Hw4_Color edge detection (Due. 6/16)

409410014 資工三 柯柏旭 (hand in 6/4)

Technical description

測試環境

```
❯ uname -a
Linux hentci-Aspire-A515-52G 5.15.0-69-generic #76-Ubuntu SMP Fri Mar 17 17:19:29 UTC 2023
 x86_64 x86_64 x86_64 GNU/Linux
```

使用語言:

`python 3.10.6`

library-requirement:

```
matplotlib==3.7.1
numpy==1.23.5
opencv_python==4.7.0.72
Pillow==9.5.0
```

如何執行

```
python3 Edge_Detection.py
```

或是

```
python Edge_Detection.py
```

便會依序顯示以下內容:

```
共3張圖, 每張圖包含 original image 和經過 Edge Detection(Sobel Operator) 後的 image
1. baboon.png (original, Gx, Gy, G)
2. peppers.png (original, Gx, Gy, G)
3. pool.png (original, Gx, Gy, G)
```

如果要單獨看各張圖，可以把程式碼中標記的地方解註解後執行。

當中還有實做除了傳統3x3以外的sobel operator, 包含5x5及10x10。如果要使用，除了替換sobel operator以外，還需要更改padding和套用sobel operator的一些數值。

也另外實做了UI版本的程式，可以輸入:

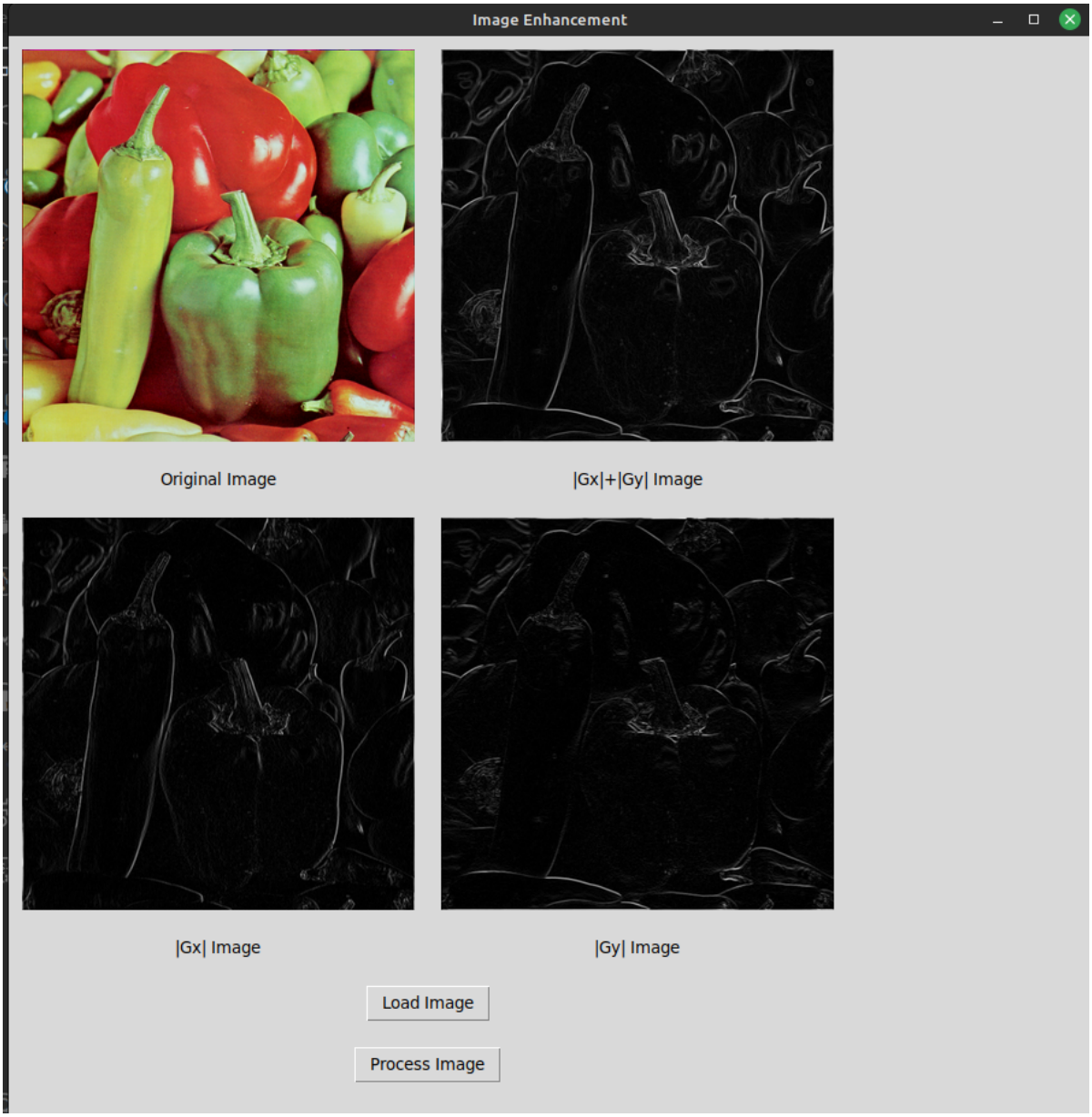
```
python3 UI_version.py
```

或是

```
python UI_version.py
```

執行後，從`Load Image`這個按鈕讀入欲執行的圖片，並且按下`Process Image`便會自動對圖片邊緣偵測。

示例圖:



程式碼解釋

Sobel Operator

```python
def sobel_operator(image):
    # Convert image to grayscale
    gray = image.mean(axis=2)

    # Define the Sobel kernels
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    # Pad the image to handle border pixels
    padded_image = np.pad(gray, ((1, 1), (1, 1)), mode='constant')

    # Initialize output arrays
    gradient_x = np.zeros_like(gray, dtype=np.float32)
```

```
    gradient_y = np.zeros_like(gray, dtype=np.float32)

    # Apply the Sobel operator
    for i in range(gray.shape[0]):
        for j in range(gray.shape[1]):
            gradient_x[i, j] = np.sum(padded_image[i:i+3, j:j+3] * sobel_x)
            gradient_y[i, j] = np.sum(padded_image[i:i+3, j:j+3] * sobel_y)

    # Compute the magnitude of the gradients
    gradient_magnitude = np.sqrt(gradient_x**2 + gradient_y**2)
    gradient_x = np.abs(gradient_x)
    gradient_y = np.abs(gradient_y)

    # Normalize the gradient magnitude to 0-255
    gradient_magnitude = (gradient_magnitude / np.max(gradient_magnitude)) * 255
    gradient_x = (gradient_x / np.max(gradient_x)) * 255
    gradient_y = (gradient_y / np.max(gradient_y)) * 255

    # Convert the gradient magnitude to uint8 format
    gradient_magnitude = gradient_magnitude.astype(np.uint8)
    gradient_x = gradient_x.astype(np.uint8)
    gradient_y = gradient_y.astype(np.uint8)

    return gradient_magnitude, gradient_x, gradient_y
```

主要步驟如下:

1. 將輸入的彩色圖像轉換為灰度圖像，以簡化處理。
2. 定義了Sobel算子的兩個卷積核（kernels）：sobel_x用於檢測水平邊緣，sobel_y用於檢測垂直邊緣。
3. 對灰度圖像進行邊界填充，以處理圖像邊緣上的像素。
4. 初始化與灰度圖像大小相同的輸出陣列，用於存儲計算後的梯度。
5. 使用Sobel算子計算每個像素的梯度值。
6. 根據梯度值計算梯度的大小（gradient_magnitude）以及水平（gradient_x）和垂直（gradient_y）方向上的梯度值。
7. 對梯度大小和梯度方向進行歸一化處理，將其範圍映射到0-255之間。
8. 將梯度大小和梯度方向的值轉換為無符號8位整數（uint8）的格式。
9. 返回計算後的梯度大小（gradient_magnitude）、水平梯度（gradient_x）和垂直梯度（gradient_y）。

sobel operator (3 x 3)

- g_x

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

- g_y

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

$$g = \sqrt{g_x^2 + g_y^2}$$

sobel operator (5 x 5)

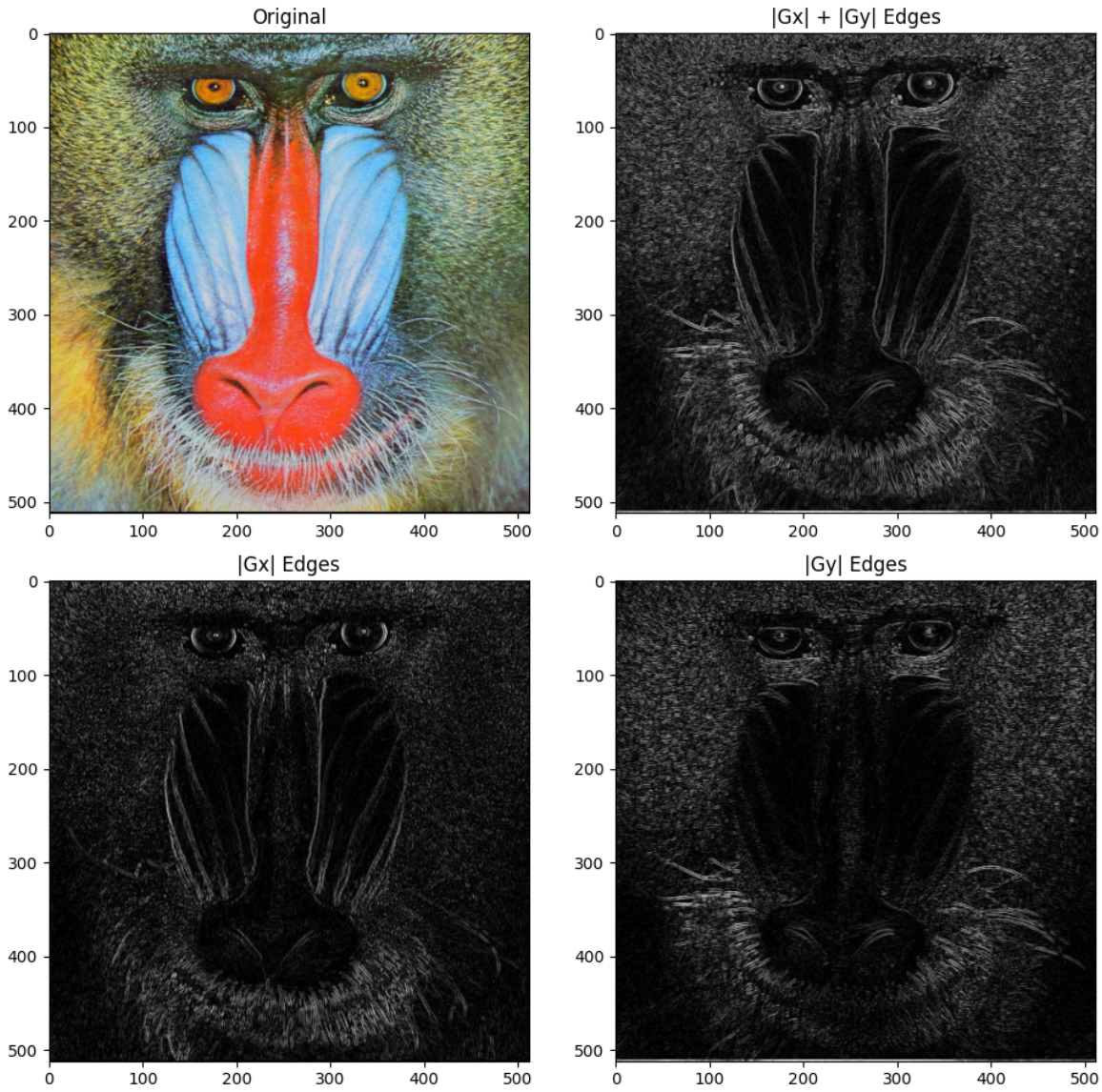
```
''' kernel = 5 x 5 '''
sobel_x = np.array([[-1, -2, 0, 2, 1],
                    [-4, -8, 0, 8, 4],
                    [-6, -12, 0, 12, 6],
                    [-4, -8, 0, 8, 4],
                    [-1, -2, 0, 2, 1]])

sobel_y = np.array([[-1, -4, -6, -4, -1],
                    [-2, -8, -12, -8, -2],
                    [0, 0, 0, 0, 0],
                    [2, 8, 12, 8, 2],
                    [1, 4, 6, 4, 1]])
```

sobel operator (10 x 10)

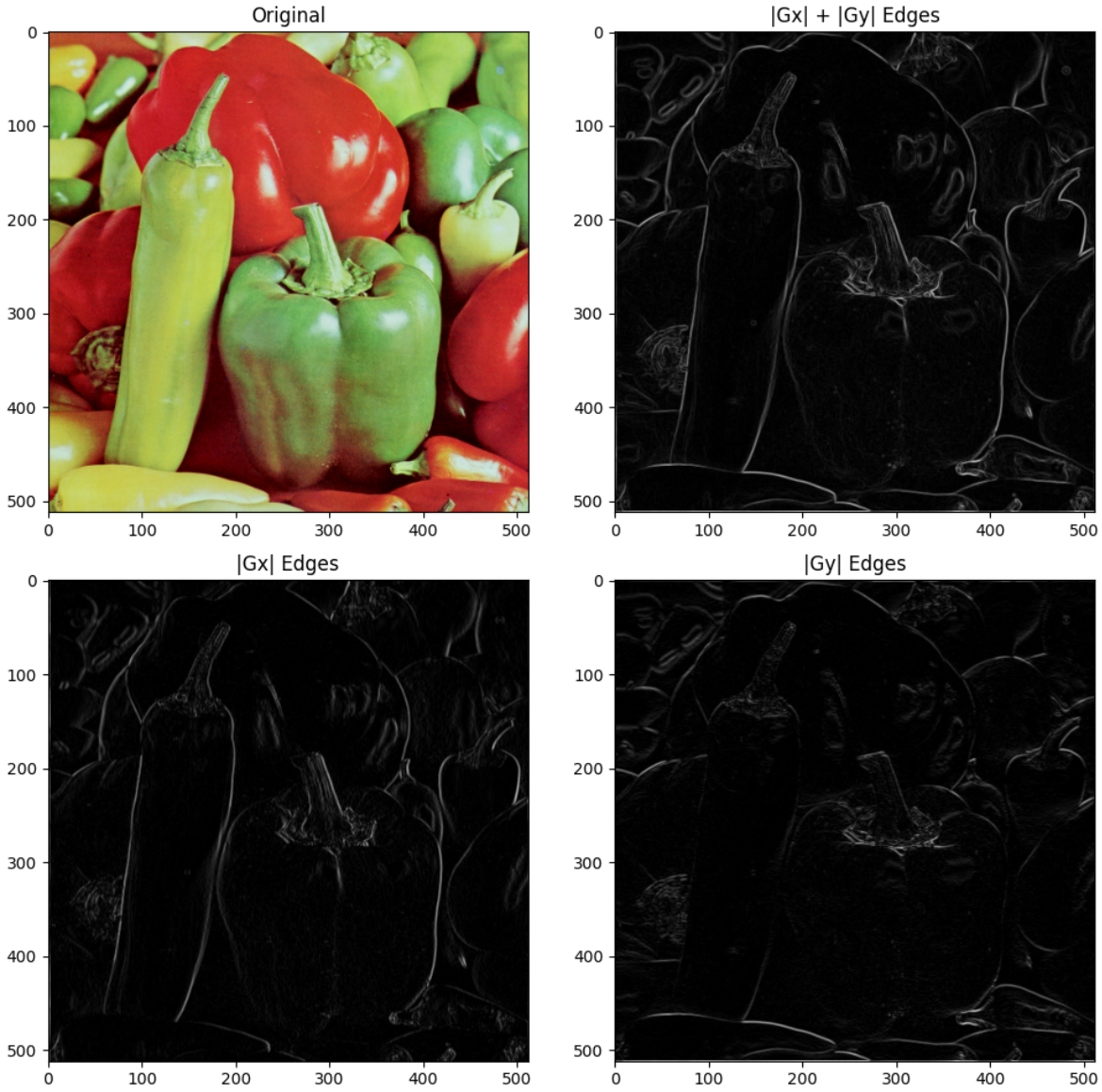
```
''' kernel = 10 x 10 '''
sobel_x = np.array([[-1, -2, -3, -4, -5, 0, 5, 4, 3, 2],
                    [-2, -4, -6, -8, -10, 0, 10, 8, 6, 4],
                    [-3, -6, -9, -12, -15, 0, 15, 12, 9, 6],
                    [-4, -8, -12, -16, -20, 0, 20, 16, 12, 8],
                    [-5, -10, -15, -20, -25, 0, 25, 20, 15, 10],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [5, 10, 15, 20, 25, 0, -25, -20, -15, -10],
                    [4, 8, 12, 16, 20, 0, -20, -16, -12, -8],
                    [3, 6, 9, 12, 15, 0, -15, -12, -9, -6],
                    [2, 4, 6, 8, 10, 0, -10, -8, -6, -4]])

sobel_y = np.array([[-1, -2, -3, -4, -5, 0, 5, 4, 3, 2],
                    [-2, -4, -6, -8, -10, 0, 10, 8, 6, 4],
                    [-3, -6, -9, -12, -15, 0, 15, 12, 9, 6],
                    [-4, -8, -12, -16, -20, 0, 20, 16, 12, 8],
                    [-5, -10, -15, -20, -25, 0, 25, 20, 15, 10],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [5, 10, 15, 20, 25, 0, -25, -20, -15, -10],
                    [4, 8, 12, 16, 20, 0, -20, -16, -12, -8],
                    [3, 6, 9, 12, 15, 0, -15, -12, -9, -6],
                    [2, 4, 6, 8, 10, 0, -10, -8, -6, -4]])
```

Experimental results

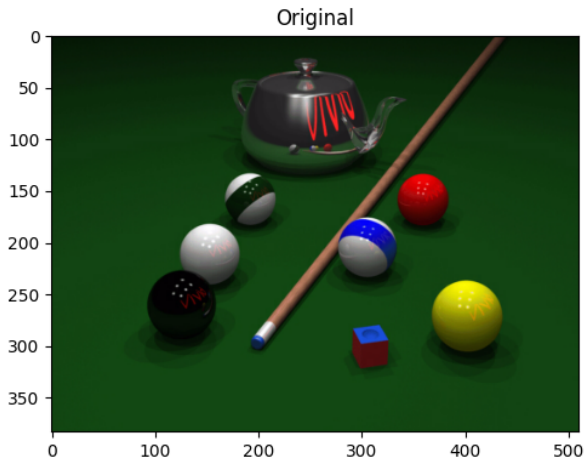
baboon.png

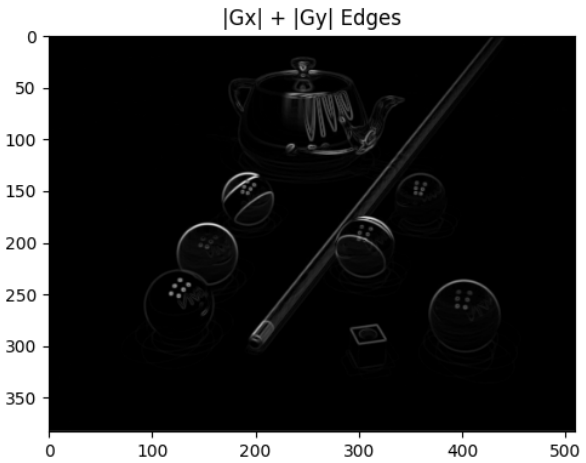


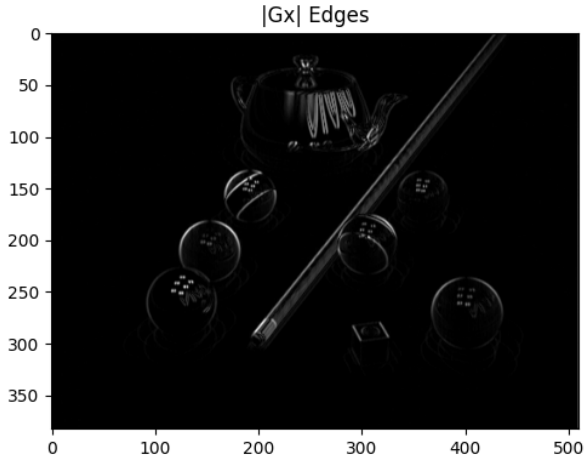
peppers.png

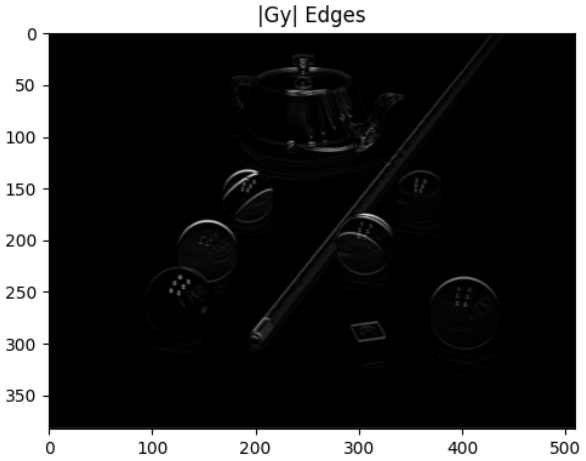


pool.png

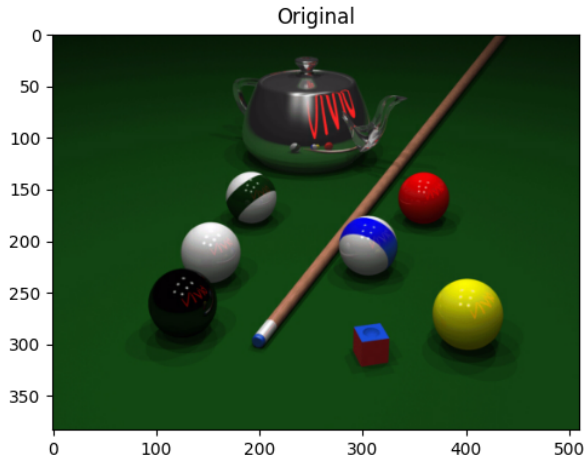

Original
0
50
100
150
200
250
300
350
0
100
200
300
400
500

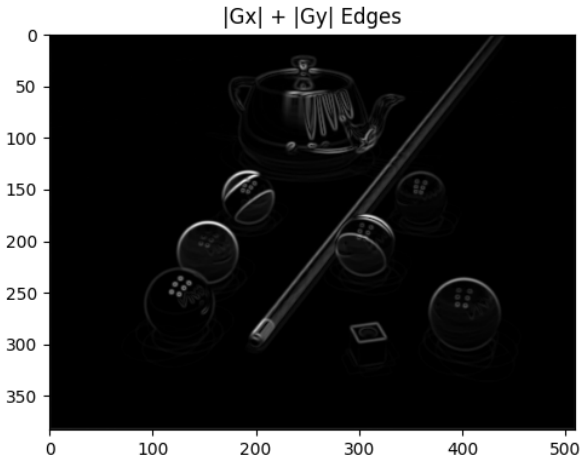

|Gx| + |Gy| Edges
0
50
100
150
200
250
300
350
0
100
200
300
400
500

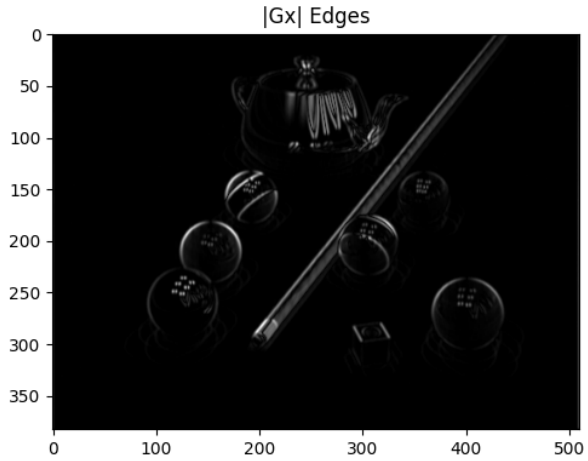

|Gx| Edges
0
50
100
150
200
250
300
350
0
100
200
300
400
500

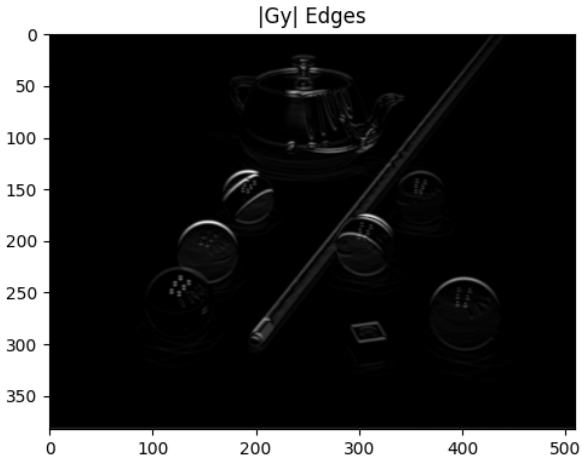

|Gy| Edges
0
50
100
150
200
250
300
350
0
100
200
300
400
500

pool.png(5 x 5)

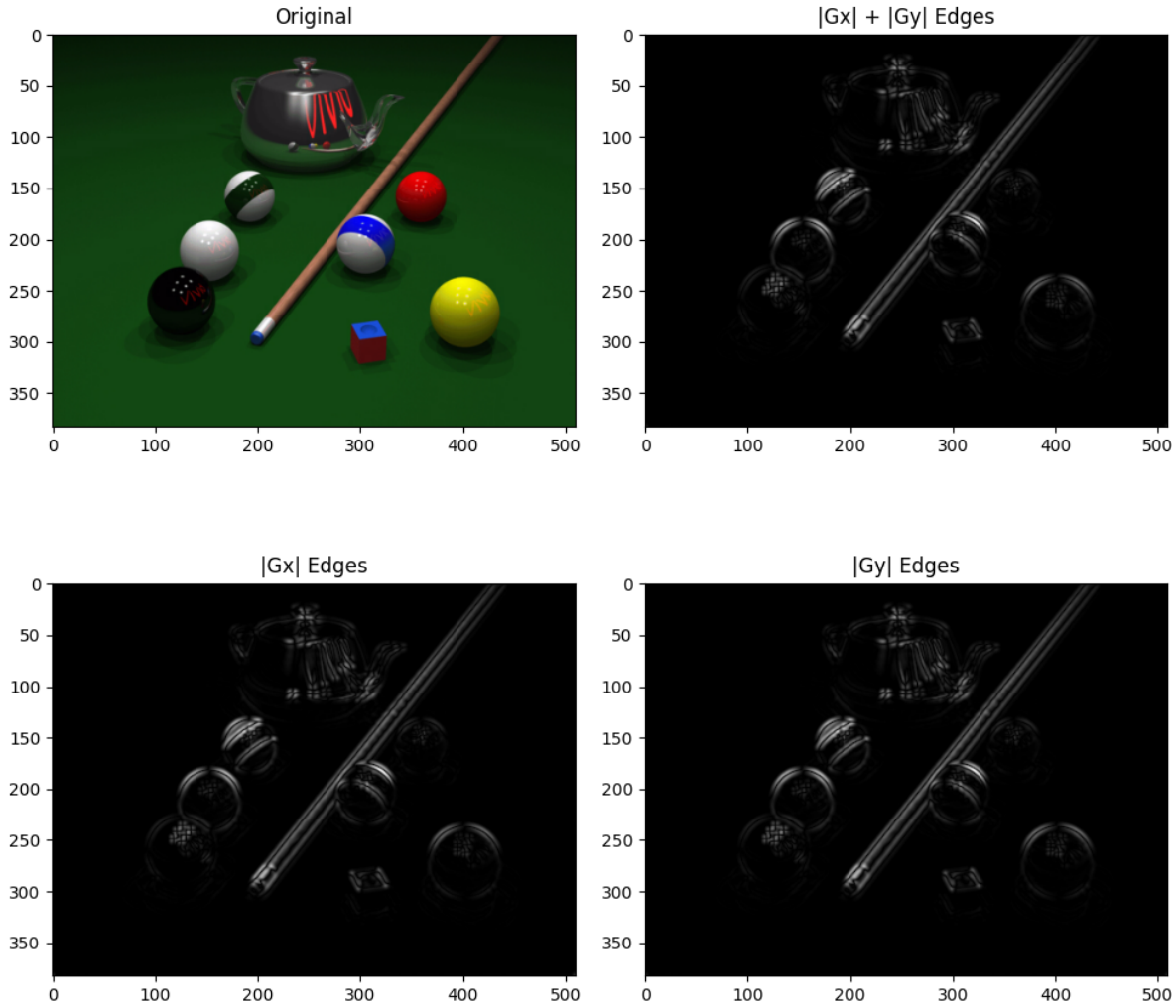

Original
0
50
100
150
200
250
300
350
0
100
200
300
400
500


|Gx| + |Gy| Edges
0
50
100
150
200
250
300
350
0
100
200
300
400
500


|Gx| Edges
0
50
100
150
200
250
300
350
0
100
200
300
400
500


|Gy| Edges
0
50
100
150
200
250
300
350
0
100
200
300
400
500

pool.png(10 x 10)

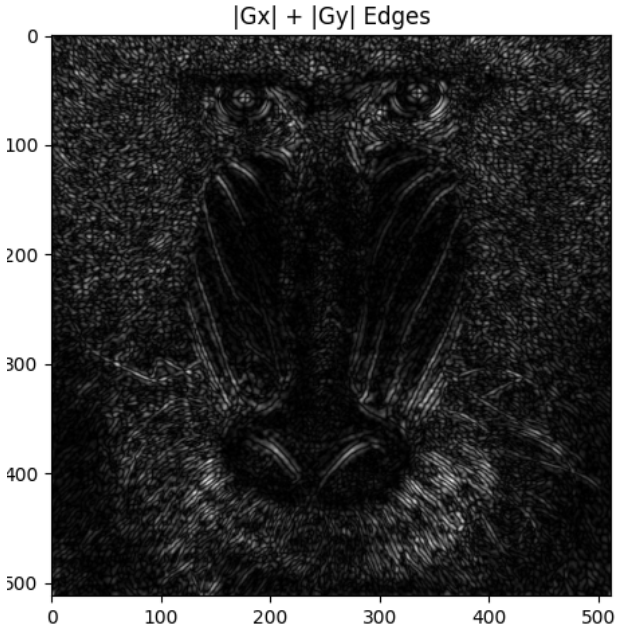


Discussions

從上方的實驗結果與比較可以發現:

水平梯度G_x和垂直梯度G_y兩者其實單獨就可以做出不錯的邊緣偵測了。不過藉由梯度合成(平方相加開根號)，可以獲得更全面的邊緣信息。

另外，額外時做了kernel 5 x 5和10 x10的sobel operator。原本以為是mask越大越好 ，結果好像大過頭也不太好，像是kernel = 10的時候，就好像在邊地震邊做edge detection一樣。如圖:



不過說真的，單用Sobel operator能獲得的信息還是有限，上面圖像的邊緣感覺還是沒說非常清楚。

總而言之，我認為Sobel operator最大的優點是，針對每個點，它只需要做八個點的整數運算就可以算出結果，這在運算量的負擔上是相當輕的。然而，它只用一個3x3的範圍來得到結果，顯然這樣算出來的結果是不太準確的。

心得

比起上一次作業，這一次真的簡單許多，沒想到來到了第10單元還是回到了Laplacian和Sobel operator，有種反樸歸真的感覺。這次的code其實一開始寫得很複雜，不過後來有用各種numpy優化，少了大概一半的篇幅。numpy真的好強，尤其是padding的部份真好扯，一行就解決了 O口O

這次也有實做UI版本的code，還請助教跑跑看，嘿嘿。

總而言之，這學期下來，雖然這堂課的作業有些部份蠻吃力的，不過看到結果成功還是會很開心。也感謝助教的批改，辛苦了！

References and Appendix

- 邊緣偵測
- 老師ppt CH10