電機所-陳治言-R06921084

Description：

一開始，先處理圖片檔的標頭資訊，包括 size，offset，width，height，bits 等資訊，之後再將圖片的色彩值讀入，因為這次範例圖片只有 8 位元色彩，所以每個畫素只需要 1byte 即可儲存。而這次作業是對圖片做邊緣偵測，並使用不同的演算法，處理完圖片之後，再將標頭與畫素資料一起寫出並存，並且產生七張圖片，即可完成這次的作業。

Algorithm：

Robert's Operator 使用下列的 mask，r_1 與 r_2，並計算 $\sqrt{r_1 + r_2}$。

-1	
	1

	-1
1	

Prewitt's Edge Detector 使用下列的 mask，p_1 與 p_2，並計算 $\sqrt{p_1 + p_2}$。

-1	-1	-1
1	1	1

-1		1
-1		1
-1		1

Sobel's Edge Detector 使用下列的 mask，s_1 與 s_2，並計算 $\sqrt{s_1 + s_2}$。

-1	-2	-1
1	2	1

-1		1
-2		2
-1		1

Frei and Chen's Gradient Operator 使用下列的 mask，f_1 與 f_2，並計算出值 $\sqrt{f_1 + f_2}$。

-1	$-\sqrt{2}$	-1
1	$\sqrt{2}$	1

-1		1
$-\sqrt{2}$		$\sqrt{2}$
-1		1

Kirsch's Compass Operator 使用下列的 mask，k_1 到 k_8，並找出他們的最大值。

-3	-3	5
-3		5
-3	-3	5

-3	5	5
-3		5
-3	-3	-3

5	5	5
-3		-3
-3	-3	-3

5	5	-3
5		-3
-3	-3	-3

5	-3	-3
5		-3
5	-3	-3

-3	-3	-3
5		-3
5	5	-3

-3	-3	-3
-3		-3
5	5	5

-3	-3	-3
-3		5
-3	5	5

Robinson's Compass Operator 使用下列的 mask，r_1 到 r_8，並找出他們的最大值。

-1		1
-2		2
-1		1

	1	2
-1		1
-2	-1	

1	2	1
-1	-2	-1

2	1	
1		-1
	-1	-2

1		-1
2		-2
1		-1

	-1	-2
1		-1
2	1	

-1	-2	-1
1	2	1

-2	-1	-3
-1		1
	1	2

Nevatia-Babu 5x5 Operator 使用下列的 mask，n_1 到 n_6，並找出他們的最大值。

100	100	100	100	100
100	100	100	100	100
0	0	0	0	0
-100	-100	-100	-100	-100
-100	-100	-100	-100	-100

100	100	100	100	100
100	100	100	78	-32
100	92	0	-92	-100
32	-78	-100	-100	-100
-100	-100	-100	-100	-100

100	100	100	32	-100
100	100	92	-78	-100
100	100	0	-100	-100
100	78	-92	-100	-100
100	-32	-100	-100	-100

-100	-100	0	100	100
-100	-100	0	100	100
-100	-100	0	100	100
-100	-100	0	100	100
-100	-100	0	100	100

-100	32	100	100	100
-100	-78	92	100	100
-100	-100	0	100	100
-100	-100	-92	78	100
-100	-100	-100	-32	100

100	100	100	100	100
-32	78	100	100	100
-100	-92	0	100	100
-100	-100	-100	-78	32
-100	-100	-100	-100	-100

PrincipalCode：

◎Robert

```
for(i=2; i<bmpInfo.biHeight; ++i)
    for(j=2; j<bmpInfo.biWidth; ++j)
        r1 = tmp[i-1][j+1] - tmp[i][j];
        r2 = tmp[i-1][j] - tmp[i][j+1];
        BMPoutput_data[i-1][j-1].color = (sqrt(pow(r1, 2) + pow(r2, 2)) > threshold) ? 0 :
        255;
```

◎Prewitt

```
for(i=2; i<bmpInfo.biHeight; ++i)
    for(j=2; j<bmpInfo.biWidth; ++j)
        p1 = tmp[i-1][j-1] + tmp[i-1][j] + tmp[i-1][j+1] - tmp[i+1][j-1] - tmp[i+1][j] -
        tmp[i+1][j+1];
        p2 = tmp[i-1][j+1] + tmp[i][j+1] + tmp[i+1][j+1] - tmp[i-1][j-1] - tmp[i][j-1] -
        tmp[i+1][j-1];
        BMPoutput_data[i-1][j-1].color = (sqrt(pow(p1, 2) + pow(p2, 2)) > threshold) ? 0 :
        255;
```

◎Sobel

```
for(i=2; i<bmpInfo.biHeight; ++i)
    for(j=2; j<bmpInfo.biWidth; ++j)
        s1 = tmp[i-1][j-1] + 2 * tmp[i-1][j] + tmp[i-1][j+1] - tmp[i+1][j-1] - 2 * tmp[i+1][j] -
        tmp[i+1][j+1];
        s2 = tmp[i-1][j+1] + 2 * tmp[i][j+1] + tmp[i+1][j+1] - tmp[i-1][j-1] - 2 * tmp[i][j-1] -
        tmp[i+1][j-1];
        BMPoutput_data[i-1][j-1].color = (sqrt(pow(s1, 2) + pow(s2, 2)) > threshold) ? 0 :
        255;
```

◎Frei_Chen

```
for(i=2; i<bmpInfo.biHeight; ++i)
    for(j=2; j<bmpInfo.biWidth; ++j)
        f1 = tmp[i-1][j-1] + sqrt(2 * tmp[i-1][j] * tmp[i-1][j]) + tmp[i-1][j+1] - tmp[i+1][j-1]
        - sqrt(2 * tmp[i+1][j] * tmp[i+1][j]) - tmp[i+1][j+1];
        f2 = tmp[i-1][j+1] + sqrt(2 * tmp[i][j+1] * tmp[i][j+1]) + tmp[i+1][j+1] - tmp[i-1][j-
        1] - sqrt(2 * tmp[i][j-1] * tmp[i][j-1]) - tmp[i+1][j-1];
        BMPoutput_data[i-1][j-1].color = (sqrt(pow(f1, 2) + pow(f2, 2)) > threshold) ? 0 :
        255;
```

◎Kirsch

```
for(i=2; i<bmpInfo.biHeight; ++i)
    for(j=2; j<bmpInfo.biWidth; ++j)
        k[0] = 5 * (tmp[i-1][j+1] + tmp[i][j+1] + tmp[i+1][j+1]) - 3 * (tmp[i+1][j] +
        tmp[i+1][j-1] + tmp[i][j-1] + tmp[i-1][j-1] + tmp[i-1][j]);
        k[1] = 5 * (tmp[i-1][j] + tmp[i-1][j+1] + tmp[i][j+1]) - 3 * (tmp[i+1][j+1] +
        tmp[i+1][j] + tmp[i+1][j-1] + tmp[i][j-1] + tmp[i-1][j-1]);
        k[2] = 5 * (tmp[i-1][j-1] + tmp[i-1][j] + tmp[i-1][j+1]) - 3 * (tmp[i][j+1] +
        tmp[i+1][j+1] + tmp[i+1][j] + tmp[i+1][j-1] + tmp[i][j-1]);
        k[3] = 5 * (tmp[i][j-1] + tmp[i-1][j-1] + tmp[i-1][j]) - 3 * (tmp[i-1][j+1] +
        tmp[i][j+1] + tmp[i+1][j+1] + tmp[i+1][j] + tmp[i+1][j-1]);
        k[4] = 5 * (tmp[i+1][j-1] + tmp[i][j-1] + tmp[i-1][j-1]) - 3 * (tmp[i-1][j] + tmp[i-
        1][j+1] + tmp[i][j+1] + tmp[i+1][j+1] + tmp[i+1][j]);
        k[5] = 5 * (tmp[i+1][j] + tmp[i+1][j-1] + tmp[i][j-1]) - 3 * (tmp[i-1][j-1] + tmp[i-
        1][j] + tmp[i-1][j+1] + tmp[i][j+1] + tmp[i+1][j+1]);
        k[6] = 5 * (tmp[i+1][j+1] + tmp[i+1][j] + tmp[i+1][j-1]) - 3 * (tmp[i][j-1] + tmp[i-
        1][j-1] + tmp[i-1][j] + tmp[i-1][j+1] + tmp[i][j+1]);
        k[7] = 5 * (tmp[i][j+1] + tmp[i+1][j+1] + tmp[i+1][j]) - 3 * (tmp[i+1][j-1] +
        tmp[i][j-1] + tmp[i-1][j-1] + tmp[i-1][j] + tmp[i-1][j+1]);
        BMPoutput_data[i-1][j-1].color = (*max_element(k, k+8) > threshold) ? 0 : 255;
```

◎Robinson

```
for(i=2; i<bmpInfo.biHeight; ++i)
    for(j=2; j<bmpInfo.biWidth; ++j)
        r[0] = tmp[i-1][j+1] + 2 * tmp[i][j+1] + tmp[i+1][j+1] - tmp[i+1][j-1] - 2 * tmp[i][j-
        1] - tmp[i-1][j-1];
        r[1] = tmp[i-1][j] + 2 * tmp[i-1][j+1] + tmp[i][j+1] - tmp[i+1][j] - 2 * tmp[i+1][j-1] -
        tmp[i][j-1];
        r[2] = tmp[i-1][j-1] + 2 * tmp[i-1][j] + tmp[i-1][j+1] - tmp[i+1][j+1] - 2 * tmp[i+1][j]
        - tmp[i+1][j-1];
```

```
r[3] = tmp[i][j-1] + 2 * tmp[i-1][j-1] + tmp[i-1][j] - tmp[i][j+1] - 2 * tmp[i+1][j+1] -
tmp[i+1][j];
r[4] = tmp[i+1][j-1] + 2 * tmp[i][j-1] + tmp[i-1][j-1] - tmp[i-1][j+1] - 2 * tmp[i][j+1]
- tmp[i+1][j+1];
r[5] = tmp[i+1][j] + 2 * tmp[i+1][j-1] + tmp[i][j-1] - tmp[i-1][j] - 2 * tmp[i-1][j+1] -
tmp[i][j+1];
r[6] = tmp[i+1][j+1] + 2 * tmp[i+1][j] + tmp[i+1][j-1] - tmp[i-1][j-1] - 2 * tmp[i-1][j]
- tmp[i-1][j+1];
r[7] = tmp[i][j+1] + 2 * tmp[i+1][j+1] + tmp[i+1][j] - tmp[i][j-1] - 2 * tmp[i-1][j-1] -
tmp[i-1][j];
BMPoutput_data[i-1][j-1].color = (*max_element(r, r+8) > threshold) ? 0 : 255;
```

◎Nevatia_Babu

```
for(i=4; i<bmpInfo.biHeight+2; ++i)
    for(j=4; j<bmpInfo.biWidth; ++j)
        n[0] = 0 - 100 * tmp[i-2][j-2] - 100 * tmp[i-2][j-1] - 100 * tmp[i-2][j] - 100 * tmp[i-
        2][j+1] - 100 * tmp[i-2][j+2] - 100 * tmp[i-1][j-2] - 100 * tmp[i-1][j-1] - 100 * tmp[i-
        1][j] - 100 * tmp[i-1][j+1] - 100 * tmp[i-1][j+2] + 100 * tmp[i+1][j-2] + 100 *
        tmp[i+1][j-1] + 100 * tmp[i+1][j] + 100 * tmp[i+1][j+1] + 100 * tmp[i+1][j+2] +
        100 * tmp[i+2][j-2] + 100 * tmp[i+2][j-1] + 100 * tmp[i+2][j] + 100 * tmp[i+2][j+1]
        + 100 * tmp[i+2][j+2];

        n[1] = 0 - 100 * tmp[i-2][j-2] - 100 * tmp[i-2][j-1] - 100 * tmp[i-2][j] - 100 * tmp[i-
        2][j+1] - 100 * tmp[i-2][j+2] + 32 * tmp[i-1][j-2] - 78 * tmp[i-1][j-1] - 100 * tmp[i-
        1][j] - 100 * tmp[i-1][j+1] - 100 * tmp[i-1][j+2] + 100 * tmp[i][j-2] + 92 * tmp[i][j-1] -
        92 * tmp[i][j+1] - 100 * tmp[i][j+2] + 100 * tmp[i+1][j-2] + 100 * tmp[i+1][j-1] +
        100 * tmp[i+1][j] + 78 * tmp[i+1][j+1] - 32 * tmp[i+1][j+2] + 100 * tmp[i+2][j-2] +
        100 * tmp[i+2][j-1] + 100 * tmp[i+2][j] + 100 * tmp[i+2][j+1] + 100 *
        tmp[i+2][j+2];

        n[2] = 100 * tmp[i-2][j-2] - 32 * tmp[i-2][j-1] - 100 * tmp[i-2][j] - 100 * tmp[i-2][j+1] -
        100 * tmp[i-2][j+2] + 100 * tmp[i-1][j-2] + 78 * tmp[i-1][j-1] - 92 * tmp[i-1][j] - 100 *
        tmp[i-1][j+1] - 100 * tmp[i-1][j+2] + 100 * tmp[i][j-2] + 100 * tmp[i][j-1] - 100 *
        tmp[i][j+1] - 100 * tmp[i][j+2] + 100 * tmp[i+1][j-2] + 100 * tmp[i+1][j-1] + 92 *
        tmp[i+1][j] - 78 * tmp[i+1][j+1] - 100 * tmp[i+1][j+2] + 100 * tmp[i+2][j-2] + 100 *
        tmp[i+2][j-1] + 100 * tmp[i+2][j] + 32 * tmp[i+2][j+1] - 100 * tmp[i+2][j+2];

        n[3] = 0 - 100 * tmp[i-2][j-2] - 100 * tmp[i-2][j-1] + 100 * tmp[i-2][j+1] + 100 *
        tmp[i-2][j+2] - 100 * tmp[i-1][j-2] - 100 * tmp[i-1][j-1] + 100 * tmp[i-1][j+1]+ 100 *
```

tmp[i-1][j+2] - 100 * tmp[i][j-2] - 100 * tmp[i][j-1] + 100 * tmp[i][j+1] + 100 * tmp[i][j+2] - 100 * tmp[i+1][j-2] - 100 * tmp[i+1][j-1] + 100 * tmp[i+1][j+1] + 100 * tmp[i+1][j+2] - 100 * tmp[i+2][j-2] + -100 * tmp[i+2][j-1] + 0 * tmp[i+2][j] + 100 * tmp[i+2][j+1]+ 100 * tmp[i+2][j+2];

n[4] = 0 - 100 * tmp[i-2][j-2] - 100 * tmp[i-2][j-1] - 100 * tmp[i-2][j] - 32 * tmp[i-2][j+1] + 100 * tmp[i-2][j+2] - 100 * tmp[i-1][j-2] - 100 * tmp[i-1][j-1] - 92 * tmp[i-1][j] + 78 * tmp[i-1][j+1] + 100 * tmp[i-1][j+2] - 100 * tmp[i][j-2] - 100 * tmp[i][j-1] + 100 * tmp[i][j+1]+ 100 * tmp[i][j+2] - 100 * tmp[i+1][j-2] - 78 * tmp[i+1][j-1] + 92 * tmp[i+1][j] + 100 * tmp[i+1][j+1] + 100 * tmp[i+1][j+2] - 100 * tmp[i+2][j-2] + 32 * tmp[i+2][j-1] + 100 * tmp[i+2][j] + 100 * tmp[i+2][j+1] + 100 * tmp[i+2][j+2];

n[5] = 0 - 100 * tmp[i-2][j-2] - 100 * tmp[i-2][j-1] - 100 * tmp[i-2][j] - 100 * tmp[i-2][j+1] - 100 * tmp[i-2][j+2] - 100 * tmp[i-1][j-2] - 100 * tmp[i-1][j-1] - 100 * tmp[i-1][j] - 78 * tmp[i-1][j+1] + 32 * tmp[i-1][j+2] - 100 * tmp[i][j-2] - 92 * tmp[i][j-1] + 92 * tmp[i][j+1] + 100 * tmp[i][j+2] - 32 * tmp[i+1][j-2] + 78 * tmp[i+1][j-1] + 100 * tmp[i+1][j] + 100 * tmp[i+1][j+1] + 100 * tmp[i+1][j+2] + 100 * tmp[i+2][j-2] + 100 * tmp[i+2][j-1] + 100 * tmp[i+2][j] + 100 * tmp[i+2][j+1] + 100 * tmp[i+2][j+2];

BMPoutput_data[i-2][j-2].color = (*max_element(n, n+6) > threshold) ? 0 : 255;

Parameters：

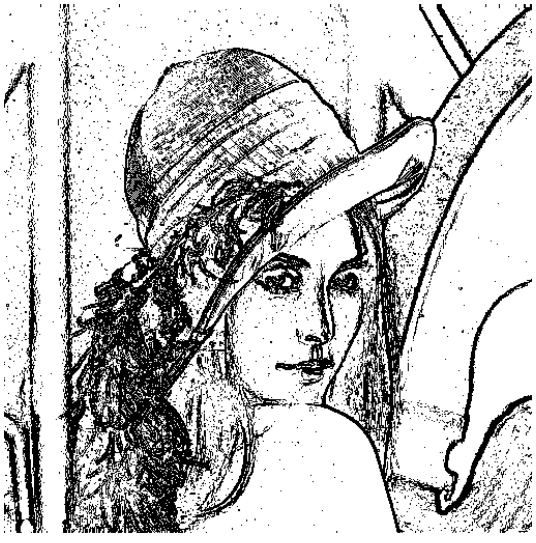
編譯程式碼 g++ -o lena lena.cpp

執行程式 ./lena lena.bmp

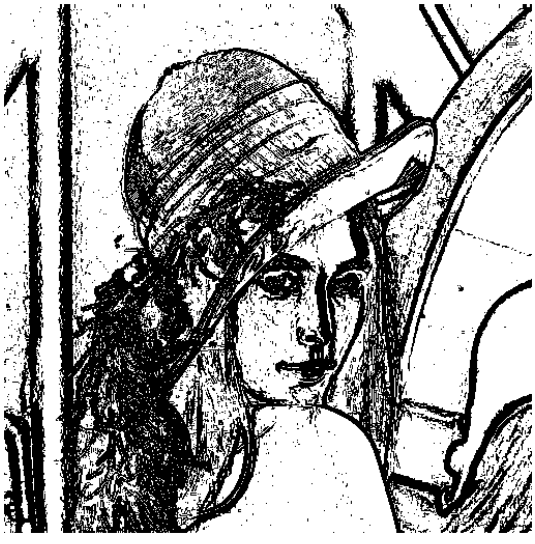
lena.bmp 是我們的 Input

ResultingImages：

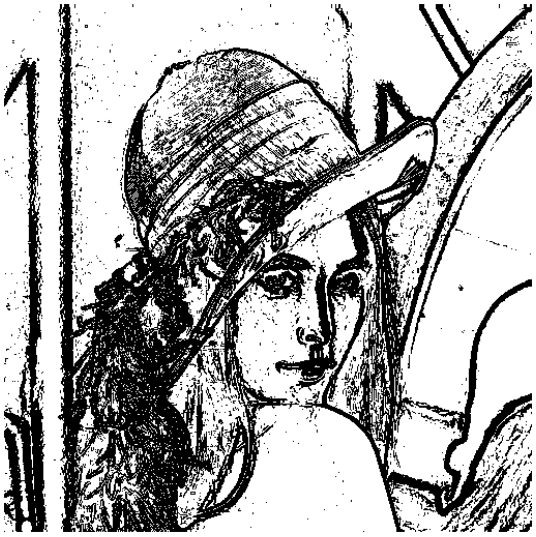
robert_12



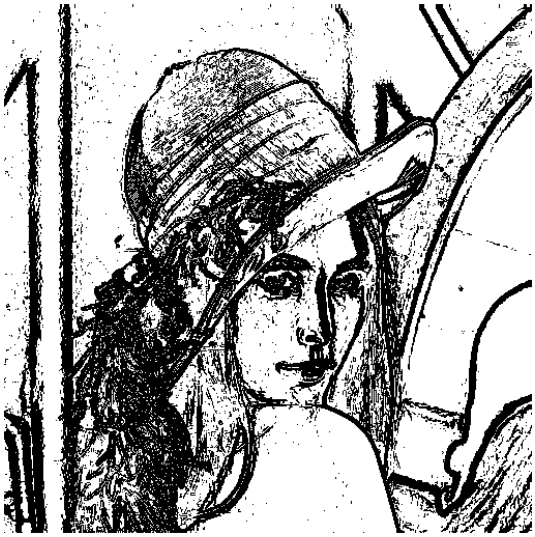
prewitt_24



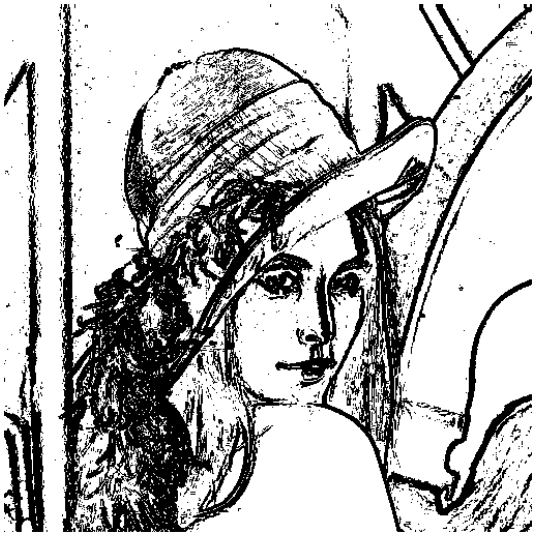
sobel_38



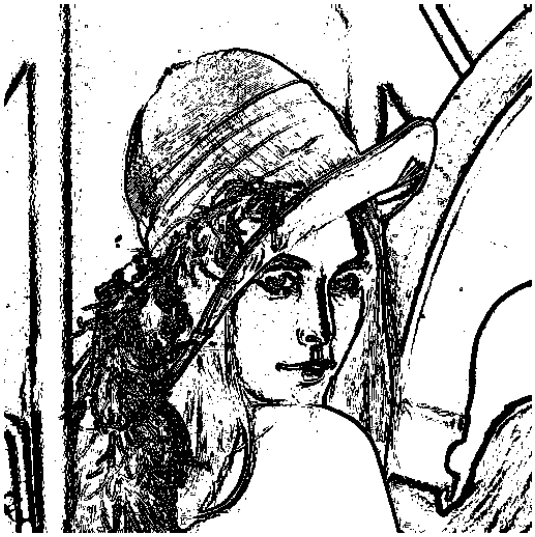
frei&chen_30



kirsch_135



robinson_43



nevatia_babu_5X5_12500