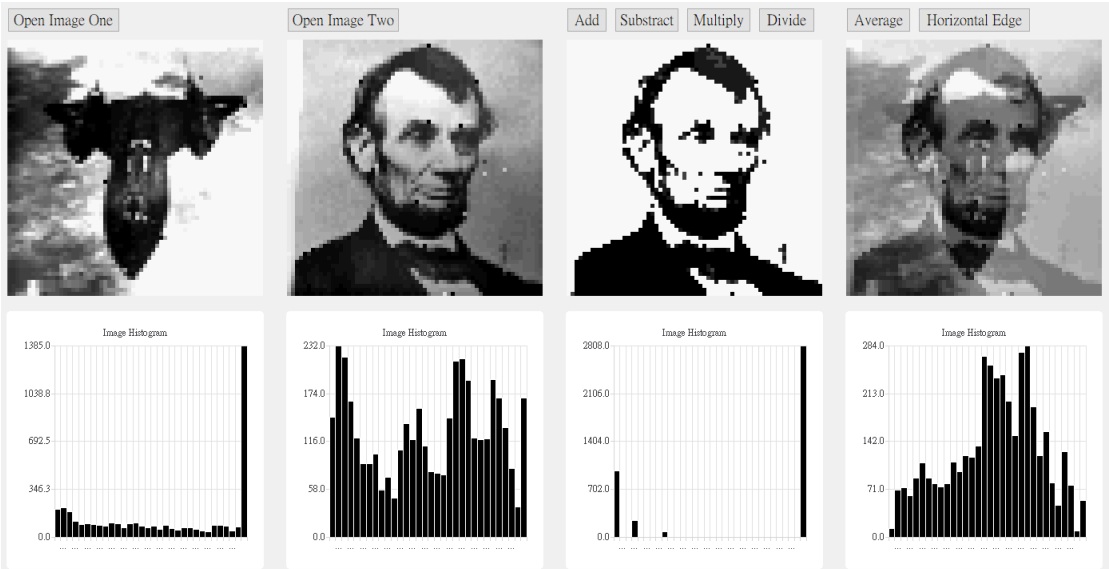
Digital Image Processing HW01 Report

生機碩一 R09631007 吳乙澤



一、演算法說明

本作業有兩個部分，分別為「Histogram of an Image」以及「Arithmetic Operations of an Image Array」。以下將說明這兩個部分的演算法。

1-1 Histogram of an Image

首先，我以讀文字檔的方式，讀取 .64 圖片中的資料。然後利用 for 迴圈，取出字串中的每個字元。再以 cast 方法，將這些字元轉成數字，並根據Ascii，使資料轉成 0~31 的形式。轉換完成後，以 for 配合 if ，將 0~31 像素的數量記錄在 pixelCount 這個一維矩陣中。最後，為了使色差明顯一些，先把轉換完的資料乘以 8 後，才放入圖片裡。如此一來，像素最小為0、最大為248。

```
// Use text file to open .64 images
int i = 0;
while(!input.atEnd())
{
    string line = input.readLine().toStdString();
    for(int j = 0 ; j < 64 ; j++)
    {
```

```
            // Cast char to int
            int pixel = (int)line[j];
            // Ascii
            if (pixel <= '9')
                pixel -= '0';
            else
                (pixel -= 'A') += 10;
            // Count 0~31 pixel sum (Histogram)
            for (int k = 0 ; k < 32 ; k++)
            {
                if (k == pixel)
                {
                   pixelCount[k] += 1;
                   break;
                }
            }
            img01.setPixel(j, i, qRgb(pixel*8, pixel*8, pixel*8));
        }
        i++;
    }
```

製作直方圖的部分，我以 QChart 來實現。首先，用 for 迴圈將 pixelCount 中數量最多的像素值找出來，再利用 QValueAxis 的 setRange 方法，調整 y 軸的範圍使直方圖顯示地更為漂亮。之後，再調整直方圖的粗細、圖標的顯示與否，便能繪出漂亮的直方圖。

```
// Show Histogram
QBarSet *set = new QBarSet("Image Histogram");
int yMax = 0;
for (int i = 0 ; i < 32 ; i++)
{
    set->append(pixelCount[i]);
    if(yMax < pixelCount[i])
        yMax = pixelCount[i];
    set->setColor(QColor::Rgb);
}
QValueAxis *axisY = new QValueAxis; // Set y axis
axisY->setLabelFormat("%d");
```

```
axisY->setRange(0,yMax);
QBarSeries *series = new QBarSeries();
series->append(set);
series->setBarWidth(1);

QChart *chart = new QChart();
chart->addSeries(series);
chart->createDefaultAxes();
chart->setTitle("Image Histogram");
chart->setAnimationOptions(QChart::SeriesAnimations);
chart->legend()->setVisible(false);

ui->showHis01->setChart(chart);
```

1-2 Arithmetic Operations of an Image Array

對圖片做加減乘除。我利用 +=、-=、*=、/= 等方法，改變像素值後，以自己寫的簡單函數**checkPixel**，來檢查像素是否超過 31 或低於 0。超過 31 使像素值為 31 ，低於 0 則使之為 0。如此一來，顯示圖片時才不會有異常發生。

```
for (int i = 0 ; i < 64 ; i++)
{
    for (int j = 0 ; j < 64 ; j++)
    {
        img02Array[i][j] += 1; // Add one
        int pixel = img02Array[i][j];
        pixel = checkPixel(pixel);
        imgAfter.setPixel(j, i, qRgb(pixel*8, pixel*8, pixel*8));
    }
}

int MainWindow::checkPixel(int pixel)
{
    if (pixel > 31)
        pixel = 31;
    else if (pixel < 0)
        pixel = 0;
    return pixel;
}
```

平均影像及 $g(x,y) = f(x,y) - f(x-1,y)$ 的程式碼如下所示。平均即是把影像一跟影像二的像素值相加除以二。$g(x,y) = f(x,y) - f(x-1,y)$ 則是將影像矩陣中，除第一排以外的 63 排像素，都扣掉每排左方那排的像素值。如此一來，若兩排像素值差異不大，經過該運算後會偏向黑色；若兩排像素值差異很大，經過該運算後會偏向白色。總的來說，$g(x,y) = f(x,y) - f(x-1,y)$ 能大致找出圖片水平方向的邊緣。

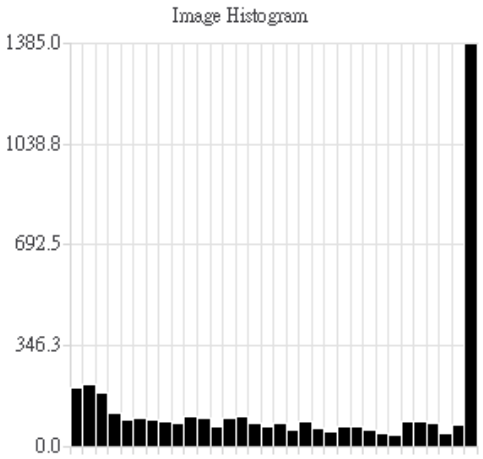
```
int pixel = (checkPixel(img01Array[i][j])+checkPixel(img02Array
[i][j]))/2;
```

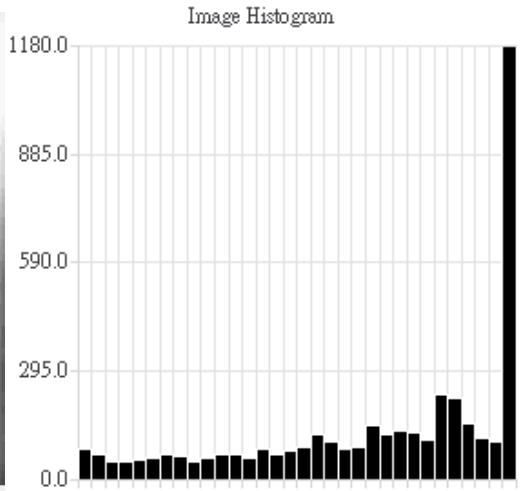
```
int pixel = checkPixel(img02Array[i][j])-checkPixel(img02Array
[i-1][j]);
```

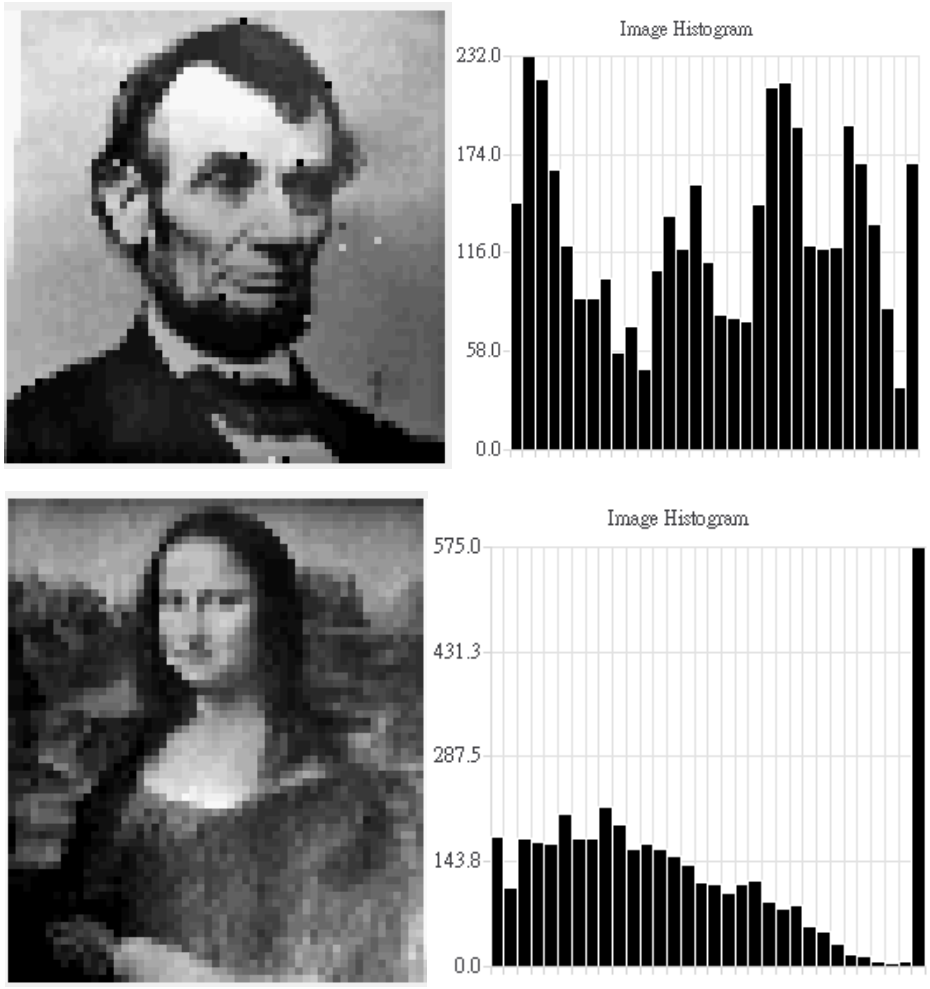
二、結果與討論

結果與討論分為兩個部分，「Histogram of an Image」及「Arithmetic Operations of an Image Array」。該章將顯示經過第一章演算法跑出的圖片，並做相關討論。

2-1 Histogram of an Image





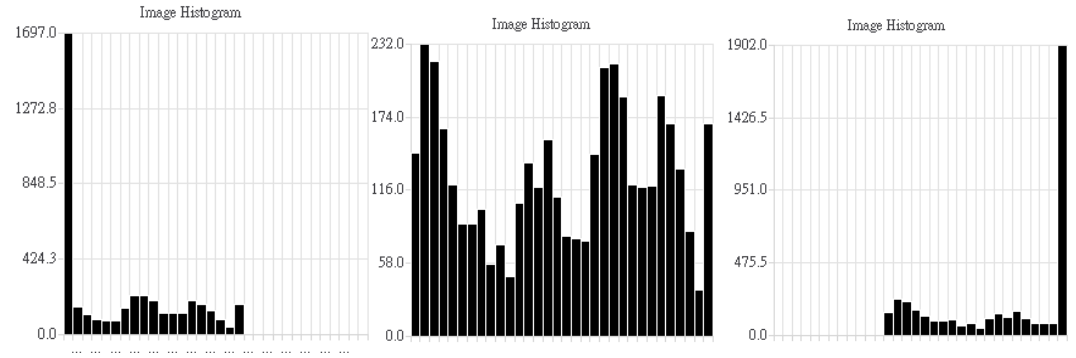


這些直方圖的 X 軸是灰階，有0~31共32種灰階變化。Y 軸則是這些灰階在該圖片中的總量。使用直方圖可顯現圖片的像素分布。

2-2 Arithmetic Operations of an Image Array

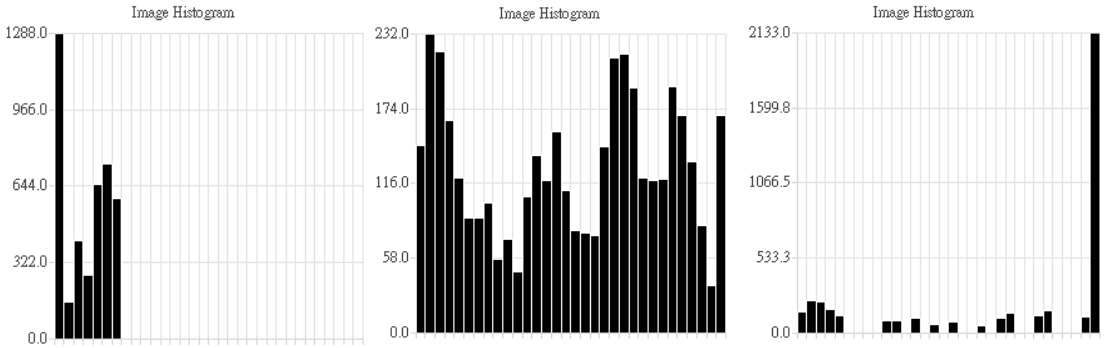
以下是 LINCOLN.64 經過加法和減法後的圖片。可以看出最右邊的圖片最亮，這是經過多次加法後的結果。最左邊的圖片則是經過多次減法後的結果。由此，可看出加減法會影響圖片的明暗度。

以下為前三張圖片的直方圖。整體看起來為亮的圖片，直方圖中的像素分布大多在右方。反之則偏向左方。



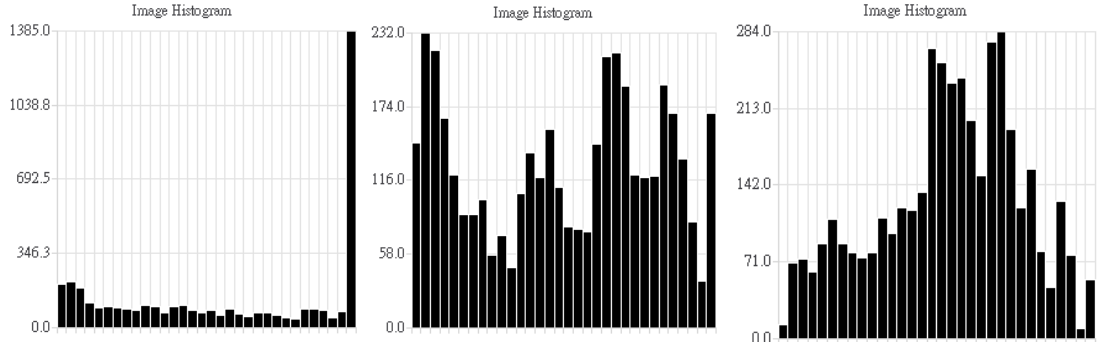
以下為 LINCOLN.64 經過乘法和除法後的圖片。可以看出最右邊的圖片最亮，像素間的對比也最大，這是經過多次乘法後的結果。最左邊的圖片則是經過多次除法後的結果。由此，可看出乘除法會影響圖片的亮度和對比度。

以下為前三張圖片的直方圖。整體看起來對比度大的圖片，直方圖中的像素分布較鬆散。反之則像素分布較緊密。



以下最右邊的圖片是 JET.64 和 LINCOLN.64 兩張圖片像素值相加除以二的結果，可以看到該圖片偏亮、對比度偏低，同時有 JET.64 和 LINCOLN.64，並有殘影效果。

以下為前三張圖片的直方圖。最右邊的圖片為 JET.64 和 LINCOLN.64 相加除以二的直方圖。可以觀察到直方圖的分布偏向右方，且分布較為緊密，因此符合偏亮、對比度偏低的描述。



以下中間的圖片是 LINCOLN.64 經過 g(x,y) = f(x,y) - f(x-1,y) 運算後的結果。如在1-2節所說，該運算能大致找出圖片水平方向的邊緣。見最右邊的直方圖，偏左且像素值為零的數量相當多，有3000多個，故影像整體是相當暗的。