Part 1. Histogram of an Image

Design a software program to read the special .64 image file described in our lecture.
Translate the .64 text file into a 64x64 image with 32 gray levels and store the data in a 2-dimensional array.
Process the image array to obtain the histogram of the image.
Test your program with the following .64 files and plot the histogram of each image. LISA.64, LINCOLN.64, JET.64, LIBERTY.64.
You may plot the image histogram directly in your software program or plot the histogram with any plotting software such as EXCEL or MATLAB.
Designing a function in your program to display the image on the screen is encouraged.

【解釋算法】

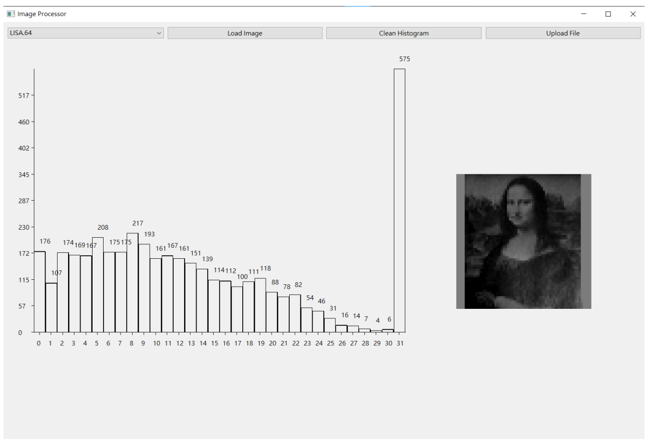
繪製直方圖:

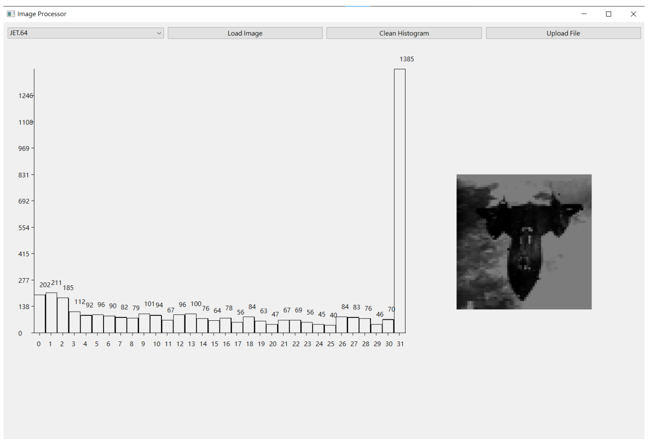
```
void ImageProcessor::generateHistogram()
{
    // 確認直方圖是淨空的
    std::fill(histogram.begin(), histogram.end(), 0);
    int count = 0;
    // 計算每個灰階強度的出現次數
    for (const auto &row : imageArray)
    {
        for (const int value : row)
        {
            histogram[value]++;
            count++;
        }
    }
}

void ImageProcessor::paintEvent(QPaintEvent *event) // 繪製事件 使用update()函數時自動觸發
{
    QPainter painter(this);

    // 繪製直方圖
    drawHistogram(painter);
}

void ImageProcessor::drawHistogram(QPainter &painter)
{
    int barWidth = 22;      // 條形寬度
    int xOffset = 50;       // x 軸起始位置
    int yOffset = 550;      // y 軸起始位置
    int maxHeight = 500;    // 最大高度
    int labelInterval = 5; // 標籤間隔，交錯顯示

    // 計算最大數值
    int maxCount = *std::max_element(histogram.begin(), histogram.end());

    // 繪製 x 軸
    painter.drawLine(xOffset, yOffset, xOffset + 32 * barWidth, yOffset);

    // 繪製 y 軸
    painter.drawLine(xOffset, yOffset, xOffset, yOffset - maxHeight);

    // 繪製直方圖條形
    for (int i = 0; i < 32; i++) // 只顯示到 32
    {
        int barHeight = static_cast<int>((static_cast<double>(histogram[i]) /
maxCount) * maxHeight);
        int x = xOffset + i * barWidth;
        int y = yOffset - barHeight;

        painter.drawRect(x, y, barWidth - 1, barHeight);

        // 繪製條形的數值
        painter.drawText(x + (barWidth - 1) / 2, y - 15,
QString::number(histogram[i]));

        // 繪製 x 軸刻度
        if (i % labelInterval == 0)
        {
            // 繪製 x 軸標籤
            painter.drawText(x + (barWidth - 1) / 2 - 5, yOffset + 25,
QString::number(i));
            // 繪製 x 軸外刻度
            painter.drawLine(x + (barWidth - 1) / 2, yOffset, x + (barWidth - 1) /
2, yOffset + 5);
        }
    }

    // 繪製 y 軸刻度 共有numTicks個刻度
    int numTicks = 10;
    int tickInterval = maxHeight / numTicks;

    for (int i = 0; i < numTicks; i++)
    {
        int y = yOffset - i * tickInterval;
        painter.drawLine(xOffset - 5, y, xOffset, y); // 繪製刻度線
        painter.drawText(xOffset - 30, y + 5, QString::number(static_cast<int>(i *
maxCount / numTicks)));
    }
}
```

繪製灰階影像:

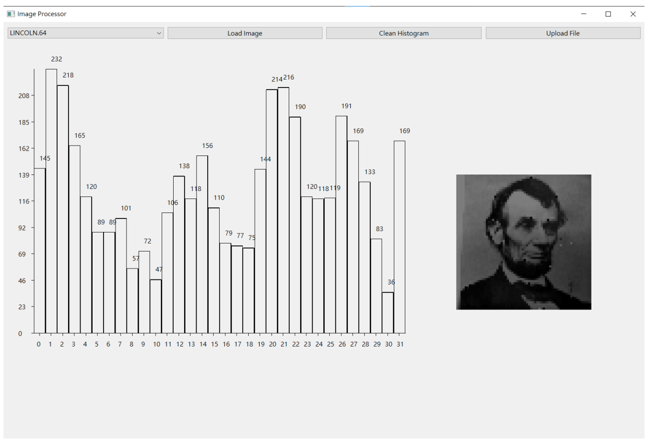
```cpp
void ImageProcessor::displayGrayImage()
{
    // 創建QImage以儲存灰階影像
    QImage image(64, 64, QImage::Format_Grayscale8);

    for (int row = 0; row < 64; row++)
    {
        for (int col = 0; col < 64; col++)
        {
            int value = imageArray[row][col] * 4;              // 從 imageArray 取得值
            image.setPixel(col, row, qRgb(value, value, value)); // 設置為灰階顏色
        }
    }

    // // 將影像儲存到檔案
    // image.save("gray_image.png");

    // 更新窗口以顯示影像
    QPixmap pixmap = QPixmap::fromImage(image);
    QLabel *imageLabel = new QLabel(this);
    imageLabel->setPixmap(pixmap.scaled(256, 256)); // 顯示 256x256 大小的影像
    imageLabel->setGeometry(850, 250, 256, 256);    // 設置影像位置(左上角x,y)和大小(w,h) 須符合圖片大小
    imageLabel->show();
}
```

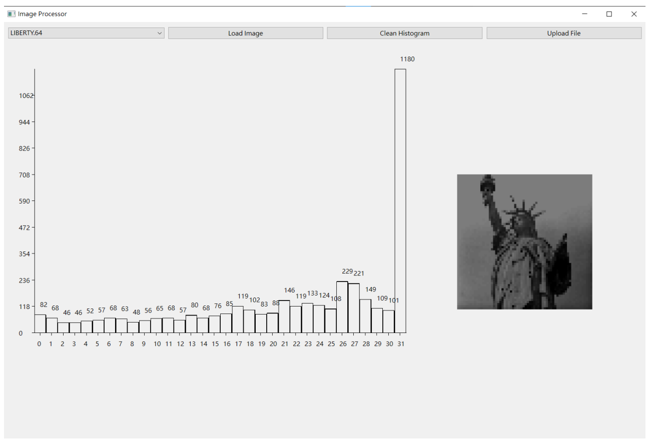
【結果圖片】





LISA.64直方圖與灰階圖	JET.64直方圖與灰階圖





LINCOLN.64直方圖與灰階圖 LIBERTY.64直方圖與灰階

【結果討論】

我設計了四個按鈕分別為:選擇預設圖片清單、載入影像、清除直方圖、上傳其他檔案。當選擇清單以後需要點選載入影像才會顯示灰階圖和直方圖。

Part 2. Arithmetic Operations of an Image Array

Design a software program that will perform the basic tasks of arithmetic operations on an image or two images.
Use the .64 image for this program.
The assigned image processing operations are as follows:

1. Add or subtract a constant value to each pixel in the image.
2. Multiply a constant to each pixel in the image.
3. Create a new image which is the average image of two input images.
4. Create a new image g(x,y) in which the value of each pixel is determined by calculating the pixel values of the input image f(x,y) using the following equation: $$ g(x,y) = f1(x,y) - f2(x-1,y) $$

【解釋算法】

線性運算:

```
void SecondProcess::p_calculateImage(int addend, double multiplier)
{
    // 計算影像的加法和乘法
    // 最後記得檢查數值會不會超過 255 或是小於 0
    for (int i = 0; i < imageArray1.size(); i++)
    {
        for (int j = 0; j < imageArray1[i].size(); j++)
        {
            // 先乘後加
            imageArrayAns[i][j] = static_cast<int>(std::round(imageArray1[i][j] *
multiplier));
            imageArrayAns[i][j] = imageArrayAns[i][j] + addend;
            // 將結果限制在 [0, 255] 範圍內
            if (imageArrayAns[i][j] < 0)
                imageArrayAns[i][j] = 0;
            if (imageArrayAns[i][j] > 255)
                imageArrayAns[i][j] = 255;
        }
    }
}
```

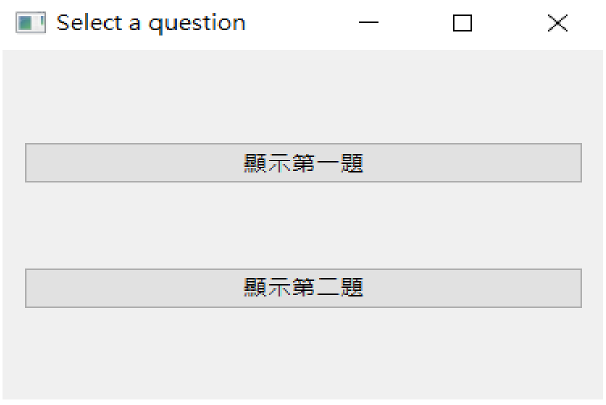
平均兩圖:

```
void SecondProcess::p_averageImage()
{
    // 檢查一定要載入兩個影像
    if (image2Path == "None")
    {
        QMessageBox::critical(this, "Error", "Please load image2 first");
        return;
    }

    // 計算兩個影像的平均值
    // 這裡只是一個示例，實際上需要根據您的需求來實現
    for (int i = 0; i < imageArray1.size(); i++)
    {
        for (int j = 0; j < imageArray1[i].size(); j++)
        {
            imageArrayAns[i][j] = (imageArray1[i][j] + imageArray2[i][j]) / 2;
        }
    }
}
```

g(x,y) = f1(x,y) - f2(x-1,y):

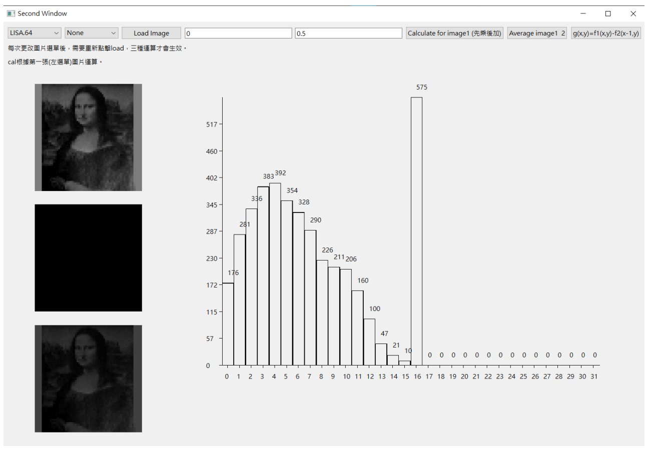
```
void SecondProcess::p_gxImage()
{
    // 檢查一定要載入兩個影像
    if (image2Path == "None")
    {
        QMessageBox::critical(this, "Error", "Please load image2 first");
        return;
    }

    // 計算兩個影像的 g(x,y)=f1(x,y)-f2(x-1,y)
    for (int i = 0; i < imageArray1.size(); i++)
    {
        for (int j = 0; j < imageArray1[i].size(); j++)
        {
            if (i > 0) // 避免 i - 1 為負數
            {
                imageArrayAns[i][j] = imageArray1[i][j] - imageArray2[i - 1][j];
            }
            else
            {
                imageArrayAns[i][j] = imageArray1[i][j]; // i == 0 時，僅使用 imageArray1
            }

            // 將結果限制在 [0, 255] 範圍內
            if (imageArrayAns[i][j] < 0)
                imageArrayAns[i][j] = 0;
            if (imageArrayAns[i][j] > 255)
                imageArrayAns[i][j] = 255;
        }
    }
}
```

【結果圖片】



題目選單

【線性運算】

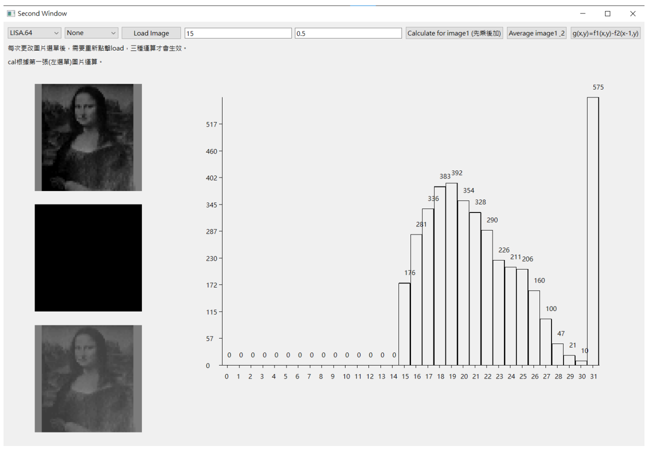


Second Window

LISA.64 | None | Load Image | 0 | 0.5 | Calculate for image1 (先乘後加) | Average image1 2 | g(x,y)=f1(x,y)-f2(x-1,y)

每次更改圖片選單後，需要重新點擊load，三種運算才會生效。

cal根據第一張(左選單)圖片運算。

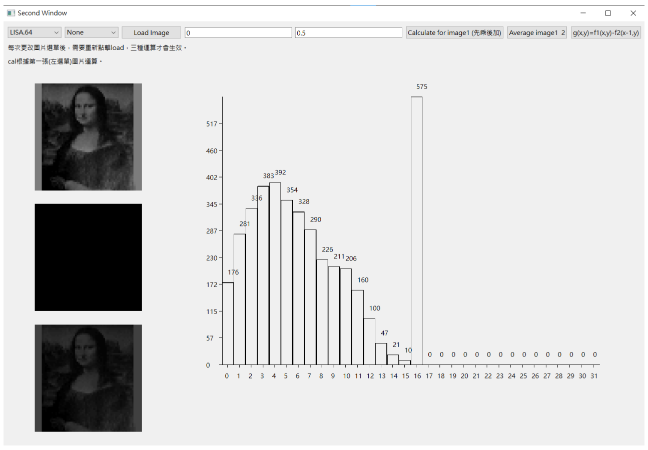


Second Window

LISA.64 | None | Load Image | 15 | 0.5 | Calculate for image1 (先乘後加) | Average image1 2 | g(x,y)=f1(x,y)-f2(x-1,y)

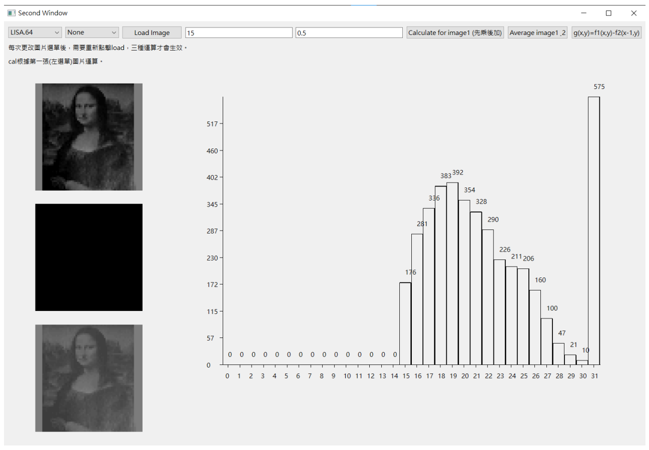
每次更改圖片選單後，需要重新點擊load，三種運算才會生效。

cal根據第一張(左選單)圖片運算。


LISA.64 | None | 0 | 0.5

每次更改圖片選單後，需要重新點擊load，三種運算才會生效。

cal根據第一張(左選單)圖片運算。

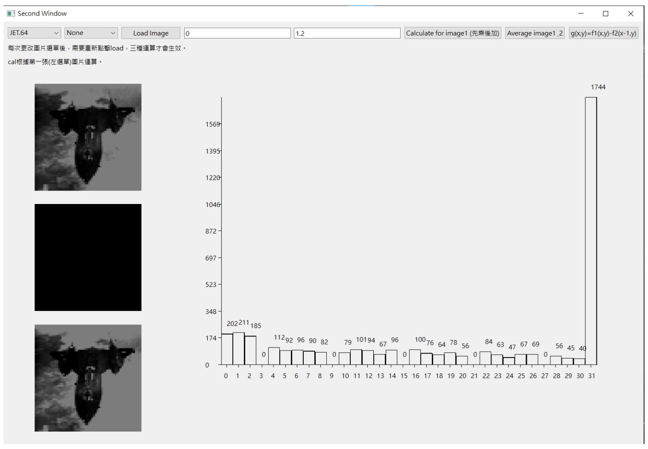

LISA.64 | None | 15 | 0.5

每次更改圖片選單後，需要重新點擊load，三種運算才會生效。

cal根據第一張(左選單)圖片運算。

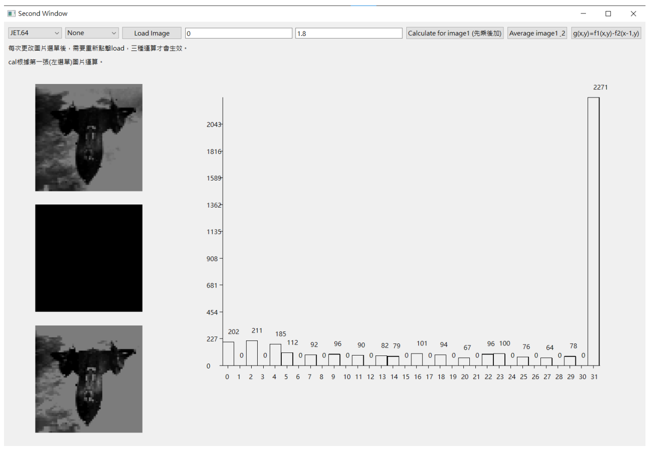
LISA * 0.5 + 0

LISA * 0.5 + 16


JET.64 | None | 0 | 1.2

每次更改圖片選單後，需要重新點擊load，三種運算才會生效。

cal根據第一張(左選單)圖片運算。


JET.64 | None | 0 | 1.8

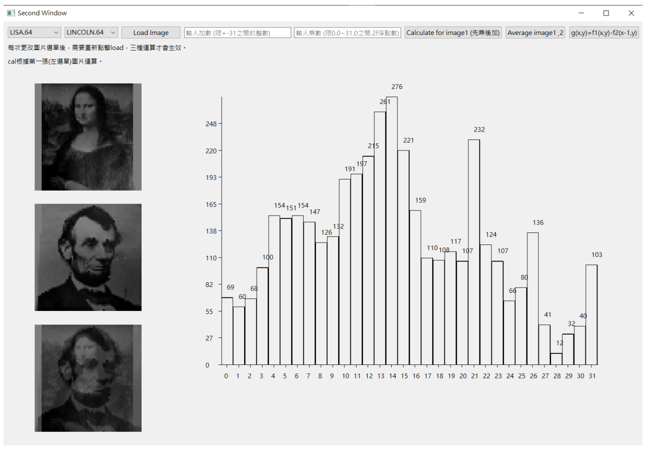
每次更改圖片選單後，需要重新點擊load，三種運算才會生效。

cal根據第一張(左選單)圖片運算。

JET * 1.2 + 0

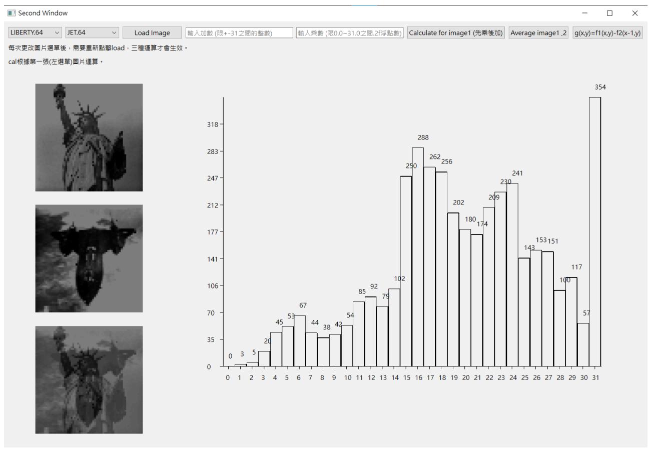
LISA * 1.8 + 0

【平均圖片】


LISA.64 | LINCOLN.64 | 輸入加數 (限+-31之間的整數) | 輸入乘數 (限0.0~31.0之間.2f浮點數)

每次更改圖片選單後，需要重新點擊load，三種運算才會生效。

cal根據第一張(左選單)圖片運算。


LIBERTY.64 | JET.64 | 輸入加數 (限+-31之間的整數) | 輸入乘數 (限0.0~31.0之間.2f浮點數)

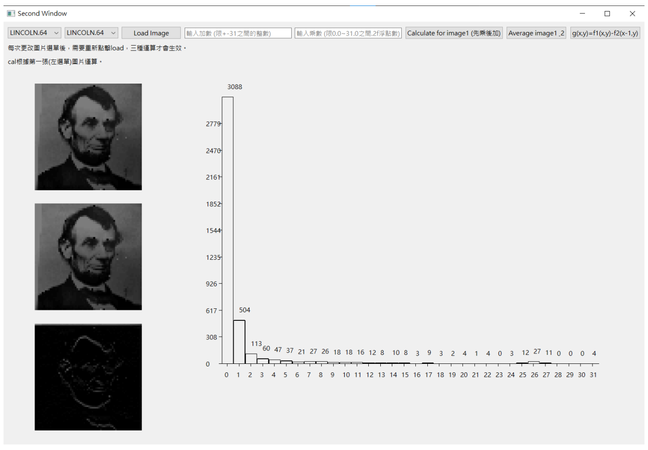
每次更改圖片選單後，需要重新點擊load，三種運算才會生效。

cal根據第一張(左選單)圖片運算。

avg(LISA, LINCOLN)

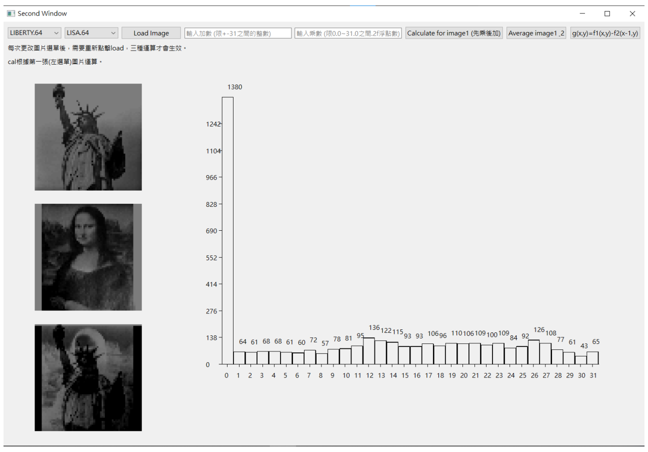
avg(LIBERTY, JET)

【$g(x,y)=f1(x,y)-f2(x-1,y)$】



每次更改圖片選單後，需要重新點擊load，三種運算才會生效。

cal根據第一張(左選單)圖片運算。



每次更改圖片選單後，需要重新點擊load，三種運算才會生效。

cal根據第一張(左選單)圖片運算。

g(LINCOLN, LINCOLN) g(LIBERTY, LISA)

【結果討論】

1. 線性運算:
   加法運算相當於調整影像亮度、乘法運算相當於調整影像對比度。
2. 平均圖片:
   平均圖片相當於將兩張所選圖片/2以後相加，因此可以看到兩張圖片被刷淡後的輪廓。
3. f1(x,y)-f2(x-1,y):
   因為兩張圖片僅差在x差了一格pixel，因此如果選擇同張圖片相減，顯示的圖片會近似於x方向邊緣檢測的效果。
   如果選擇不同圖片的話，f2會產生有點像是負片的效果。