數位影像處理第二次作業

自動碩一 111618018 吳祐毅

NTUT, Institute of Automation Technology

Digital Image Processing

Instructor： Chin-Sheng Chen (陳金聖)

Homework Assignment #2 Due date: 2022/11/01

1. (15%) Do the following:

(1) Propose a method fo rextracting the bit planes of an image based on converting the value of its pixels to binary.

(2) Find all th bit palnes of the following 4-bit image.

0	1	8	6
2	2	1	1
1	15	14	12
3	6	9	10

(1)

0000	0001	1000	0110
0010	0010	0001	0001
0001	1111	1110	1100
0011	0110	1001	1010

(2)

0:

0	1	0	0
0	0	1	1
1	1	0	0
1	0	1	0

1:

0	0	0	2
2	2	0	0
0	2	2	0
2	2	0	2

2:

0	0	0	4
0	0	0	0
0	4	4	4
0	4	0	0

3:

0	0	8	0
0	0	0	0
0	8	8	8
0	0	8	8

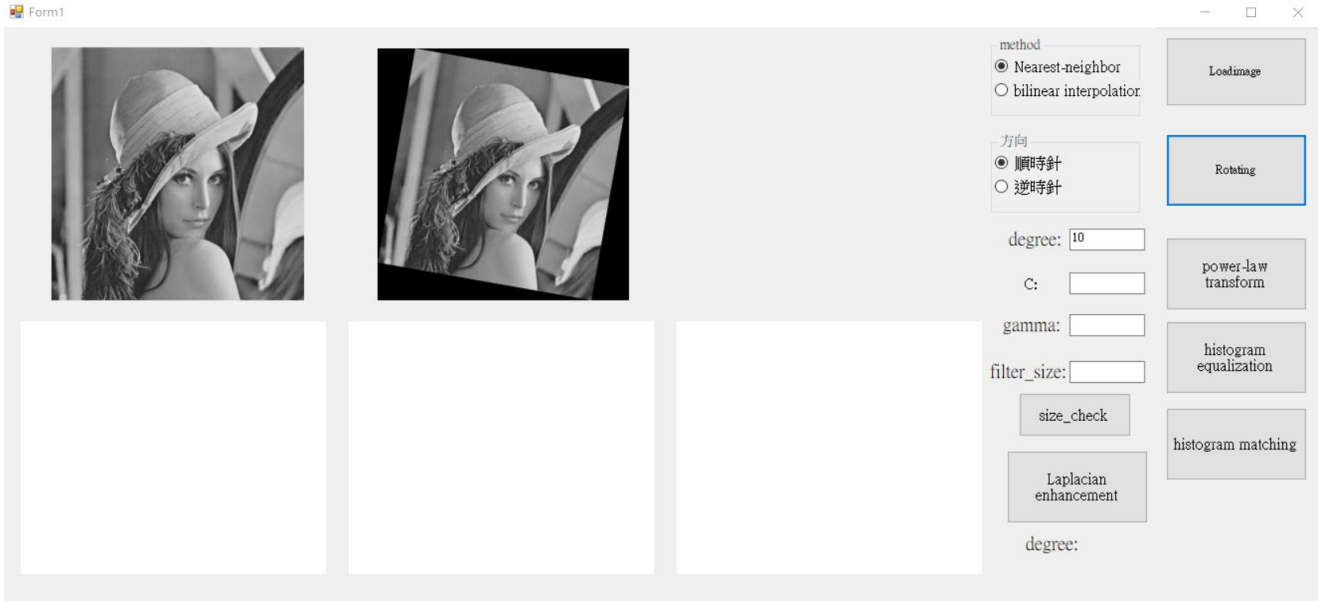
2. (20%) **Rotating Images by Nearest-neighbor and bilinear interpolation**

(1) Write a computer program capable of rotating an image. The desired rotated factors are needed to be variables that can be input into your program.

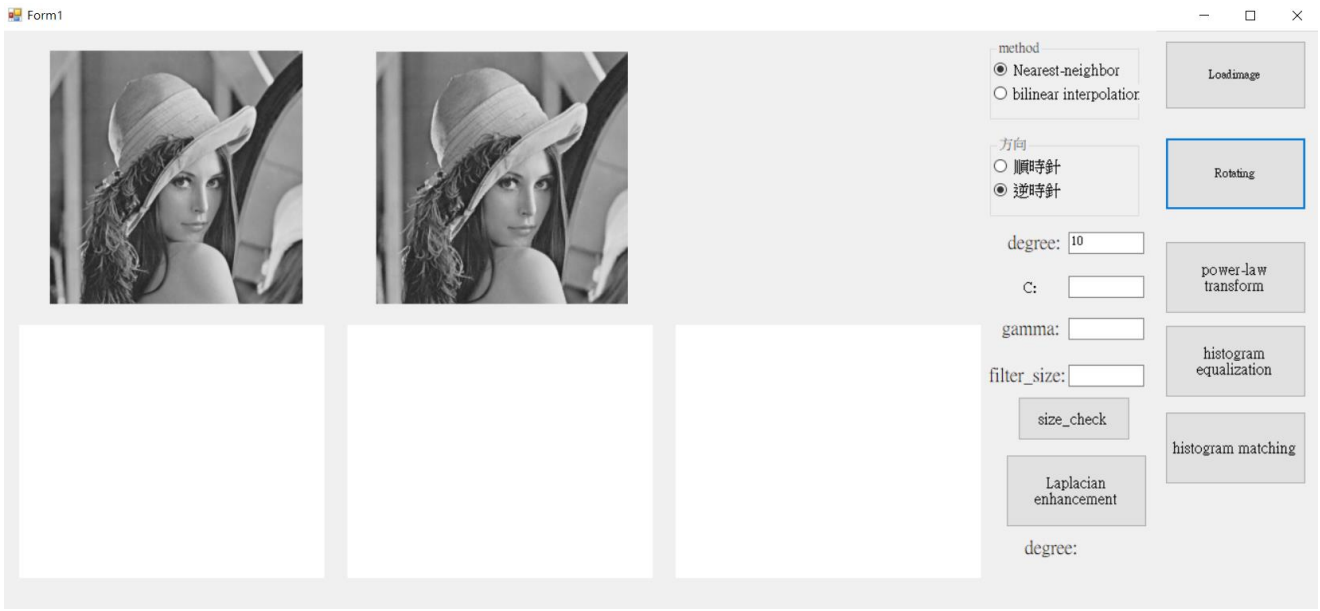
(2) Rotate “Lena image” and your own image 10 degrees clockwise, then rotate the image from above result back to the original. Explain the reasons for their differences.

介面設計

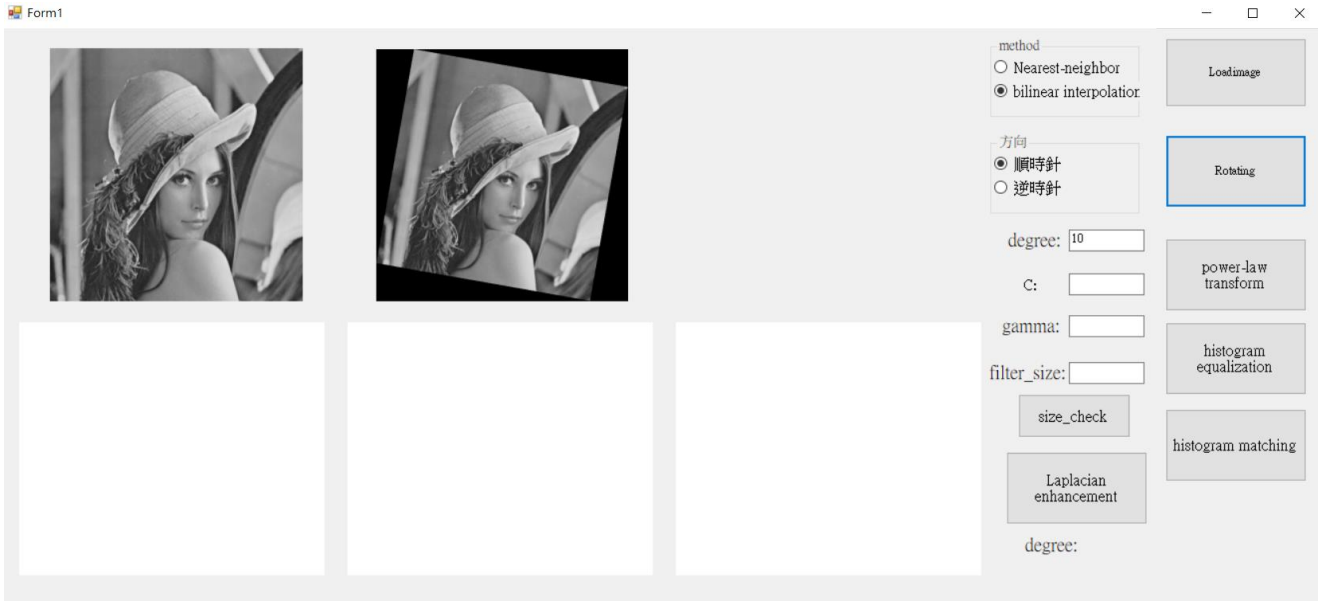
1. Nearest-neighbor 順時針旋轉 10 度



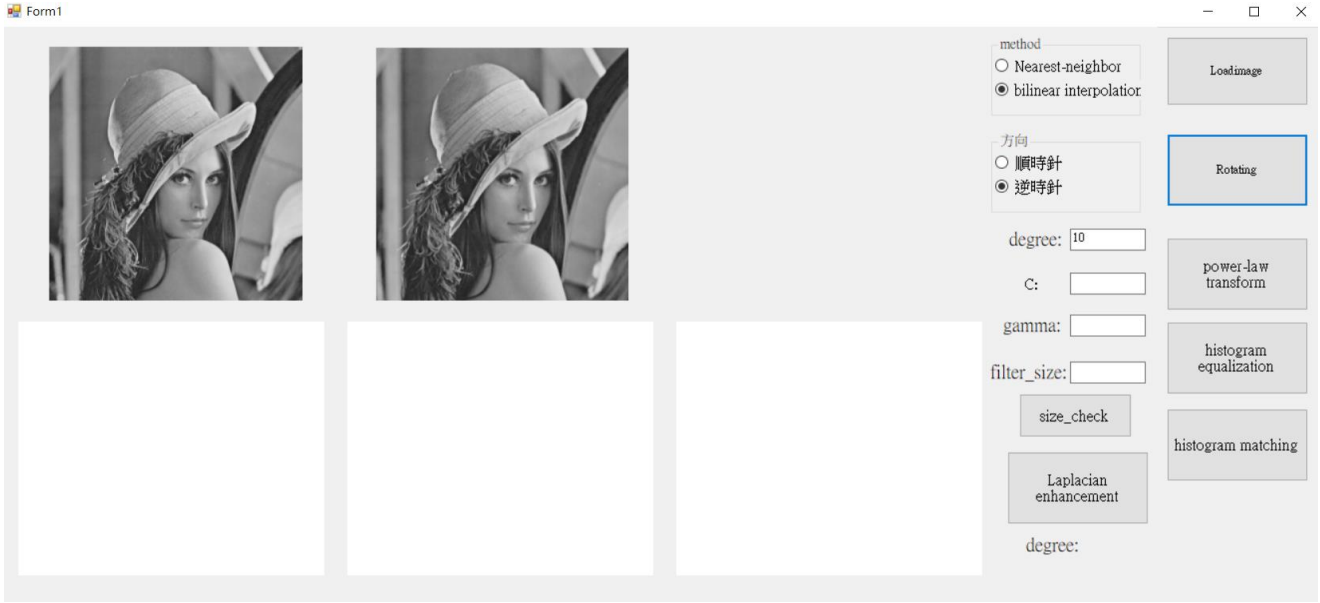
2. Nearest-neighbor 逆時針旋轉 10 度



3. **bilinear interpolation** 順時針旋轉 10 度



4. **bilinear interpolation** 逆時針旋轉 10 度



透過旋轉矩陣和平移矩陣去求得影像中每一個像素旋轉之後的位置:

逆時針旋轉平移矩陣:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & \mathcal{X} \\ \sin(\theta) & \cos(\theta) & \mathcal{Y} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \mathbb{X} \\ \mathbb{Y} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbb{X}' \\ \mathbb{Y}' \\ 1 \end{bmatrix}$$

順時針旋轉平移矩陣:

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) & \mathcal{X} \\ -\sin(\theta) & \cos(\theta) & \mathcal{Y} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \mathbb{X} \\ \mathbb{Y} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbb{X}' \\ \mathbb{Y}' \\ 1 \end{bmatrix}$$

由於 **Nearest-neighbor and bilinear interpolation** 需要考慮原本影像附近的像素值，所以需要由轉移後的圖像去求轉移前的圖像位置，需要先對旋轉平移矩陣求反矩陣。

逆時針旋轉平移矩陣的反矩陣:

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) & -\mathcal{Y}sin(\theta) - \mathcal{X}\cos(\theta) \\ -\sin(\theta) & \cos(\theta) & -\mathcal{Y}\cos(\theta) + \mathcal{X}\sin(\theta) \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \mathbb{X}' \\ \mathbb{Y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbb{X} \\ \mathbb{Y} \\ 1 \end{bmatrix}$$

順時針旋轉平移矩陣的反矩陣:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & \mathcal{Y}sin(\theta) - \mathcal{X}\cos(\theta) \\ \sin(\theta) & \cos(\theta) & -\mathcal{Y}\cos(\theta) - \mathcal{X}\sin(\theta) \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \mathbb{X}' \\ \mathbb{Y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbb{X} \\ \mathbb{Y} \\ 1 \end{bmatrix}$$

旋轉平移反矩陣的代碼如下:

```
(rad_cl.Checked == true)

 target_x = Math.Cos(angle) * (i) + (-1) * Math.Sin(angle) * (j)  + (rotat_trans_y * Math.Sin(angle) - rotat_trans_x * Math.Cos(angle)) + trans_x;
 target_y = Math.Sin(angle) * (i) + Math.Cos(angle) * (j) + ((-1) * rotat_trans_y * Math.Cos(angle) - rotat_trans_x * Math.Sin(angle)) + trans_y;


(rad_uncl.Checked == true)

 target_x = Math.Cos(angle) * (i) + Math.Sin(angle) * (j) + ((-1) * rotat_trans_y * Math.Sin(angle) - rotat_trans_x * Math.Cos(angle)) + trans_x;
 target_y = (-1) * Math.Sin(angle) * (i) + Math.Cos(angle) * (j) + ((-1) * rotat_trans_y * Math.Cos(angle) + rotat_trans_x * Math.Sin(angle)) + trans_y;
```

求得原本圖像的位置後，依照選擇 **Nearest-neighbor and bilinear interpolation**，來決定轉換後的位置其像素值，**Nearest-neighbor** 為選擇最近的位置為其像素值，而 **bilinear interpolation** 需要考慮原本圖像的鄰近座標的像素值來決定旋轉過後的像素值為多少。

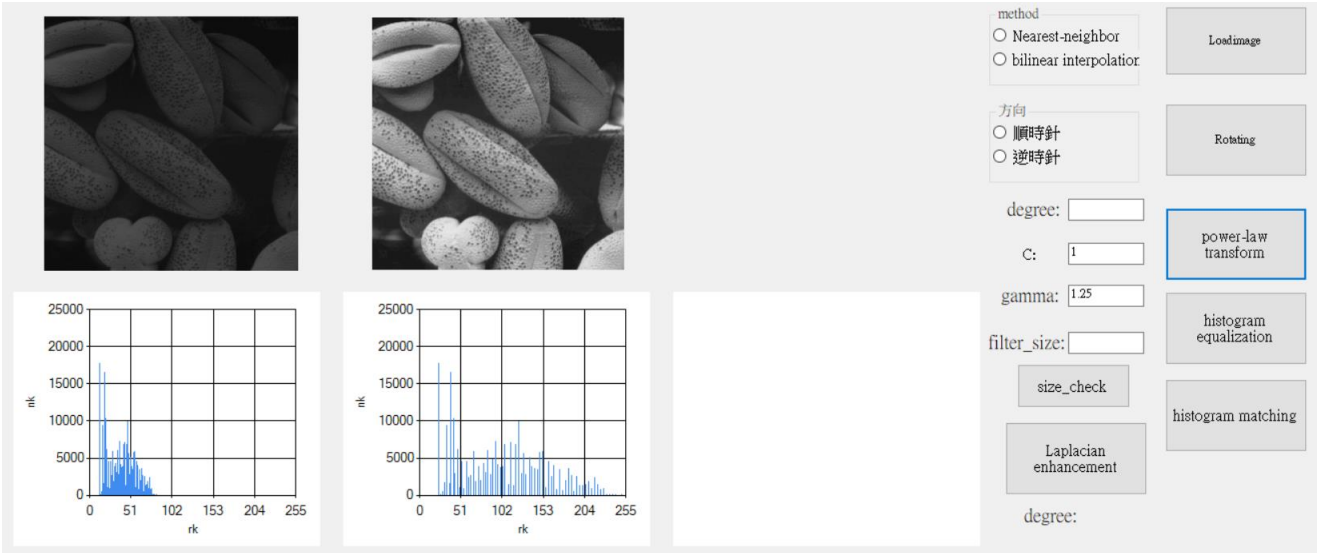
```
if (target_x > 0 && target_x < Width && target_y > 0 && target_y < Height)
{
    if (rad_1.Checked == true) //Nearest-neighbor
    {
        Graymatrix_rotat[i, j] = Graymatrix[(int)Math.Floor(target_x), (int)Math.Floor(target_y)];
    }
}
if (target_x > 0 && target_x < Width - 1 && target_y > 0 && target_y < Height - 1) {
    if (rad_2.Checked == true) //bilinear interpolation
    {
        int a_1, a_2, b_1, b_2;
        double u, v;
        a_1 = (int)Math.Floor(target_x);
        a_2 = (int)Math.Ceiling(target_x);
        b_1 = (int)Math.Floor(target_y);
        b_2 = (int)Math.Ceiling(target_y);
        u = target_x - a_1;
        v = target_y - b_1;
        Graymatrix_rotat[i, j] = (Byte)((1 - u) * (1 - v) * Graymatrix[a_1, b_1] +
        (1 - u) * v * Graymatrix[a_1, b_2] + u * (1 - v) * Graymatrix[a_2, b_1] +
        u * v * Graymatrix[a_2, b_2]);
    }
```

3. (20%) **General Intensity transformation**.

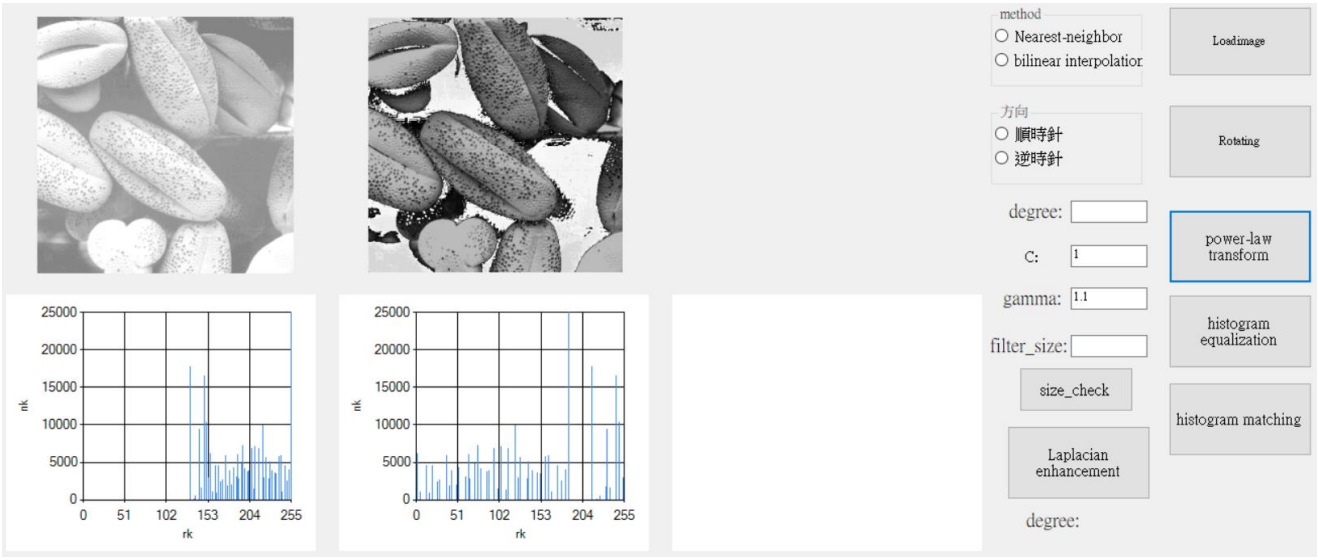
Enhance the images of Figure 3.16, which are four basic image types: dark, light, low contrast, high contrast, respectively. Please write programs with corresponding detail comments in each important codes and give your opinions for the results.

(1) Perform the “power-law transform” to enhance the images. Please adjust the power papameters to output the “best” images and corresponding histograms. Show the parameters, output images and the histograms.

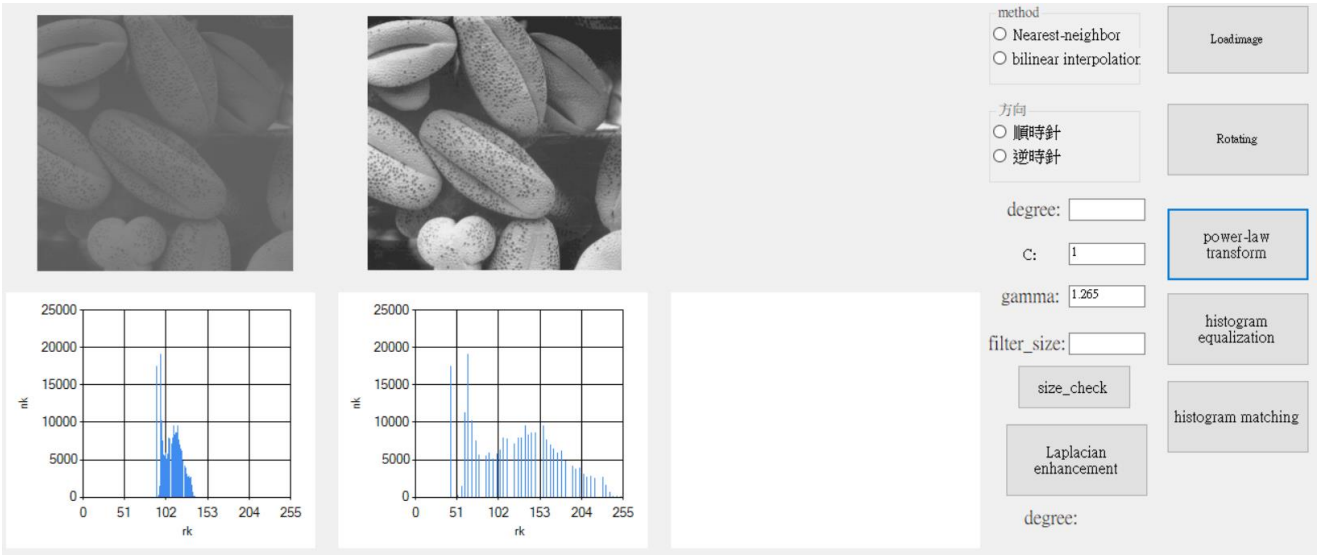
Dark:



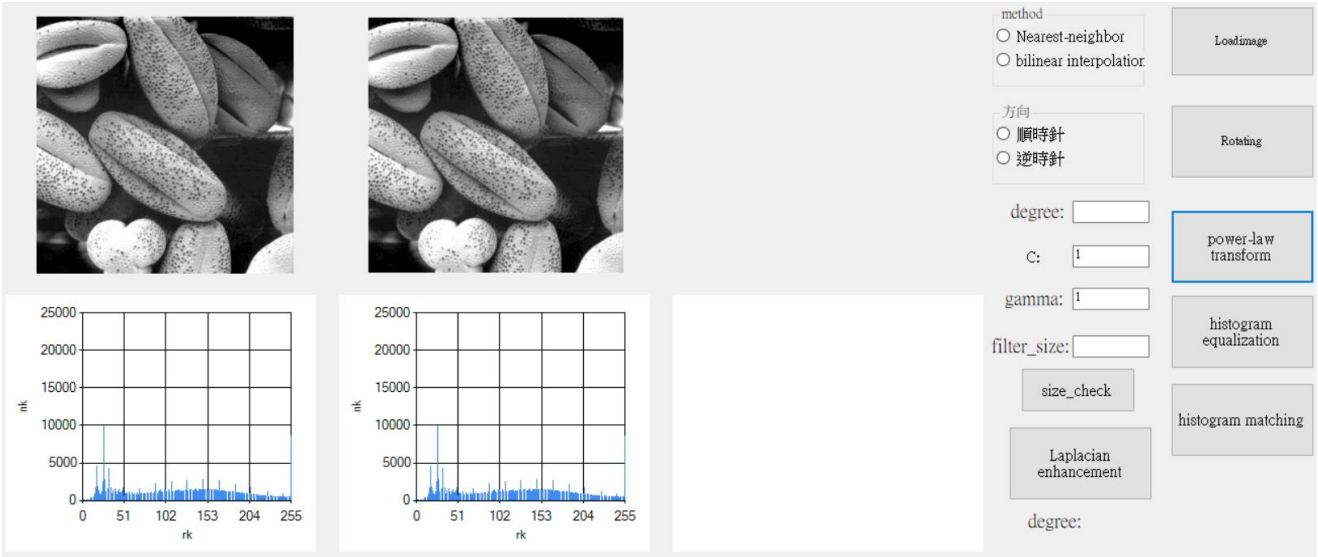
Light:



Low contrast:



High contrast:



增強後的影像 = c * 原始影像 ^ gamma

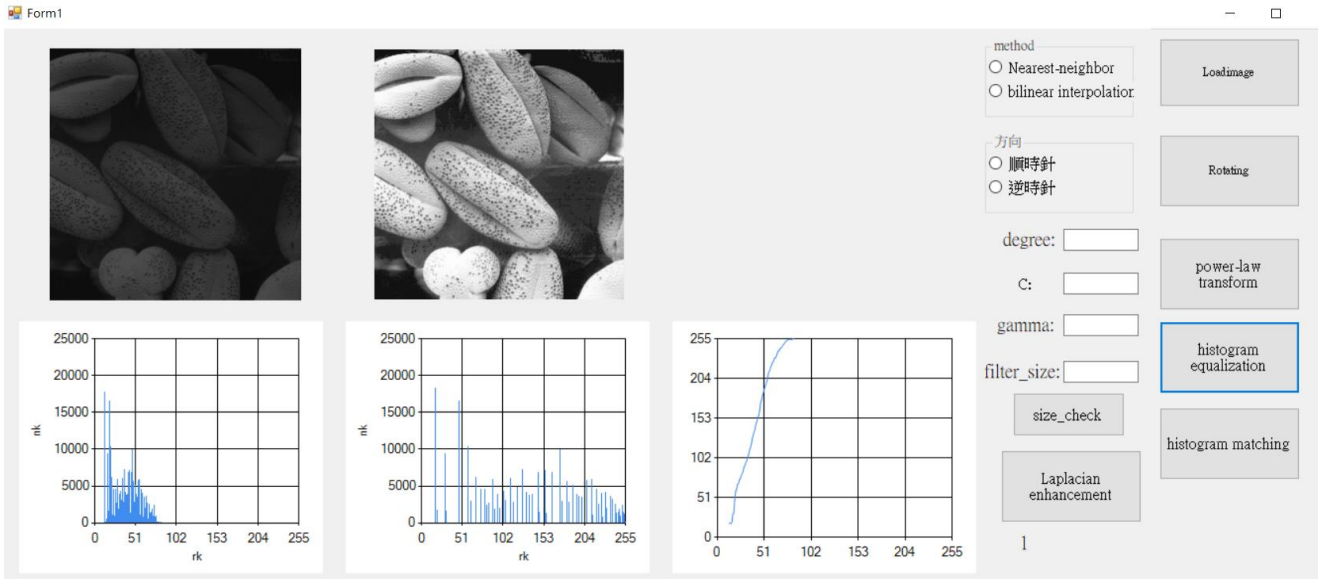
```
// start sliding the matrix
for (int i = 0; i < Height; i++)                // i index for cols ( 0~Hight-1)(Because th
{
    for (int j = 0; j < Width; j++)             // j index for rows ( 0~Width-1)
    {
        Graymatrix_enh[i, j] = (Byte)(c_con * Math.Pow((double)Graymatrix[i, j], gamma));
    }
}
```

比對增強影像前後的灰階值，並且統計前後影像灰階值出現的次數，並繪出直方圖。

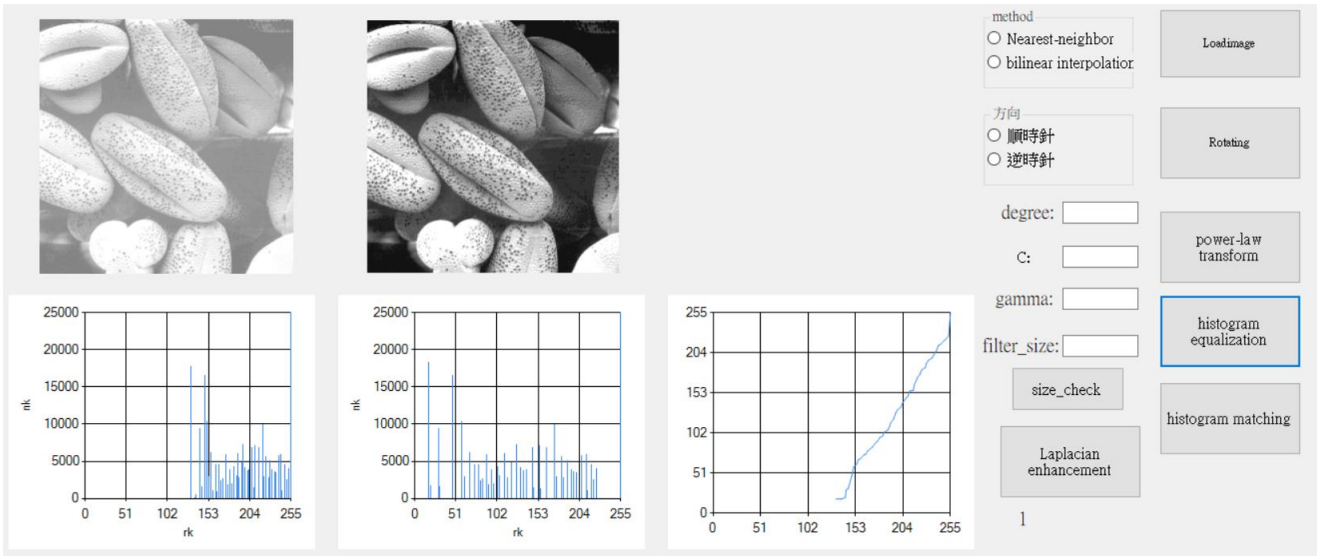
```
int[] gray_level = new int[256];
int[] gray_level_af = new int[256];
for(int i = 0; i < gray_level.Length; i++)
{
    gray_level[i] = i;
}

for (int i = 0; i < Height; i++)
{
    for (int j = 0; j < Width; j++)
    {
        gray_level_af[Graymatrix[i, j]] = Graymatrix_enh[i, j];
    }
}

int[] gray_org_count = new int[256];
int[] gray_af_count = new int[256];
for (int i = 0; i < Height; i++)
{
    for (int j = 0; j < Width; j++)
    {
        gray_org_count[Graymatrix[i, j]] = gray_org_count[Graymatrix[i, j]] + 1;
        gray_af_count[Graymatrix_enh[i, j]] = gray_af_count[Graymatrix_enh[i, j]] + 1;
    }
```

(2) Perform the “histogram equalization” to enhance the images. Show the output images, the curves of the gray-level transform and the histograms.

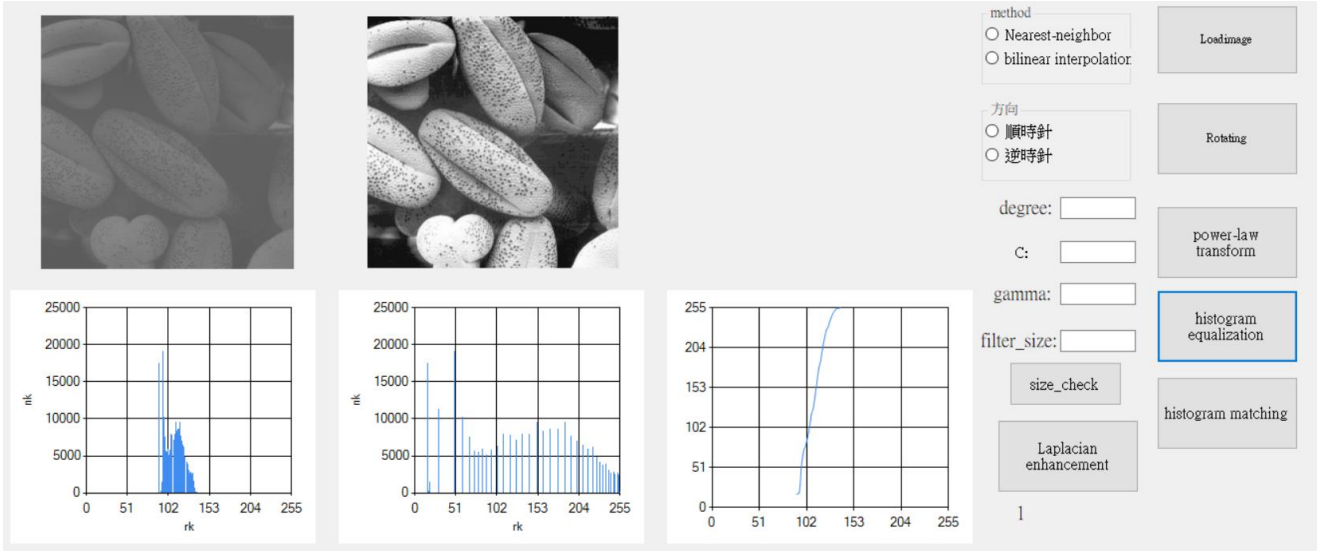
Dark:



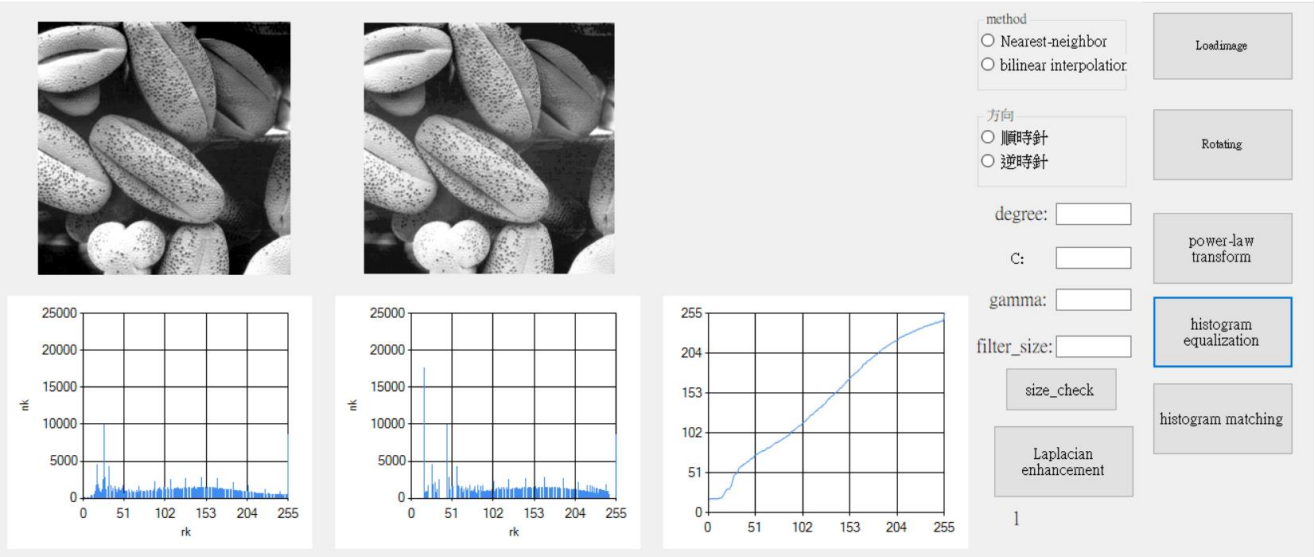
Light:



Low contrast:



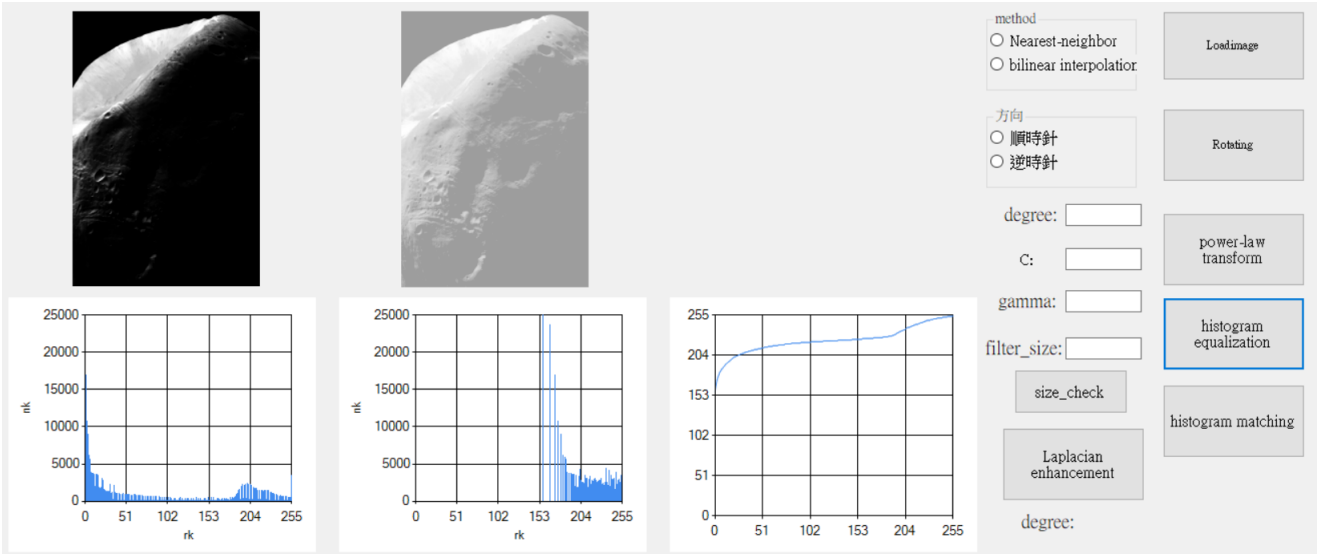
High contrsdt:



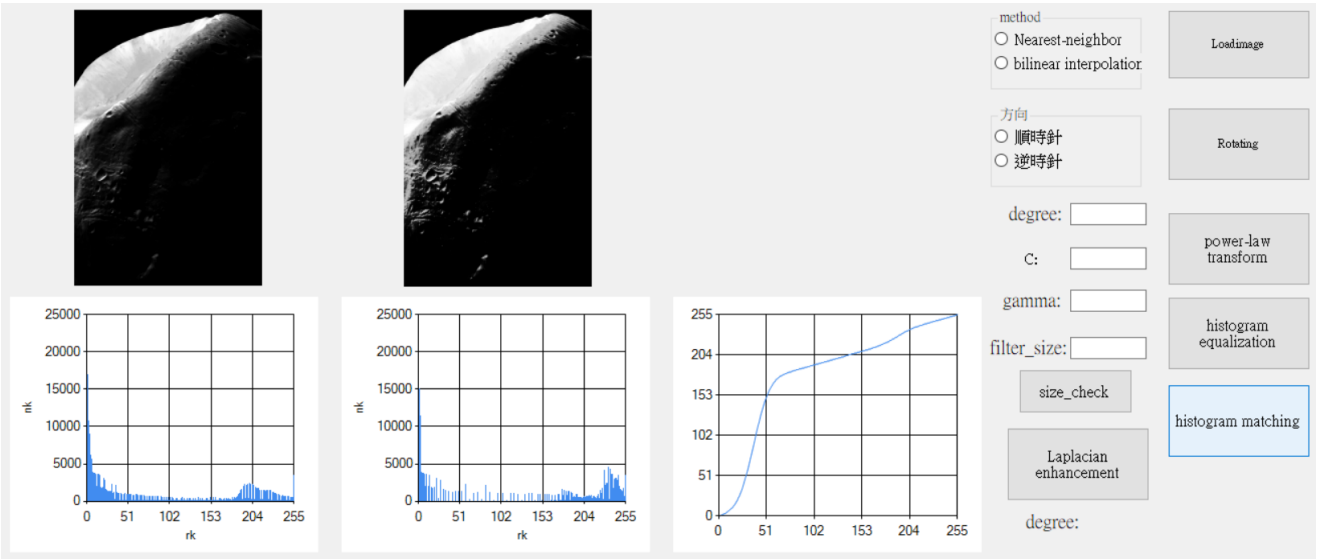

```
for (int i = 0; i < Height; i++)//統計每個灰階值出現次數
{
    for (int j = 0; j < Width; j++)
    {
        gray_org_count[Graymatrix[i, j]] = gray_org_count[Graymatrix[i, j]] + 1;
    }
}
for (int i = 0; i < gray_level.Length; i++)//算出各個灰階值的 PDF
{
    gray_org_pr[i] = gray_org_count[i] / (double)(Width * Height);
}
for(int i = 0; i < 256; i++) //將PDF 做累加求出 CDF
{
    for(int j = 0; j <= i; j++)
    {
        gray_af_pr[i] = gray_af_pr[i] + gray_org_pr[j];
    }
}
for (int i = 0; i < Height; i++)
{
    for (int j = 0; j < Width; j++)
    {
        Graymatrix_his[i, j] = (Byte)(gray_af_pr[Graymatrix[i, j]] * 255);
        gray_level_af[Graymatrix[i, j]] = Graymatrix_his[i, j];
    }
}
```

4. (20%) In Figure 3.23(a), the above “histogram equalization” is not the best enhancement approach. Perform the “histogram matching” to let the histogram of original image transform to as the specified histogram in Figure 3.25(a). Comparing the output performance between “histogram equalization” and “histogram matching”. Show the output images, the curves of the gray-level transformtion and the histograms.

Histogram equalization:



Histogram matching:



由於原始影像的直方圖用波形來表示的話，可以觀察到在暗處的部分和亮處的部分有明顯的波峰，而指定的直方圖除了要滿足在暗處的部分有明顯的對比度之外，其直方圖的分布也要與原始影像相似，所以使用了常態分佈來表示規定直方圖的 pdf。

$$f(x) = \sum_{i=0}^{n} \frac{A_i}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x-\mu_i)^2}{2{\sigma_i}^2}\right)$$

```
double[] pdf = new double[256];
int[] means = new int[2] { 38, 191 };
int[] stds = new int[2] { 13, 13 };
double[] ampl = new double[2] { 0.93, 0.07 };
double bias = 0.002;
```

```
double pdf_total = 0;
for (int j = 0; j < pdf.Length; j++)
{
    for (int i = 0; i < 2; i++)
    {
        pdf[j] += ampl[i] * Math.Exp(-1 * Math.Pow(j - means[i], 2) / (2 * stds[i] * stds[i])) / (stds[i] * Math.Sqrt(2 * Math.PI));
    }
    pdf[j] += bias;
    pdf_total += pdf[j];
}
double[] pdf_nor = new double[256];
for(int i = 0; i < pdf.Length; i++)
{
    pdf_nor[i] = pdf[i] / pdf_total;
}
```

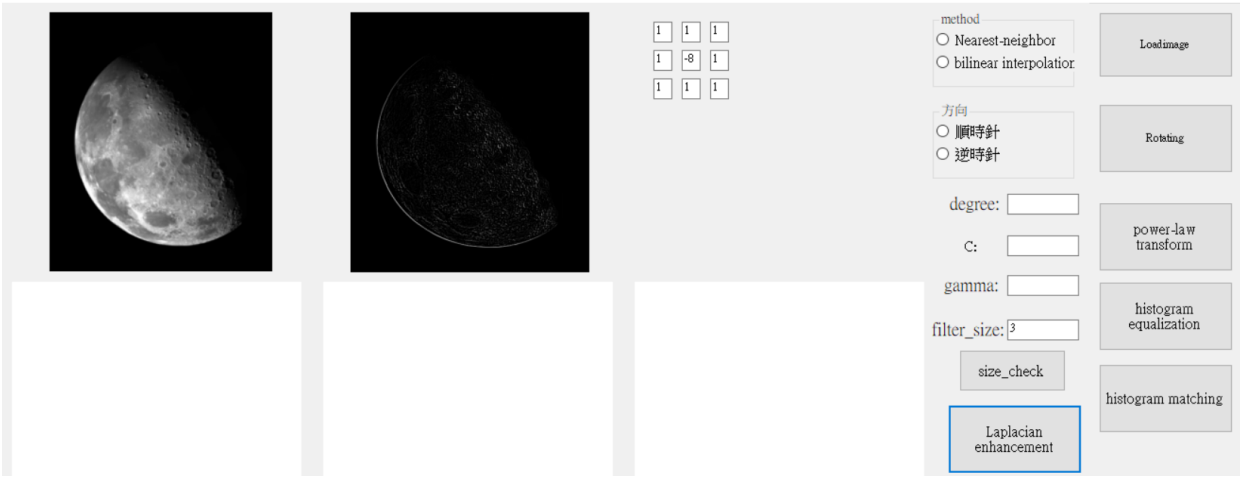
```
for (int i = 0; i < Height; i++)//統計每個灰階值出現次數
{
    for (int j = 0; j < Width; j++)
    {
        gray_org_count[Graymatrix[i, j]] = gray_org_count[Graymatrix[i, j]] + 1;
    }
}

for (int i = 0; i < gray_level.Length; i++)//每個灰階值分布機率
{
    gray_org_pr[i] = gray_org_count[i] / (double)(Width * Height);
}

for (int i = 0; i < 256; i++)
{
    for (int j = 0; j <= i; j++)
    {
        gray_af_pr[i] = gray_af_pr[i] + pdf_nor[j];
    }
}

for (int i = 0; i < Height; i++)
{
    for (int j = 0; j < Width; j++)
    {
        Graymatrix_mat[i, j] = (Byte)(gray_af_pr[Graymatrix[i, j]] * 255);
        gray_level_af[Graymatrix[i, j]] = Graymatrix_mat[i, j];
    }
```

5. (25%) **Spatial Filtering**.

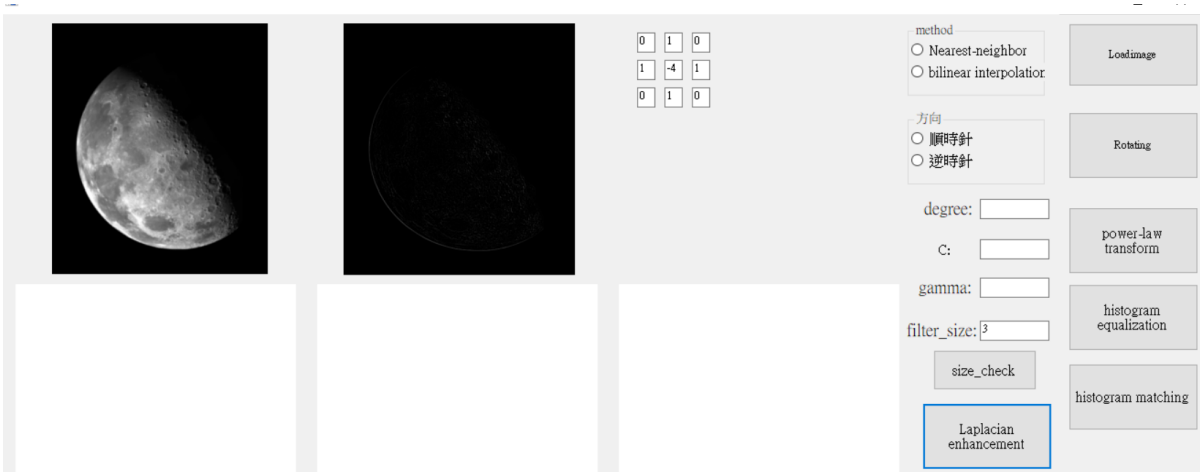
(1) .Write program to perform spatial filtering of an image (see Section 3.4 regarding implementation). In general, you can flexibly change the size of the spatial mask, and the coefficients need to be variables that can be input into your program.

(2) Use the programs developed in (1) to implement the Laplacian enhancement technique, where the filter mask could be any one in Fig. 3.45. The Laplacian with -8 in the center yields shaper results than the one with a -4 in the center. Explain the reason in detail, and prove your results by the output images.

-8 in the center:



-4 in the center:



原因探討:

第一張圖月球周圍的輪廓較第二張清楚，原因為拉普拉斯是倒數運算子，所以在凸顯影像中變化較大的區域，如月球周圍與黑色接觸的交接處會較為明顯，並且降低強度準位緩慢變化的區域，而 filter 中心為 -8 比中心為 -4 其乘上的權重數 -8 會比 -4 要來的大，所以更容易凸顯出影像中變化較大的區域。

影像中掃過每一個像素值，每一個像素值的周圍與 filter 相乘並加總得出轉換後的像素值，在邊緣的部分，需要判斷像數值的周圍是否超過影像的大小，若超過的話則設為 0 與 filter 進行相乘。

```
for(int i = 0; i < Height; i++)
{
    for(int j = 0; j < Width; j++)
    {
        number = 0;
        for(int s = -1 * (size_num / 2); s < (size_num / 2) + 1; s++)
        {
            for(int t = -1 * (size_num / 2); t < (size_num / 2) + 1; t++)
            {
                if((i + s) >= 0 && (i + s) < Height && (j + t) >= 0 && (j + t) < Width)
                {
                    number += Graymatrix[i + s, j + t] * filter_array[s + (size_num / 2), t + (size_num / 2)];
                }
                else
                {
                    number += 0 * filter_array[s + (size_num / 2), t + (size_num / 2)];
                }
            }
        }
        if(number > 255)
        {
            number = 255;
        }
        if(number < 0)
        {
            number = 0;
        }
        Graymatrix_af[i, j] = (Byte)number;
    }
```