Digital Image Processing Homework Assignment #1

D06922014 劉育綸

1. In DIP_hw1\DIP_hw1\DIPhw1.cpp:

```
void Resize(const Mat &mInput, Mat &mOutput, double dScalingFactor, string sOption)
```

- mInput – input image.
- mOutput – output image.
- dScalingFactor -- scale factor along the horizontal and vertical axis.
- sOption – interpolation method:
  - “bilinear” – a bilinear interpolation.
  - “bicubic” – a bicubic interpolation.

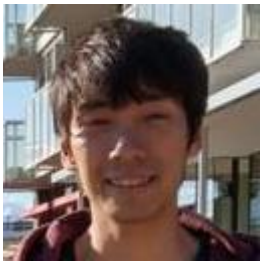
DIP_hw1\x64\Release\DIP_hw1.exe usage:

DIP_hw1 [input_image_name] [output_image_name] [scaling_factor] [interpolation_method]

e.g. DIP_hw1 myphoto.png bilinear_0.png 0.28 bilinear

e.g. DIP_hw1 myphoto.png bicubic_2.png 19 bicubic

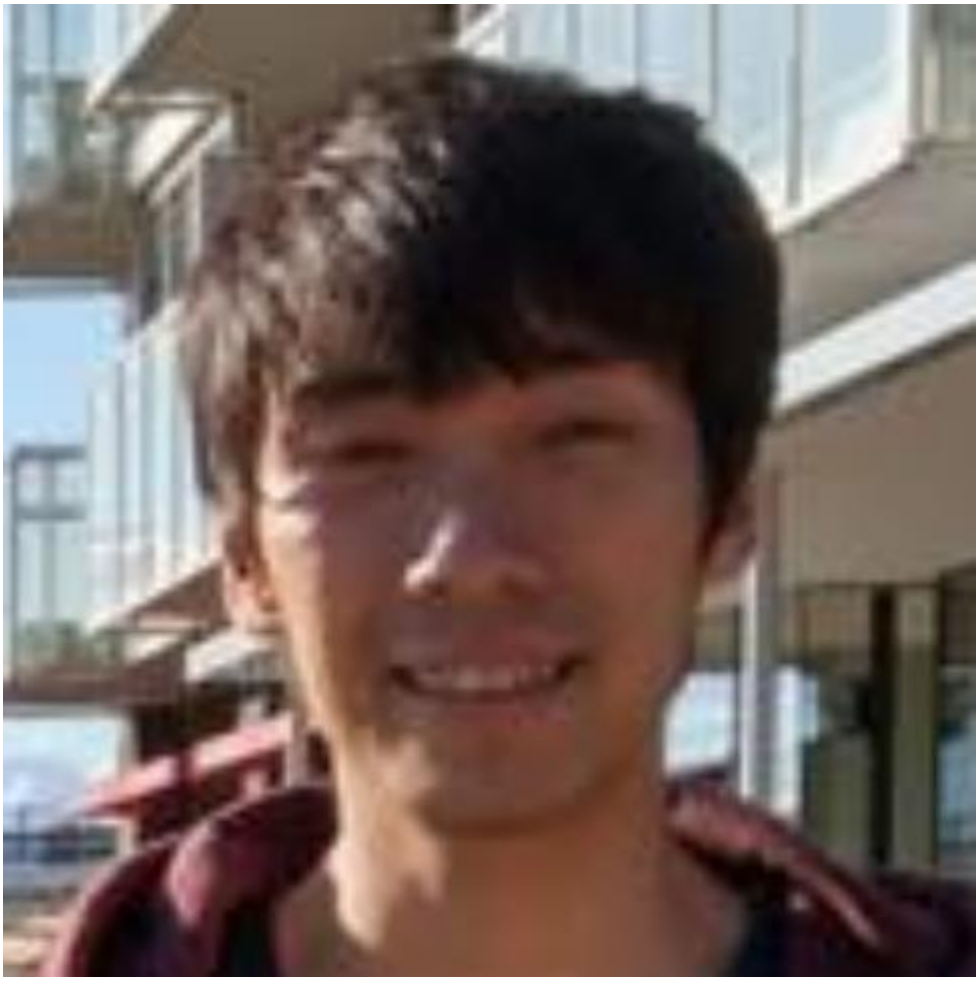
2.

 Input image (128x128).

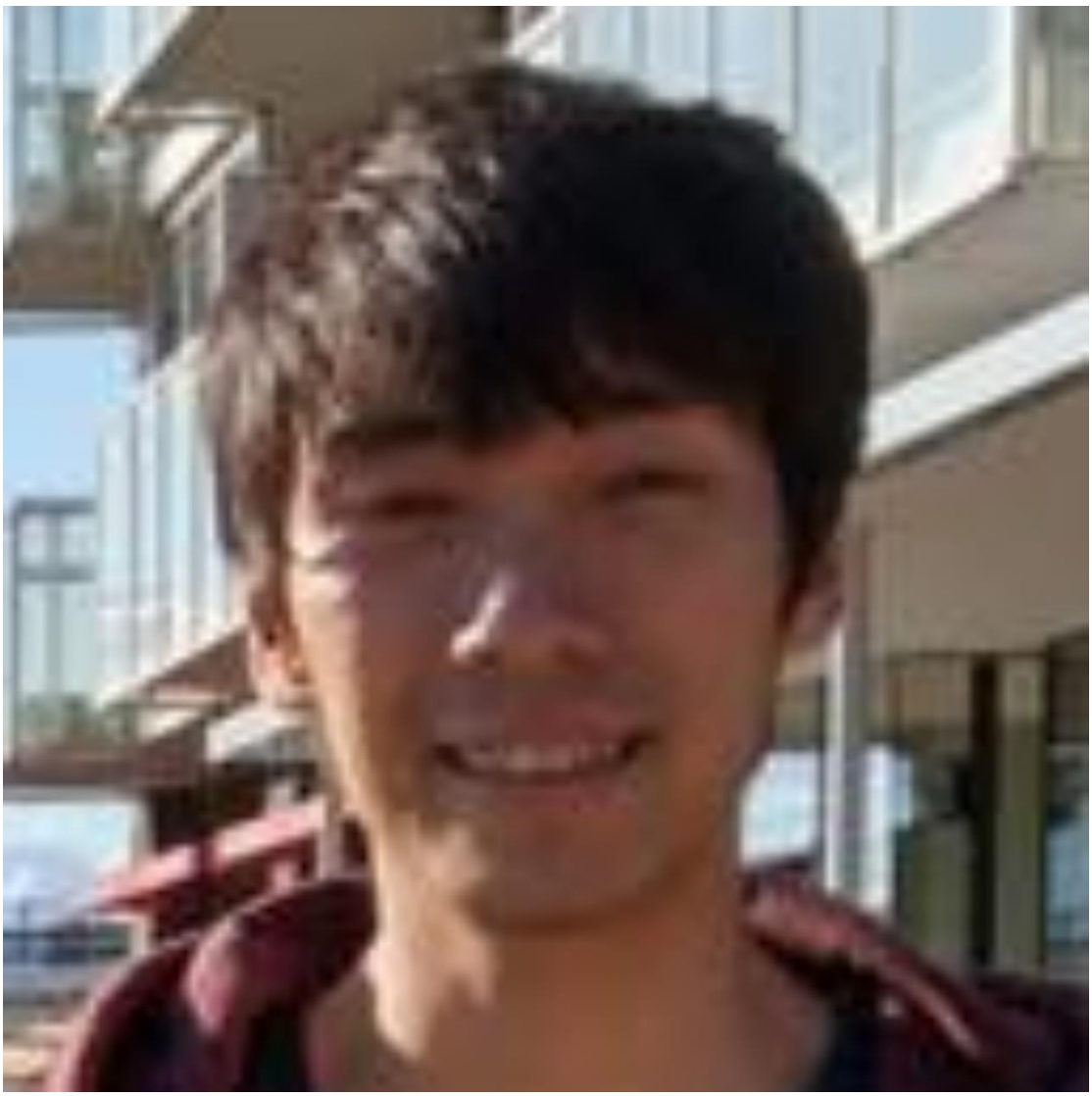
DIP_hw1\x64\Release\myphoto.png.

 bilinear interpolation, scaling factor = 0.28 (36x36).

DIP_hw1\x64\Release\bilinear_0.png.

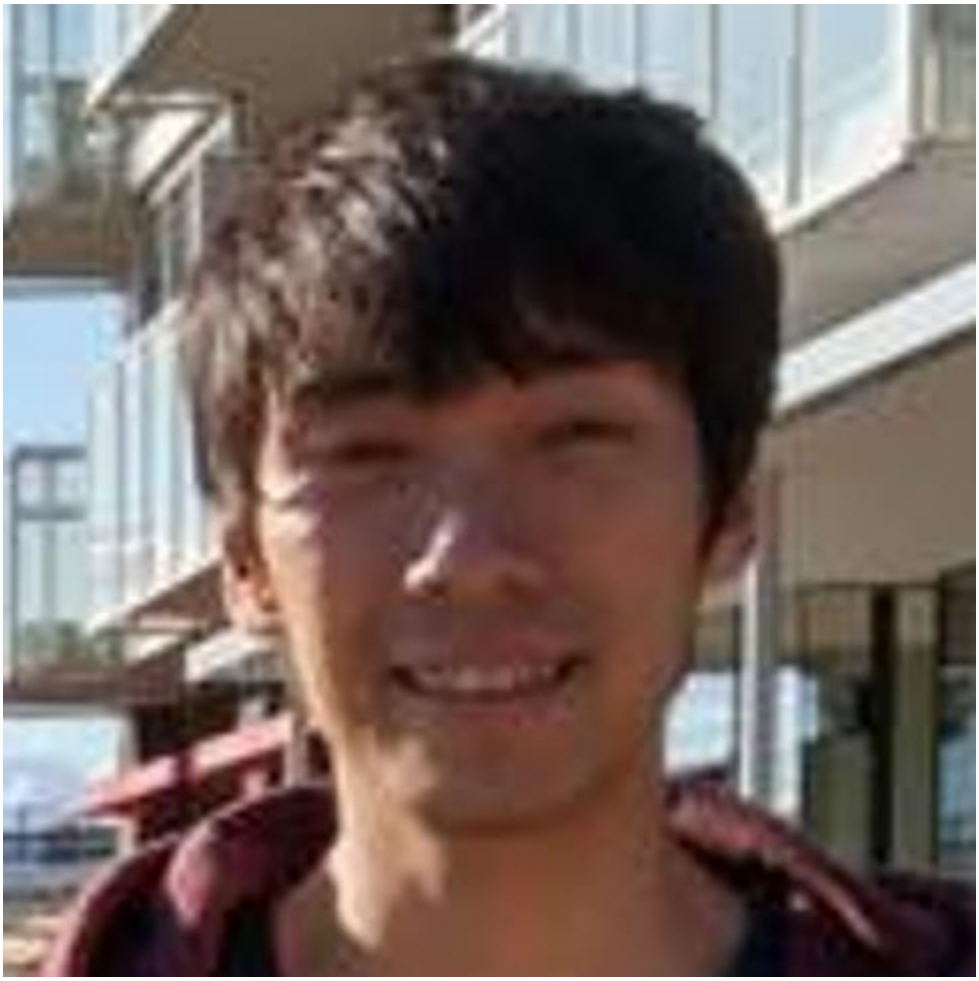


bilinear interpolation, scaling factor = 3.8 (486x486).
DIP_hw1\x64\Release\bilinear_1.png.

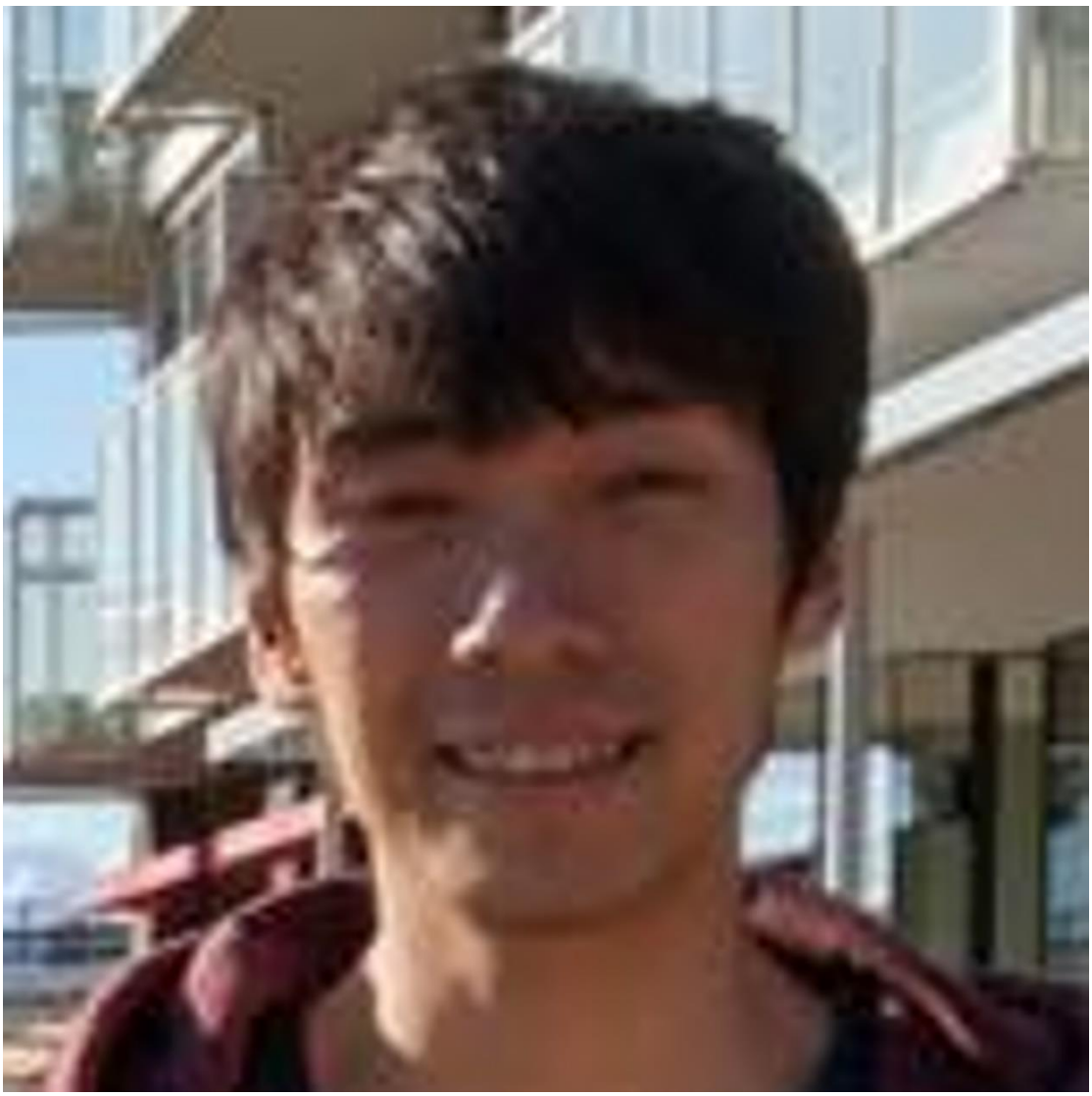


bilinear interpolation, scaling factor = 19 (2432x2432).
DIP_hw1\x64\Release\bilinear_2.png(因版面大小限制所以 word 有將它縮小).

 bicubic interpolation, scaling factor = 0.28 (36x36).
DIP_hw1\x64\Release\bicubic_0.png.

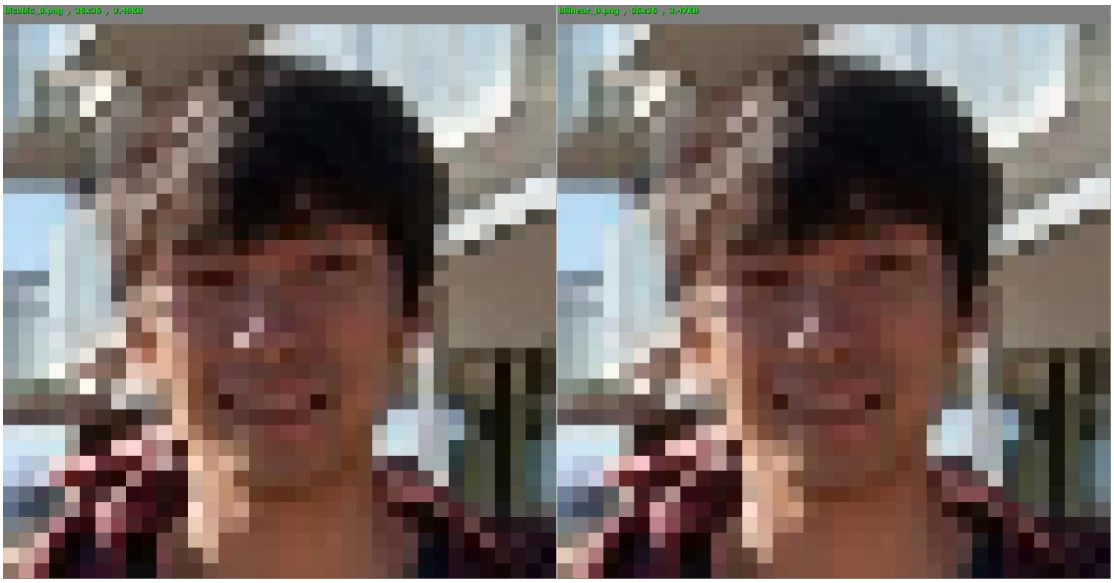


bicubic interpolation, scaling factor = 3.8 (486x486).
DIP_hw1\x64\Release\bicubic_1.png.



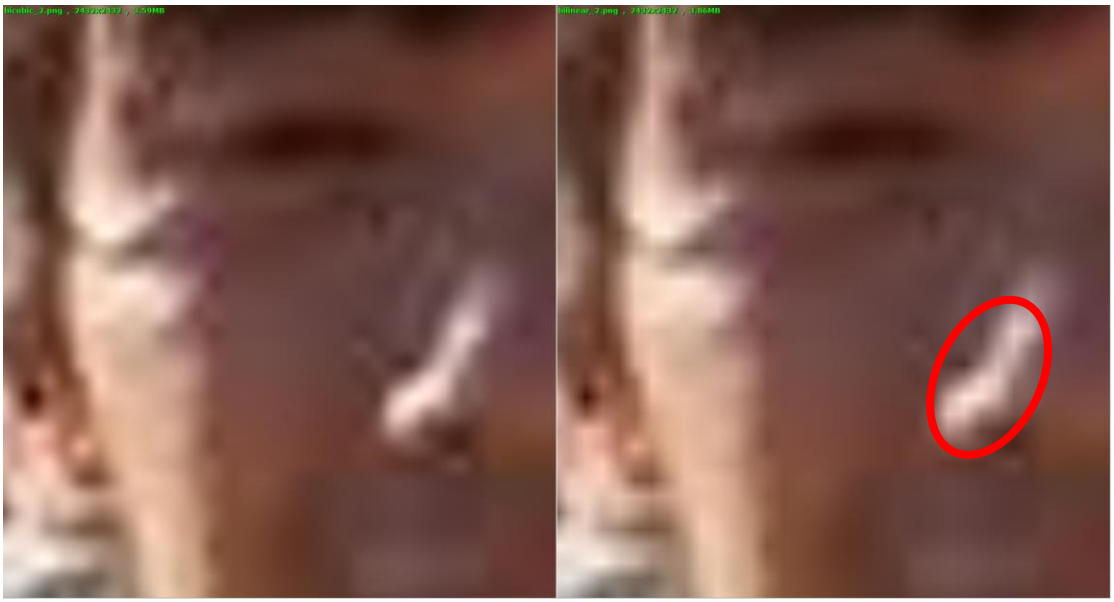
bicubic interpolation, scaling factor = 19 (2432x2432).
DIP_hw1\x64\Release\bicubic_2.png.

3. scaling factor = 0.28: 因為 output 圖像解析度過小, 所以在 bilinear 與 bicubic 之間看不出太多差異, 但是兩者都有產生 aliasing 現象, 所有的像素顏色都變成一格一格的. (下左圖: bicubic, 下右圖: bilinear)

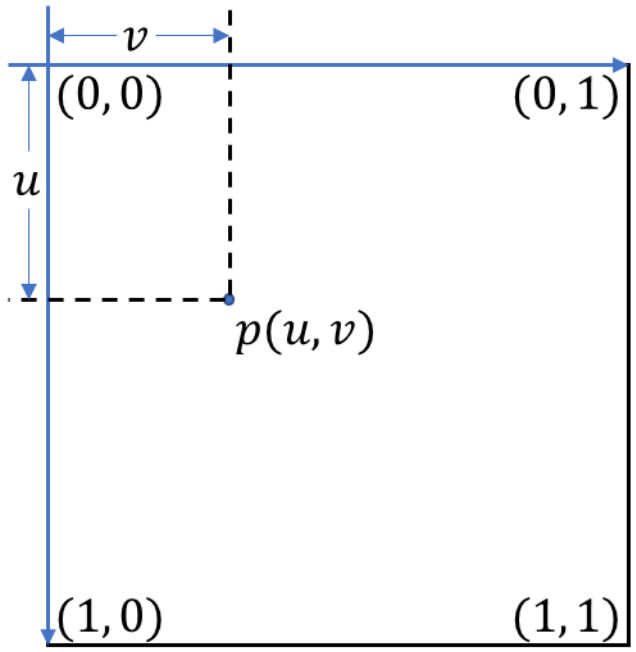


scaling factor = 3.8: bilinear 與 bicubic 的表現在這個倍率下差不多，不過 bilinear 會出現線性的十字亮點 artifacts，而 bicubic 保留了較多的影像細節. (下左圖: bicubic，下右圖: bilinear)

scaling factor = 19: 在這個倍率之下的 bicubic 明顯比 bilinear 平順許多，bilinear 會產生很多線性的十字形亮點 artifacts, bicubic 則可以把這些地方抹平，但又不失細節. (下左圖: bicubic，下右圖: bilinear)



4. 這邊先說明一般性的 bicubic 內插法，後半部會再說明如何將這一般性的內插應用於整張圖的縮放，假設位於單位方格內的任意非整數座標點(u, v)的亮度為$p(u, v)$.



那我們可以用 16 個係數來決定這單位方格內的內插曲面$p(u, v) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}u^i v^j$. 由於需要求解 16 個係數，所以我們需要 16 個方程式來求解. 先列出跟單位方格四個角點$(0,0), (1,0), (0,1), (1,1)$有關的 4 個方程式:

1. $p(0,0) = a_{00}$,
2. $p(1,0) = a_{00} + a_{10} + a_{20} + a_{30}$,
3. $p(0,1) = a_{00} + a_{01} + a_{02} + a_{03}$,

4. $p(1,1) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}$.

接著列出跟u方向(垂直)微分有關的 4 條方程式:

1. $\frac{\partial p(0,0)}{\partial u} = p_u(0,0) = a_{10}$,

2. $\frac{\partial p(1,0)}{\partial u} = p_u(1,0) = a_{10} + 2a_{20} + 3a_{30}$,

3. $\frac{\partial p(0,1)}{\partial u} = p_u(0,1) = a_{10} + a_{11} + a_{12} + a_{13}$,

4. $\frac{\partial p(1,1)}{\partial u} = p_u(1,1) = \sum_{i=1}^{3}\sum_{j=0}^{3} a_{ij}\, i$.

其中定義$p_u(u,v)$為此內插曲面在(u,v)這點的u方向偏微分, 接著同樣的列出跟v方向(水平)微分有關的 4 條方程式:

1. $\frac{\partial p(0,0)}{\partial v} = p_v(0,0) = a_{01}$,

2. $\frac{\partial p(1,0)}{\partial v} = p_v(1,0) = a_{01} + a_{11} + a_{21} + a_{31}$,

3. $\frac{\partial p(0,1)}{\partial v} = p_v(0,1) = a_{01} + 2a_{02} + 3a_{03}$,

4. $\frac{\partial p(1,1)}{\partial v} = p_v(1,1) = \sum_{i=0}^{3}\sum_{j=1}^{3} a_{ij}\, j$.

其中定義$p_v(u,v)$為此內插曲面在(u,v)這點的v方向偏微分, 最後是u與v方向的混合微分, 列出 4 條式子:

1. $\frac{\partial^2 p(0,0)}{\partial u \partial v} = p_{uv}(0,0) = a_{11}$,

2. $\frac{\partial^2 p(1,0)}{\partial u \partial v} = p_{uv}(1,0) = a_{11} + 2a_{21} + 3a_{31}$,

3. $\frac{\partial^2 p(0,1)}{\partial u \partial v} = p_{uv}(0,1) = a_{11} + 2a_{12} + 3a_{13}$,

4. $\frac{\partial^2 p(1,1)}{\partial u \partial v} = p_{uv}(1,1) = \sum_{i=1}^{3}\sum_{j=1}^{3} a_{ij}\, ij$.

其中定義$p_{uv}(u,v)$為此內插曲面在(u,v)這點的u與v方向混合微分.

綜合以上 16 條式子我們可以得到一個 Ax=b 的形式:

$$\begin{bmatrix}1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\0&1&2&3&0&0&0&0&0&0&0&0&0&0&0&0\\0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\0&1&2&3&0&1&2&3&0&1&2&3&0&1&2&3\\0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\0&0&0&0&1&0&0&0&2&0&0&0&3&0&0&0\\0&0&0&0&1&1&1&1&2&2&2&2&3&3&3&3\\0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&1&2&3&0&0&0&0&0&0&0&0\\0&0&0&0&0&1&0&0&0&2&0&0&0&3&0&0\\0&0&0&0&0&1&2&3&0&2&4&6&0&3&6&9\end{bmatrix}\begin{bmatrix}a_{00}\\a_{10}\\a_{20}\\a_{30}\\a_{01}\\a_{11}\\a_{21}\\a_{31}\\a_{02}\\a_{12}\\a_{22}\\a_{32}\\a_{03}\\a_{13}\\a_{23}\\a_{33}\end{bmatrix}=\begin{bmatrix}p(0,0)\\p(1,0)\\p(0,1)\\p(1,1)\\p_u(0,0)\\p_u(1,0)\\p_u(0,1)\\p_u(1,1)\\p_v(0,0)\\p_v(1,0)\\p_v(0,1)\\p_v(1,1)\\p_{uv}(0,0)\\p_{uv}(1,0)\\p_{uv}(0,1)\\p_{uv}(1,1)\end{bmatrix}$$

將左邊這個 16x16 的矩陣做 inverse 可以得到:

$$\begin{bmatrix}1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\-3&3&0&0&-2&-1&0&0&0&0&0&0&0&0&0&0\\2&-2&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\0&0&0&0&0&0&0&0&-3&3&0&0&-2&-1&0&0\\0&0&0&0&0&0&0&0&2&-2&0&0&1&1&0&0\\-3&0&3&0&0&0&0&0&-2&0&-1&0&0&0&0&0\\0&0&0&0&-3&0&3&0&0&0&0&0&-2&0&-1&0\\9&-9&-9&9&6&3&-6&-3&6&-6&3&-3&4&2&2&1\\-6&6&6&-6&-3&-3&3&3&-4&4&-2&2&-2&-2&-1&-1\\2&0&-2&0&0&0&0&0&1&0&1&0&0&0&0&0\\0&0&0&0&2&0&-2&0&0&0&0&0&1&0&1&0\\-6&6&6&-6&-4&-2&4&2&-3&3&-3&3&-2&-1&-2&-1\\4&-4&-4&4&2&2&-2&-2&2&-2&2&-2&1&1&1&1\end{bmatrix}\begin{bmatrix}p(0,0)\\p(1,0)\\p(0,1)\\p(1,1)\\p_u(0,0)\\p_u(1,0)\\p_u(0,1)\\p_u(1,1)\\p_v(0,0)\\p_v(1,0)\\p_v(0,1)\\p_v(1,1)\\p_{uv}(0,0)\\p_{uv}(1,0)\\p_{uv}(0,1)\\p_{uv}(1,1)\end{bmatrix}=\begin{bmatrix}a_{00}\\a_{10}\\a_{20}\\a_{30}\\a_{01}\\a_{11}\\a_{21}\\a_{31}\\a_{02}\\a_{12}\\a_{22}\\a_{32}\\a_{03}\\a_{13}\\a_{23}\\a_{33}\end{bmatrix}$$

如此一來，當有任意單位方格的四個角點的亮度與微分資訊，我們就可以用上式來得出此單位方格內的 16 個內插參數.

```
Mat A = (Mat_<double>(16, 16) <<
    1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    -3, 3, 0, 0, -2, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    2, -2, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, -3, 3, 0, 0, -2, -1, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 2, -2, 0, 0, 1, 1, 0, 0,
    -3, 0, 3, 0, 0, 0, 0, 0, -2, 0, -1, 0, 0, 0, 0, 0,
    0, 0, 0, 0, -3, 0, 3, 0, 0, 0, 0, 0, -2, 0, -1, 0,
    9, -9, -9, 9, 6, 3, -6, -3, 6, -6, 3, -3, 4, 2, 2, 1,
    -6, 6, 6, -6, -3, -3, 3, 3, -4, 4, -2, 2, -2, -2, -1, -1,
    2, 0, -2, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 2, 0, -2, 0, 0, 0, 0, 0, 1, 0, 1, 0,
    -6, 6, 6, -6, -4, -2, 4, 2, -3, 3, -3, 3, -2, -1, -2, -1,
    4, -4, -4, 4, 2, 2, -2, -2, 2, -2, 2, -2, 1, 1, 1, 1
    );
```

但現在問題是，我們要如何得到一張影像像素點的微分值，這邊我使用 finite difference，更精確地來說是 central difference:

1. $p_u(u,v) = \frac{p(u+1,v) - p(u-1,v)}{2}$,
2. $p_v(u,v) = \frac{p(u,v+1) - p(u,v+1)}{2}$,

而$p_{uv}(u,v)$混合微分則是先算出u方向再算v方向，反之亦可．而在影像的邊界因為缺少其中一邊的像素點，所以會改成採用 forward 或是 backward difference:

1. $forward\ difference: p_u(u,v) = p(u+1,v) - p(u,v)$,
2. $forward\ difference: p_v(u,v) = p(u,v+1) - p(u,v)$
3. $backward\ difference: p_u(u,v) = p(u,v) - p(u-1,v)$,
4. $backward\ difference: p_v(u,v) = p(u,v) - p(u,v-1)$

```
inline void FiniteDeference(const Mat &mInput, Mat &mOutput, bool bDirection)
{
    mOutput = Mat(mInput.rows, mInput.cols, CV_64FC3);
    for (int u = 0; u < mOutput.rows; u++)
    {
        for (int v = 0; v < mOutput.cols; v++)
        {
            if (!bDirection) // u
            {
                int u_n = u - 1;
                int u_p = u + 1;
                if (u_n < 0 && u_p < mOutput.rows)
                    mOutput.at<Vec3d>(u, v) = (mInput.at<Vec3d>(u_p, v) - mInput.at<Vec3d>(u, v));
                else if (u_p >= mOutput.rows && u_n >= 0)
                    mOutput.at<Vec3d>(u, v) = (mInput.at<Vec3d>(u, v) - mInput.at<Vec3d>(u_n, v));
                else
                    mOutput.at<Vec3d>(u, v) = (mInput.at<Vec3d>(u_p, v) - mInput.at<Vec3d>(u_n, v)) / 2.0;
            }
            else // v
            {
                int v_n = v - 1;
                int v_p = v + 1;
                if (v_n < 0 && v_p < mOutput.cols)
                    mOutput.at<Vec3d>(u, v) = (mInput.at<Vec3d>(u, v_p) - mInput.at<Vec3d>(u, v));
                else if (v_p >= mOutput.cols && v_n >= 0)
                    mOutput.at<Vec3d>(u, v) = (mInput.at<Vec3d>(u, v) - mInput.at<Vec3d>(u, v_n));
                else
                    mOutput.at<Vec3d>(u, v) = (mInput.at<Vec3d>(u, v_p) - mInput.at<Vec3d>(u, v_n)) / 2.0;
            }
        }
    }
}
```

如此一來我們便可以先算出整張影像的微分值，之後內差的時候僅需要查表就可以了.

```
Mat u_prime, v_prime, u_prime_v_prime;
if (sOption == "bicubic")
{
    Mat mInput64F;
    mInput.convertTo(mInput64F, CV_64FC3);
    FiniteDeference(mInput64F, u_prime, 0);
    FiniteDeference(mInput64F, v_prime, 1);
    FiniteDeference(u_prime, u_prime_v_prime, 1);
}
```

有了這 16 個內插係數我們便可以使用以下式子計算出內插的影像強度:

$$p(u,v) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 \\ v \\ v^2 \\ v^3 \end{bmatrix}.$$

最後說明如何將整張影像縮放，首先用 input 影像的大小與 scaling factor 算出 output 影像大小，接著走訪每個 output 影像像素，計算它源自於 input 影像的哪個座標位置，計算方法如下(以一維舉例):

Input 影像(長度 w)的像素 p (p: 0~w-1)在影像上的位置比例為$\frac{1}{2} + \frac{p}{w} =$

$\frac{1+2p}{2w}$, 同樣的, output 影像(長度 W)的像素 P (P: 0~W-1)在影像上的位置比例為$\frac{1}{2}+\frac{P}{W}=\frac{1+2P}{2W}$, 其中 W = w * scalingFactor, 帶入後得到$p=\frac{1+2P-scalingFactor}{2\times scalingFactor}$, 用這條式子就可以得知 output 影像上任意座標像素點對應到 input 影像上的座標位置

```
// Convert P to image_ratio to p
double duInput = (1 + 2 * u - dScalingFactor) / (2 * dScalingFactor);
double dvInput = (1 + 2 * v - dScalingFactor) / (2 * dScalingFactor);
```

得到 input 座標後在長寬兩個維度取地板函數與地板函數+1 便可得到包覆此點的單位方格, 便可用前述的 interpolation 方法內插出影像強度.

```
Mat u_vec = (Mat_<double>(1, 4) << 1, beta, beta * beta, beta * beta * beta);
Mat v_vec = (Mat_<double>(4, 1) << 1, alpha, alpha * alpha, alpha * alpha * alpha);
for (int channel = 0; channel < 3; channel++)
{
    Mat b = (Mat_<double>(16, 1) <<
        mInput.at<Vec3b>(iFlooruInput, iFloorvInput)[channel],
        mInput.at<Vec3b>(iCeiluInput, iFloorvInput)[channel],
        mInput.at<Vec3b>(iFlooruInput, iCeilvInput)[channel],
        mInput.at<Vec3b>(iCeiluInput, iCeilvInput)[channel],
        u_prime.at<Vec3d>(iFlooruInput, iFloorvInput)[channel],
        u_prime.at<Vec3d>(iCeiluInput, iFloorvInput)[channel],
        u_prime.at<Vec3d>(iFlooruInput, iCeilvInput)[channel],
        u_prime.at<Vec3d>(iCeiluInput, iCeilvInput)[channel],
        v_prime.at<Vec3d>(iFlooruInput, iFloorvInput)[channel],
        v_prime.at<Vec3d>(iCeiluInput, iFloorvInput)[channel],
        v_prime.at<Vec3d>(iFlooruInput, iCeilvInput)[channel],
        v_prime.at<Vec3d>(iCeiluInput, iCeilvInput)[channel],
        u_prime_v_prime.at<Vec3d>(iFlooruInput, iFloorvInput)[channel],
        u_prime_v_prime.at<Vec3d>(iCeiluInput, iFloorvInput)[channel],
        u_prime_v_prime.at<Vec3d>(iFlooruInput, iCeilvInput)[channel],
        u_prime_v_prime.at<Vec3d>(iCeiluInput, iCeilvInput)[channel]
        );

    Mat a = A * b;

    Mat out = u_vec * (a.reshape(0, 4).t()) * v_vec;
    // Clipping
    uchar p = (out.at<double>(0, 0) > 255) ? 255 : uchar(out.at<double>(0, 0));
    p = (out.at<double>(0, 0) < 0) ? 0 : p;
    mOutput.at<Vec3b>(u, v)[channel] = p;
```

再來是計算複雜度的部分, 我就直接量測原影像(128x128)在 bilinear 與 bicubic 縮放 0.28, 3.8 與 19 倍的執行時間(量測 10 次取平均), 單位為 microseconds, 電腦是 Intel Core i7-7500U CPU @ 2.70GHz:

microseconds (us)	0.28	3.8	19
bilinear	106	17092.6	416873.8
bicubic	34612.3	1993047	49576145

不管是 bilinear 或是 bicubic, 隨著 scaling factor 上升, 執行時間也會平方上升(與 scaling factor 平方成正比). 而在 scaling factor 相同的情況下, bicubic

的計算量明顯比 bilinear 高出許多，大約是 117 倍左右，小圖的差異較大(bicubic 是 bilinear 的 327 倍左右)可能是因為, bicubic 會因為需要計算微分 map 而有一些不可避免的 overhead，所以在小圖上的速度快不起來.

5. 我這次去參觀了白晝之夜當天晚上的風動四方 VR，晚上 7 點到的時候被告之 6 點開始發的票都已經被拿光了...所以本來打算 12 點再來排隊，不過後來有好心人士因為時間關係所以把 10:20~10:30 的體驗時間給我，還在現場遇到了公司的部門主管雷少民處長，跟雷博聊完天以後我便先前往椰林大道看看其他白晝之夜的活動, 10:20 再回來體驗風動四方. 風動四方的體驗非常好，在我轉動的時候畫面的延遲幾乎讓我感覺不到，此外加上風吹這種另外的體感更能讓使用者知道要注意 VR 的哪個方向，我覺得是非常好的創意跟實作，遊戲內容似乎是比較像本土版的太鼓達人，搭配豬哥亮與鄧麗君等早期藝人的歌聲，以下給出三點建議:
   - 一開始的”吸氣~吐氣~吸氣~吐氣~”聲音太大，我最後只得先拿掉耳機讓我的耳朵休息一下.
   - 工作人員一開始似乎沒有講到握把感測器在遊戲中是鼓棒，我一直到看到遊戲中的另一位透明人玩家在打鼓我才恍然大悟那原來是鼓棒.
   - 在椅子轉動的過程中 display 的線容易去勾到椅背，也許工作人員可以隨時注意幫忙把線解開.