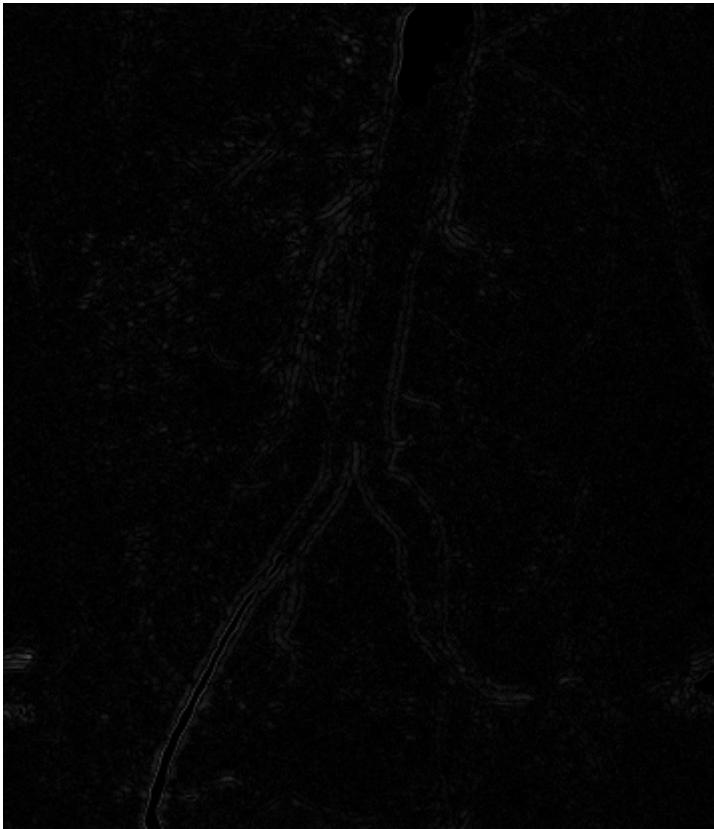
### 对于图一：

robert算子边缘检测模板大小只有 $2 \times 2$ 大小，对于原图而言，可能会损失一些细节，从而使图一的robert算子检测出来的图像看起来过暗，看不到太多细节，其次，由于该算子对于水平或者垂直的边缘响应不是很剧烈，也容易丢弃一些细节，是图片看起来过于简单。

prewitt算子边缘检测的算子相对于robert算子所囊括的面积更大，能够保留更多的细节，所以图片看起来细节增多，一些较为不明显的边缘也能够被检测出来。

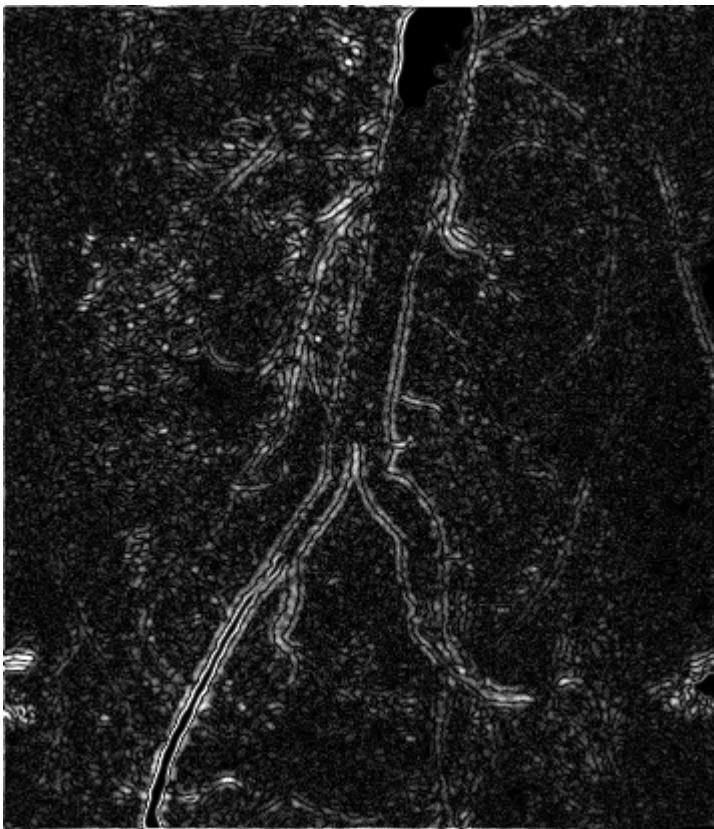
soobel算子相较于prewitt算子加大了中间部分的权重，使得更加突出边缘，所以图一的soobel算子检测结果比prewitt算子的边缘更加明显，更加明亮。

拉普拉斯算子由于对噪声过于敏感，将边缘信息覆盖，使得拉普拉斯算子检测出来的结果如下图所示



几乎看不见检测出来的边缘。

于是用高斯拉普拉斯算子检测得出如下结果：



此结果相较于单纯的拉普拉斯算子边缘检测较好，但是还是收到噪声影响，损失了过多的细节，只能看到些许轮廓。

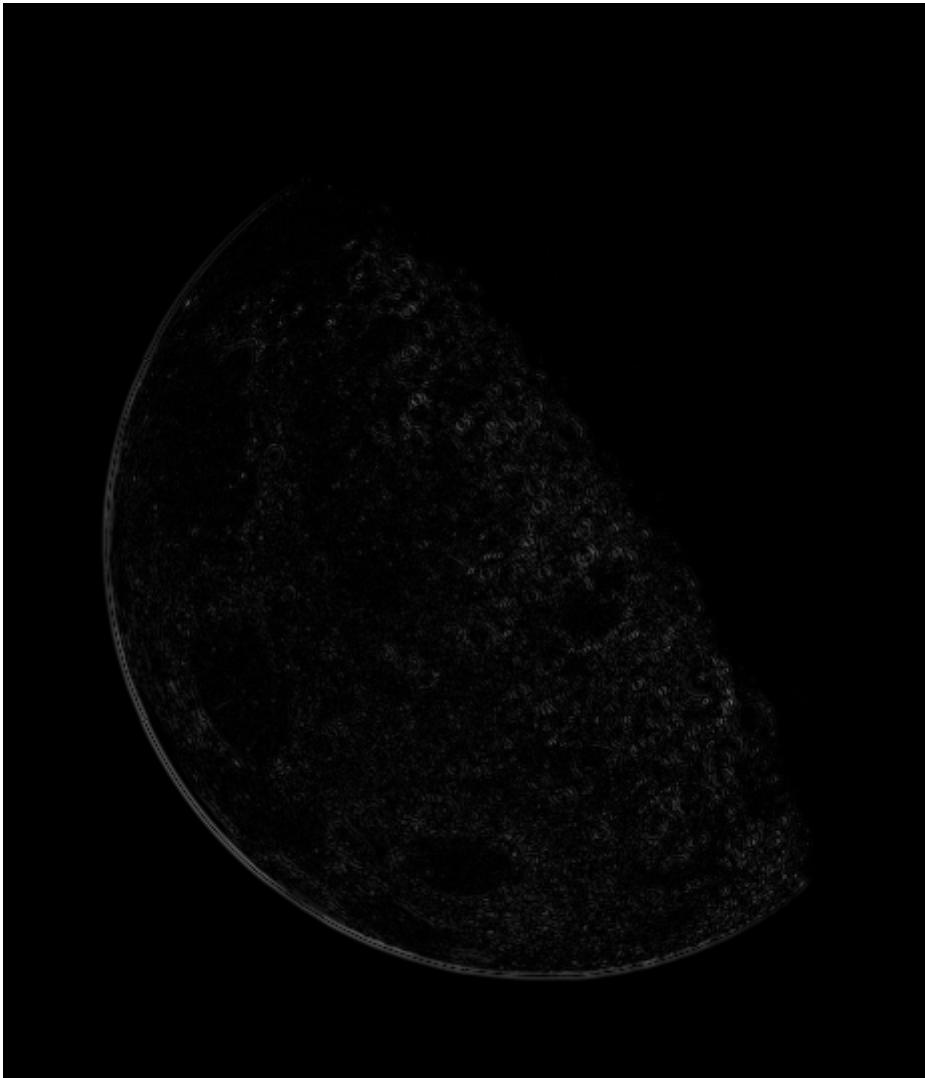
## 对于图二：

robert算子同样因为卷积核过小，不能体现细节部分，只有大概的轮廓，即月亮的最边缘部分，月球表面上的边缘并不是很明显

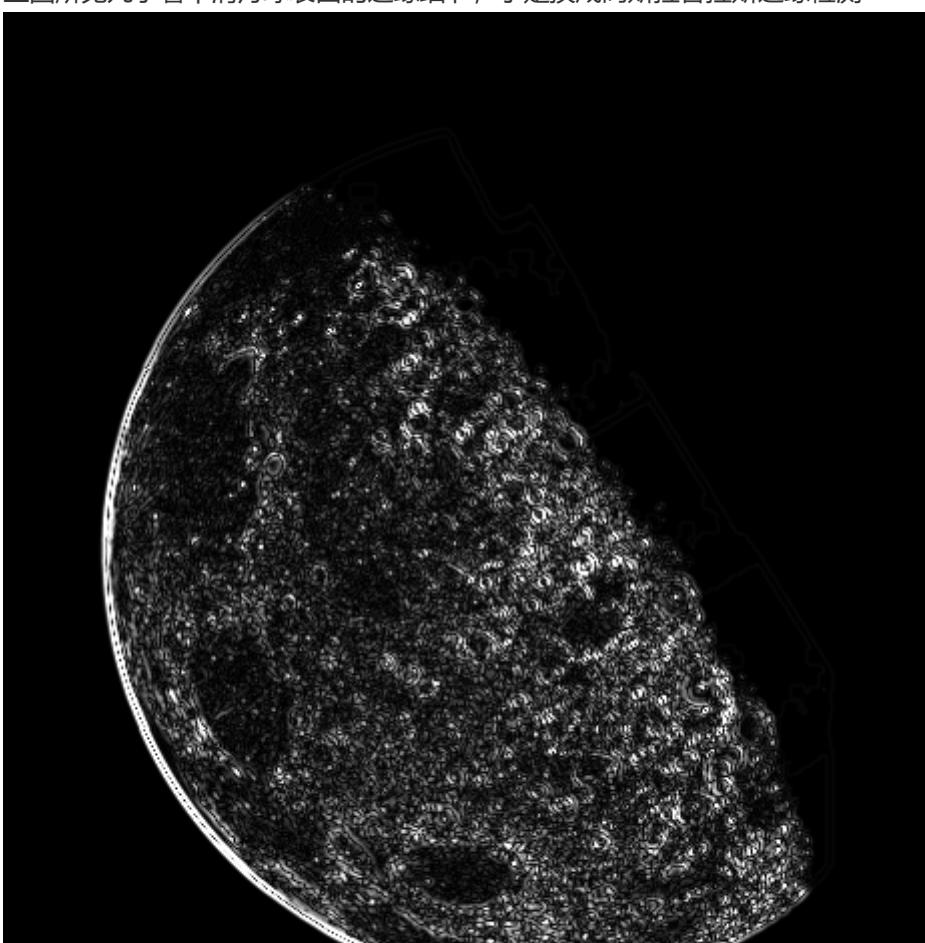
prewitt算子相较于robert算子的卷积核较为复杂，能保存的月球表面的细节更多

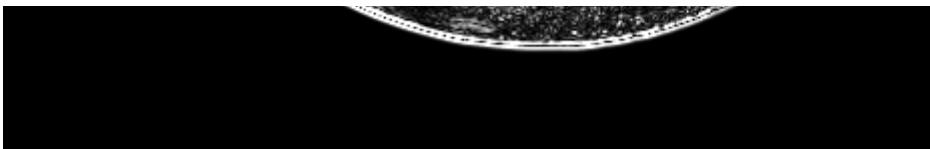
sobel算子在prewitt算子的基础上增加了核心像素周围的权重，使得核心像素占比更大，边缘突出明显

拉普拉斯算子由于受噪声影响太大，检测结果如图：



上图所见几乎看不清月球表面的边缘细节，于是换成高斯拉普拉斯边缘检测





可以看出边缘明显增强，但是由于噪声影响依旧在，没有完全消除，导致假边缘出现过多。

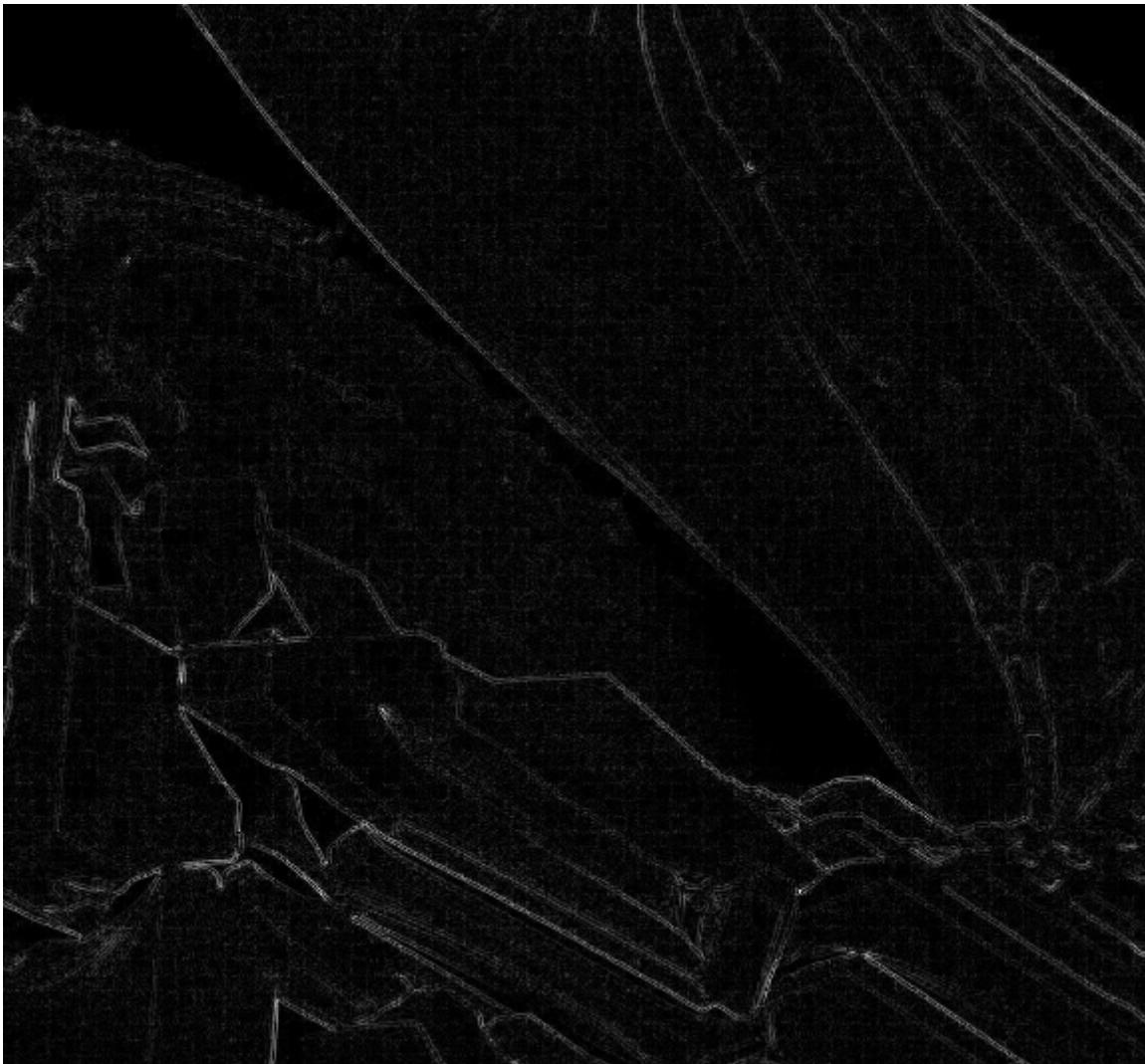
### 对于图三：

robert算子同样因为卷积核过小，不能体现细节部分，只有大概的轮廓，即建筑物的最边缘部分，建筑物上的边缘细节并不是很明显

prewitt算子相较于robert算子的卷积核较为复杂，能保存表面的细节更多

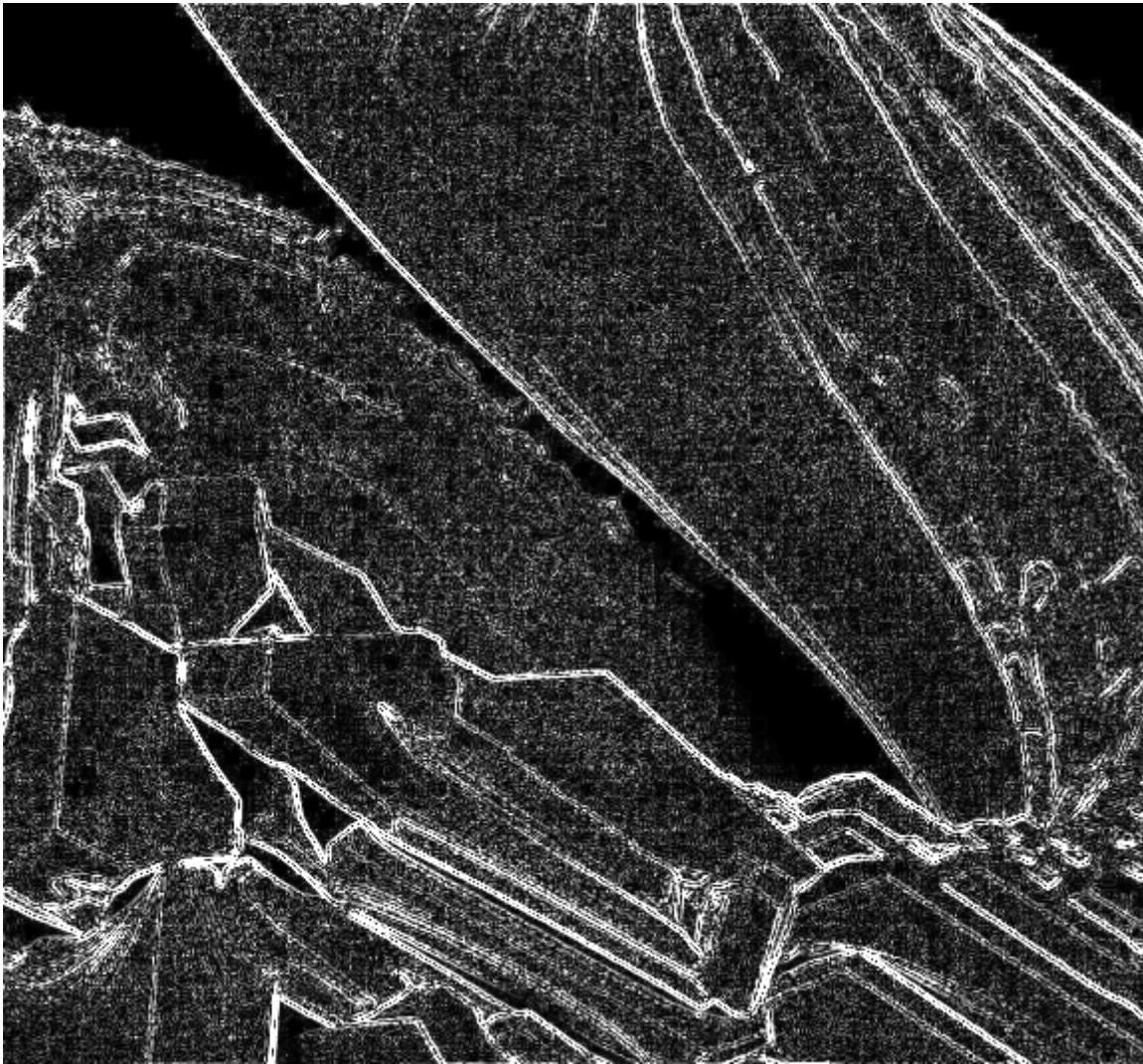
sobel算子在prewitt算子的基础上增加了核心像素周围的权重，使得核心像素占比更大，边缘突出明显，也增多了一些边缘细节

拉普拉斯算子边缘检测由于受噪声影响，检测结果如下图：



可以看出，只有较为明显的边缘能检测出来，并且检测出的结果也很模糊，受噪声影响过大

换用高斯拉普拉斯算子检测得到结果如下：



也能看出噪声污染严重，但是能检测出一些细节部分，相较于原拉普拉斯算子效果变好

#### 对于图四：

robert算子同样因为卷积核过小，不能体现细节部分，只有大概的轮廓。且robert算子对于水平和垂直部分检测效果不是很好，从图中可以看出房子的垂直的边缘检测效果不好，几乎检测不出房子的垂直边缘和水平边缘，最明显的就是倾斜边缘。

prewitt算子相较于robert算子的卷积核较为复杂，能保存表面的细节更多，并且其垂直边缘和水平边缘明显强于robert算子检测结果。

sobel算子在prewitt算子的基础上增加了核心像素周围的权重，使得核心像素占比更大，边缘突出明显

拉普拉斯算子因为对噪声过于敏感，检测出的图像几乎都是很明显的边缘，对于房子墙体的边缘则没有过大响应



于是改用高斯拉普拉斯算子检测，结果如图：



上图虽然对墙体的边缘有了较大的响应，但是整体有过重的噪声污染，较拉普拉斯算子检测效果好。