

EE569 Digital Image Processing

Name: Shuo Wang

USC ID: 8749390300

Email: [wang133@usc.edu](mailto:wang133@usc.edu)

Date: Feb. 24, 2017

## EE 569: Homework #2

Issued: 2/3/2017 Due: 11:59PM, 2/26/2017

### Problem 1: Geometric Image Modification (40%)

#### (a) Geometrical Warping (Basic: 10%)

##### Motivation

To design a function to warp one part of image from one shape to another shape.

For a triangle, there are three control points for the input image and the output one. For a specific pair of input and output control points, we can get a transform matrix. Then, we can change all other point coordinates in the image through the transform matrix.

To find a method to change the square image to be a diamond image by the function above.

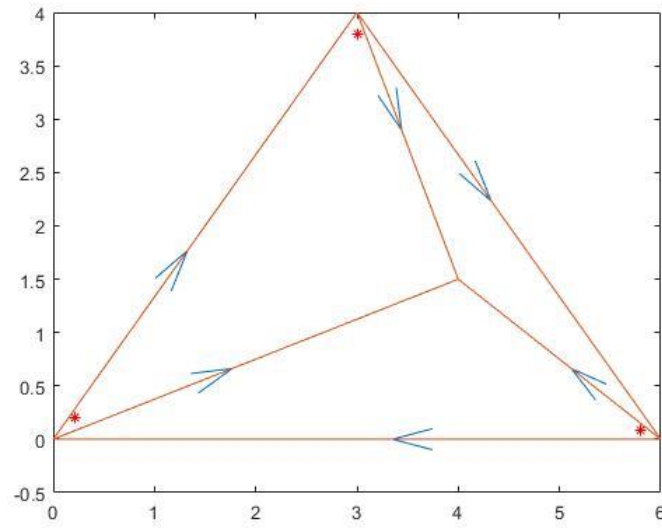
##### Approach and Procedures

In this section, I designed the function to warp a triangle to be an another triangle through input the desired output control point coordinates and the transformation matrix. According to the warping definition, we can get that

$$\begin{bmatrix} x_{out} \\ y_{out} \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \\ 1 \end{bmatrix}$$

Therefore, for a specific warping processing, we should calculate six parameters, which can be obtained by solving six equations generated by three input and output control points. In this part, I will use the MATLAB to get the warping matrix.

For the warping process, I should clarify which points are in the triangle and which ones are not. For this question, I calculate the three angles between the vector whose start point is the triangle vertex and the end point is the one we want to detect and the vector whose start point is the same triangle vertex and the end point is the another one clockwise, which can be shown below (the angle is shown as the mark of star):



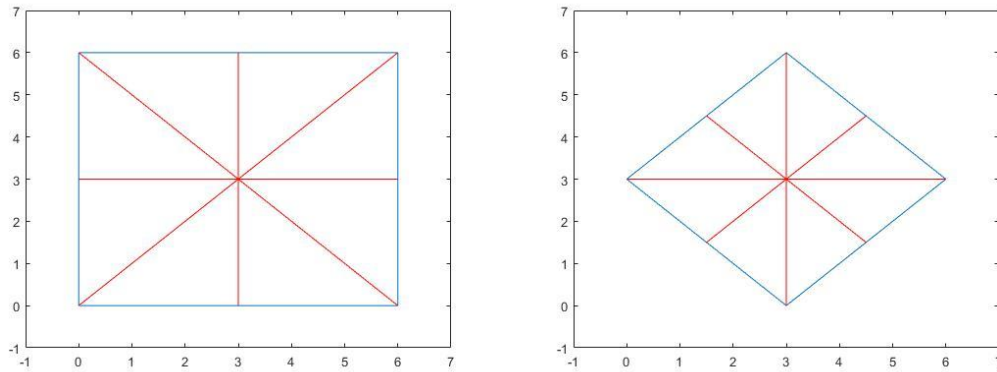
### The sketch map that a point is in the triangle

If all three angles are less than 90 degrees, we will regard it as the point in the triangle. Then, we will use the formula below to calculate the pixel at such point when the input coordinate corresponding to the output point are not integers (Bilinear interpolation):

$$F(p', q') = (1 - \Delta x)(1 - \Delta y)F(p, q) + (\Delta x)(1 - \Delta y)F(p + 1, q) + (1 - \Delta x)(\Delta y)F(p, q + 1) \\ + (\Delta x)(\Delta y)F(p + 1, q + 1)$$

$$\Delta x = p' - p \quad \Delta y = q' - q$$

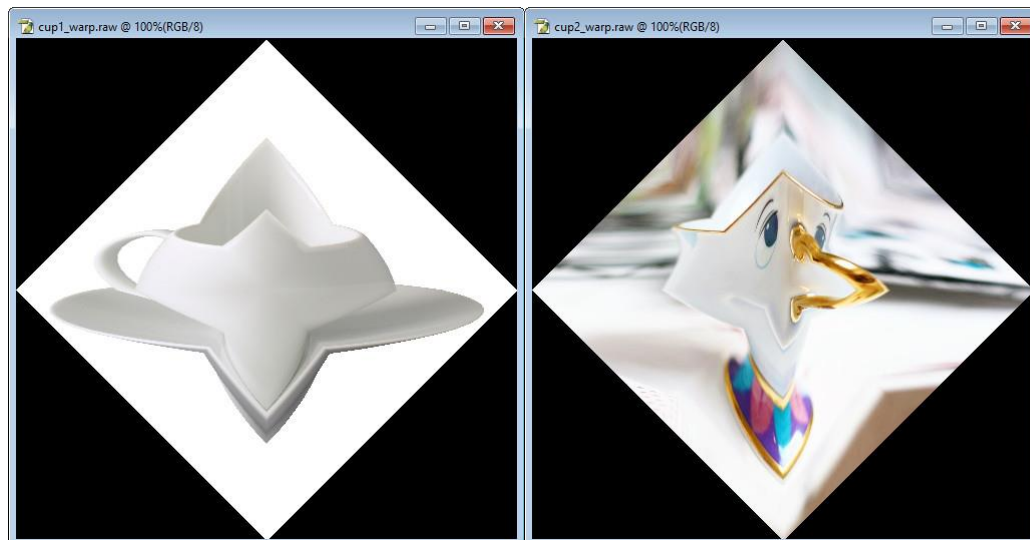
For the square image, I will divide the image into 8 parts which can be shown as follow, then, we will use the warp matrix to turn it to be the diamond shape as follow:



**The shape before the transformation (Left) and after the transformation (Right)**

### Experimental Results

The result of both the cup A and cup B can be shown as follows



**The warping image of cup A (Left) and cup B (Right)**

### **Discussion**

For the result, we can find that at the specific angle which can be divided by 45 degrees, we can get the inconsistent line as if it was cut. The reason is the method we divide the square. As is shown in the Approach and Procedures section, we use the triangle warp and divide the picture along the diagonal line. After using the warp, the distance along the diagonal is much less than the line along the perpendicular bisector, which is totally different from the origin image. Therefore, the relative distance is changed, which makes the inconsistent lines.

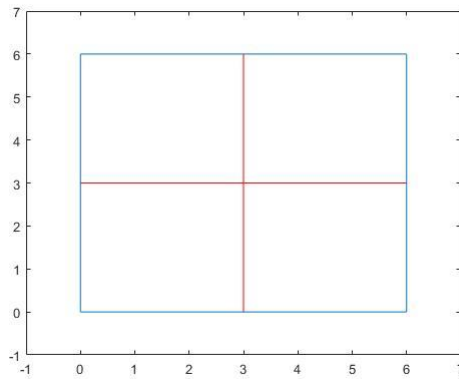
However, compared with the example image which is shown as follow:



**The transformation of the dog image in the example**

We can find that if we use the method above, we can get inconsistent line in the specific angle which can be divided by 45 degrees just like the image in the result, while there is not such line in the example. Therefore, we should consider another solution for the question.

At first, I divide the image into four parts:



### The method of division an image

From the image, I find that for a specific row, the number of pixels is equal to the row number while the relative distance between the points in the row is same. Then, for the piece of the left top, I find the idea that we can pick several points in the row whose deviation between two points will be

$$\text{Size of width}/(2 \times \text{the number of the row})$$

If we get the points which is not the integer, we can do the linear transformation to represent. If the desired coordinate is  $d$  (not integer), the closest integer on the left is  $d_0$ , the closest integer on the right is  $d_1$ , the value at the  $d$  will be

$$d = (d_1 - d) \times F(d_0) + (d_0 - d) \times F(d_1)$$

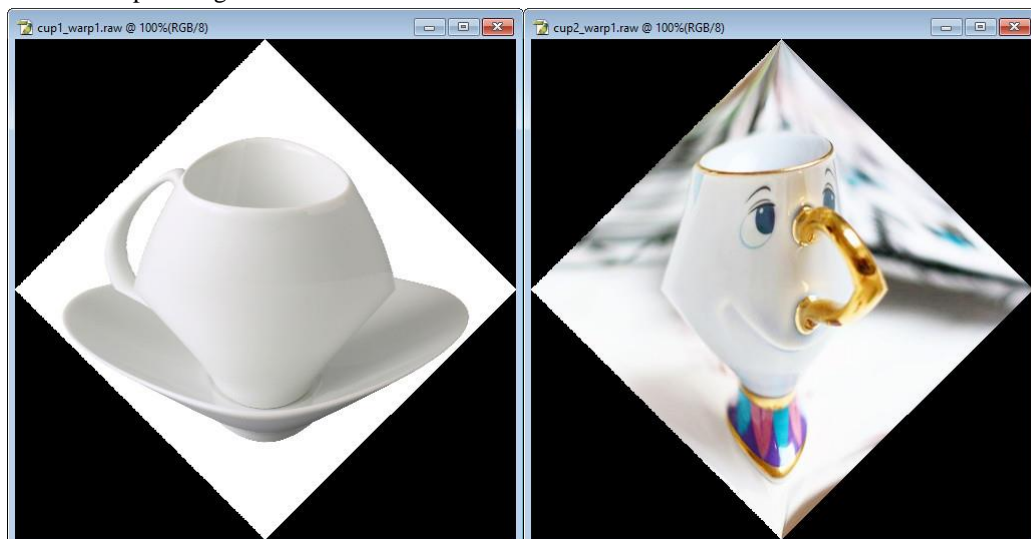
For the image of the top left, we can get the result as follow:

|         |         |         |         |
|---------|---------|---------|---------|
| $F(1)$  | $F(2)$  | $F(3)$  | $F(4)$  |
| $F(5)$  | $F(6)$  | $F(7)$  | $F(8)$  |
| $F(9)$  | $F(10)$ | $F(11)$ | $F(12)$ |
| $F(13)$ | $F(14)$ | $F(15)$ | $F(16)$ |

|                 |  |   |         |
|-----------------|--|---|---------|
| 0               | 0  | 0   | $F(4)$  |
| 0               | 0  | $F(5)$  | $F(8)$  |
| $\rightarrow$ 0 | $\frac{1}{3} \times F(10) + \frac{2}{3} \times F(9)$ | $\frac{1}{3} \times F(10) + \frac{2}{3} \times F(11)$ | $F(12)$ |
| $F(13)$         | $F(14)$  | $F(15)$   | $F(16)$ |

The operations for other squares are similar with the one of the top left square. By the operation, we can get the new output images as follow:



### **The output image of cup A (Left) and cup B (Right) by advanced methods**

From the result, we compared with the example image, we can see that this transformation satisfies the demands of the example image.

#### **(b) Puzzle Matching (Basic: 15%)**

##### **Motivation**

Assume we only know that the upper piece in the pieces.raw belongs to the Hillary and the lower one belongs to Trump. We have to know the coordinates of four edges in two images first.

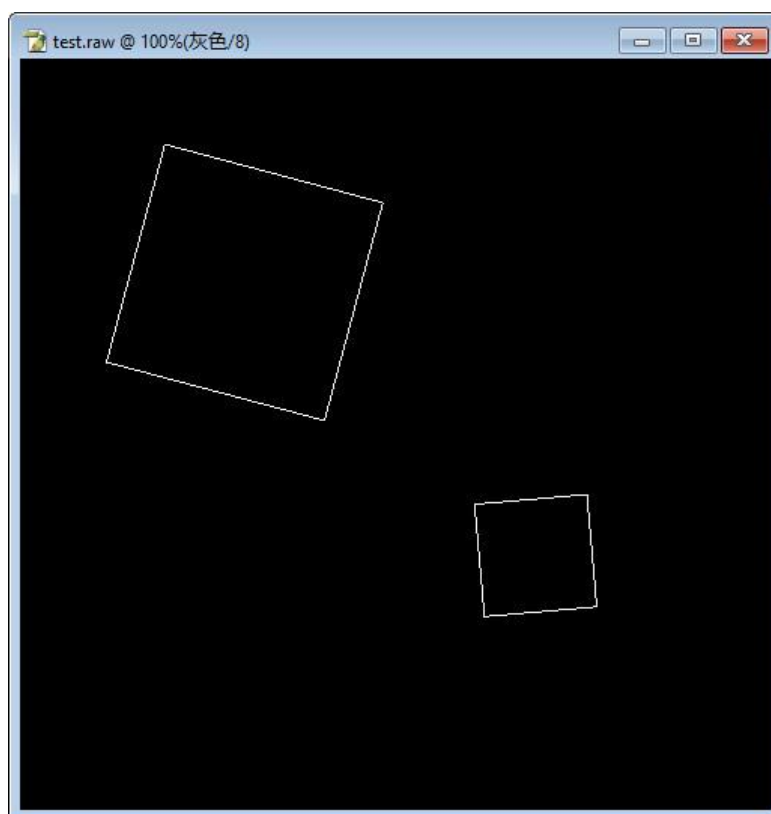
Having known the coordinates of edges in two images, we will propose a solution to make the puzzle pieces back to the origin image correctly.

##### **Approach and Procedures**

Assume we only know that the upper piece in the pieces.raw belongs to the Hillary and the lower one belongs to Trump, we can use the C++ program to detect the boundary of two images.

At first, we should scan the piece.raw from the start point. If we can get the pixel that it is not the background while the some of the pixels around it are background, we will regard it as the edge and save the coordinates in an array. Then we keep scanning. When we find that a row all pixels of which are background, we will record the row coordinates and stop recording the pixel in to the array.

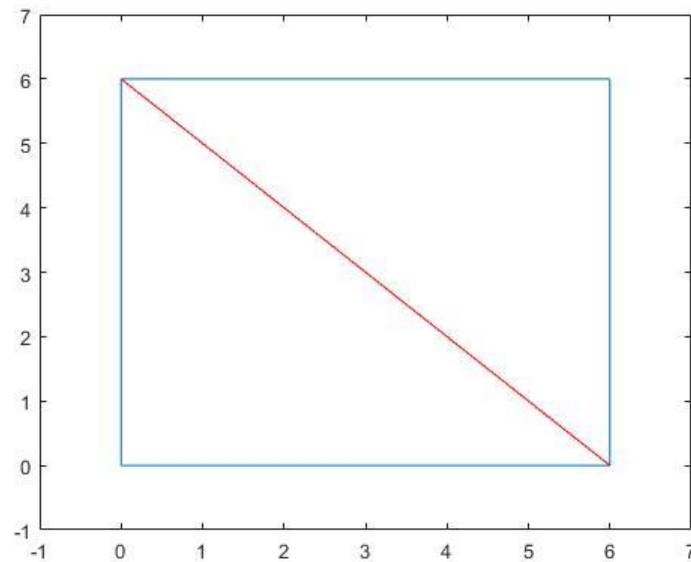
Then, we will scan the edge of another image. We will continue scanning the pixels from the recorded row and save the coordinates in another array if we can get the pixel that it is not the background while the some of the pixels around it are background. Then, we keep scanning and recording the coordinate of edge. When we find that a row all pixels of which are background, we will stop recording the coordinates of edge. As a result, we can get the edge image which can be shown below:



### The edge of piece.raw

Secondly, we will get the coordinates of corners in each images from the data of edges above.

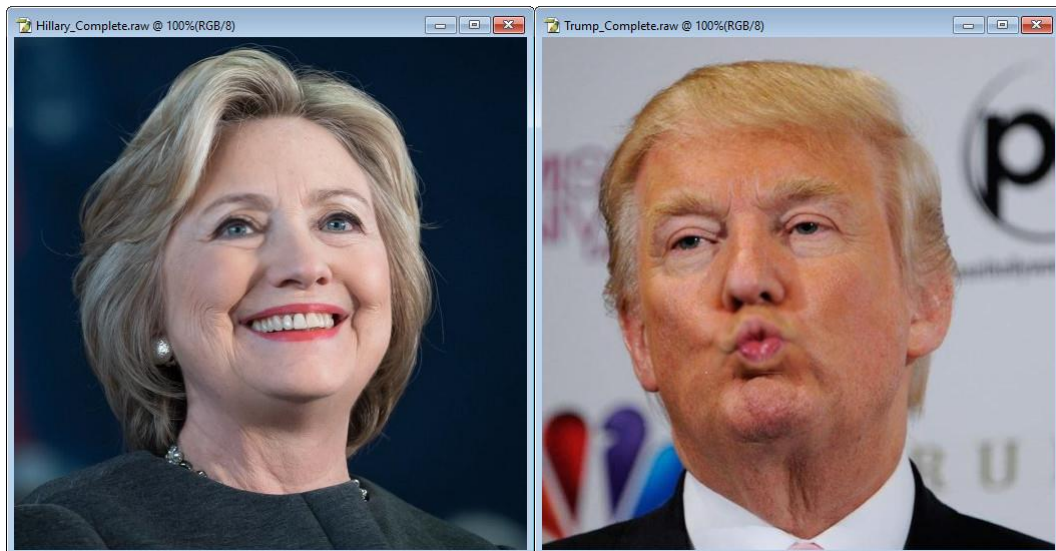
Then, we will use the warping to warp the piece image to the holes in the origin image, and the warping parameter matrix can be obtained from the MATLAB having got the corresponding warping coordinate pairs. As a result, we can get the complete image we want. The warping division can be shown as follow:



### The division that the triangles will be warped

#### Experimental Results

The result of Hillary and Trump can be shown below:



The completed image of Hillary (Left) and Trump (Right)

#### Discussion

In this question, there will be a series of questions.

The first problem is to find the coordinates of corners. If the image is analog, we can regard the first point

saved in the array above is one of the corner, and the last point is the corner in the same diagonal. Then, assume that it is a square, we can get other corners of the squares.

However, there is a problem: the image is digital, which means that it has a series of steps if it is not placed evenly. As a result, if we use the method above to get the corner, there will be some deviation between the real corners and obtained points. As a result, we should do some revision for the question.

For the piece of Hillary, we can find that the first point is to the left of the image. As a result, we should find the corner as follow:

1. Get the first point as the first corner.
2. Get the maximum and minimum value of the coordinates of the pieces (use the pixel coordinates)
3. Regard the first point with the maximum column pixel-coordinate value as the second corner.
4. Regard the last point with the maximum row pixel-coordinate value as the third corner
5. Regard the last point with the minimum column pixel-coordinate value as the forth corner

Then, we can get the coordinate of the four corners as follow: (represented by the x-y coordinates)

(96.5, 442.5) (241.5, 403.5) (202.5, 259.5) (57.5, 297.5)

For the piece of Trump, we can find that the first point is to the right of the image. As a result, we should find the corner as follow:

1. Get the maximum and minimum value of the coordinates of the pieces (use the pixel coordinates)
2. Regard the last point with the minimum row pixel-coordinate value as the second corner
3. Regard the last point with the maximum column pixel-coordinate value as the second corner.
4. Regard the first point with the maximum row pixel-coordinate value as the third corner
5. Regard the first point with the minimum column pixel-coordinate value as the forth corner

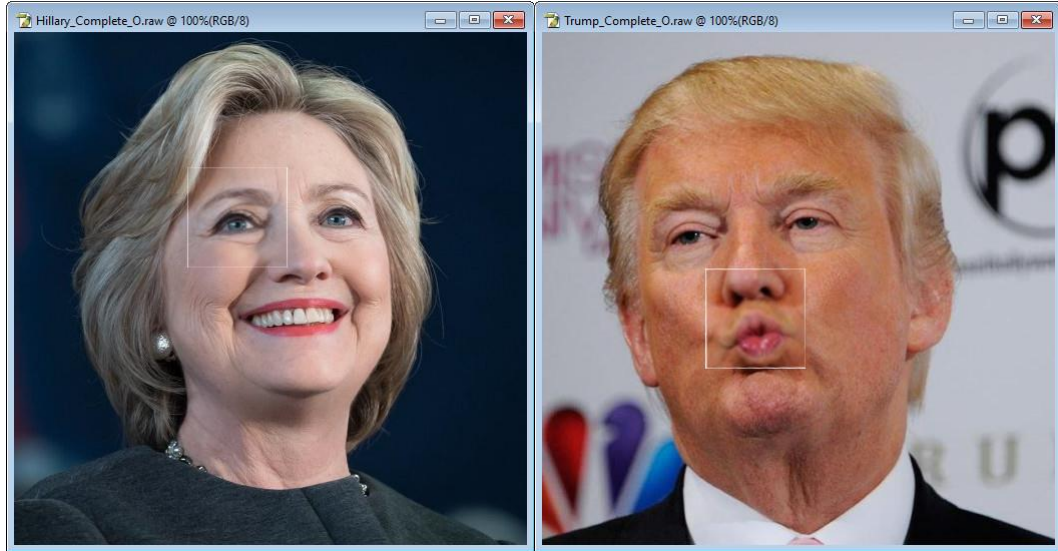
Then, we can get the coordinate of the four corners as follow: (represented by the x-y coordinates)

(377.5, 209.5) (383.5, 135.5) (308.5, 128.5) (302.5, 203.5)

The second Problem is the detection of the value of rotation angle. Since the piece of the image has been rotated, we have to put forward some procedures to match the puzzle pieces correctly. As a result, I decide to compare the pixel of one of the corner in the puzzle piece and the four corners in the hole of origin images. If the piece is matched, we will warp the piece to a specific direction and put it into the hole.

In addition, there will be another problem for this question. When we put the matched puzzle patch to the origin image, there will be some white gaps because the digital image patch has the steps when rotating. (which can be shown below)





**The assembled images of Hillary (left) and Trump (right) with gaps**

It is not good because it has not assembled completely. Therefore, we should use some calculations to fill in the pixel value in the gaps and smooth the boundary.

For the case of Hillary, the gaps locate in the left, right, top and bottom edge evenly and the gap is only one. We can fill the average of the pixels (those pixel values are not equal to 255) next to the gaps. For the larger gap (the width of gaps is 2, which locate in the left, top and bottom edge) in the Trump, we can use the average of pixels (those pixel values are not equal to 255) 2-pixel away from the white regions for each gap. Having filled the gap, we can get the result in the experimental result section.

### **The non-programming question**

1 Find the coordinates of four corners

The result has been shown in the first question of the discussion section.

2 The geometric transformation method Choose:

For this section, I use two-triangle warping to put the puzzle patch back to the origin image. The method is straightforward: You can just know the input and output control points, and you can get the warp transformation matrix through the MATLAB. Then, you can use the matrix directly to acquire the result of the completed image. Also, the Bilinear interpolation has been used in the function of the warping.

3 Find the holes in the origin image: the method has been discussed in the section of the Discussion when talking about the second question.

### **(c) Homographic Transformation and Image Overlay (Advanced: 15%)**

#### **Motivation**

Use the Homographic transformation to transform the patch picture into a photograph background. The Homographic transformation can be depicted as:

$$\begin{bmatrix} x'_{out} \\ y'_{out} \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \\ 1 \end{bmatrix} \quad \begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = \begin{bmatrix} \frac{x'_{out}}{w} \\ \frac{y'_{out}}{w} \end{bmatrix}$$



In the matrix, “in” is the final coordinate in the background image corresponding the pixel of the origin patch picture, “out” is the origin patch picture.

### Approach and Procedures

In this question, the basic theory is similar with the warping transformation. We have to use the four control points to transform the image from one shape to another, which requires us to get the Homographic transformation matrix. However, the only thing we know is the 4 input control points and four output control point corresponding to the 4 input control points instead of four output weight number  $w$ . For this situation, we should leave the number out and consider the eight parameters in the martrix.

The solution can be shown as follow:

$$x_{out} = \frac{x'_{out}}{w} = \frac{ax_{in} + by_{in} + c}{gx_{in} + hy_{in} + 1} \rightarrow x_{out} = ax_{in} + by_{in} + c - gx_{in} - hy_{in}$$

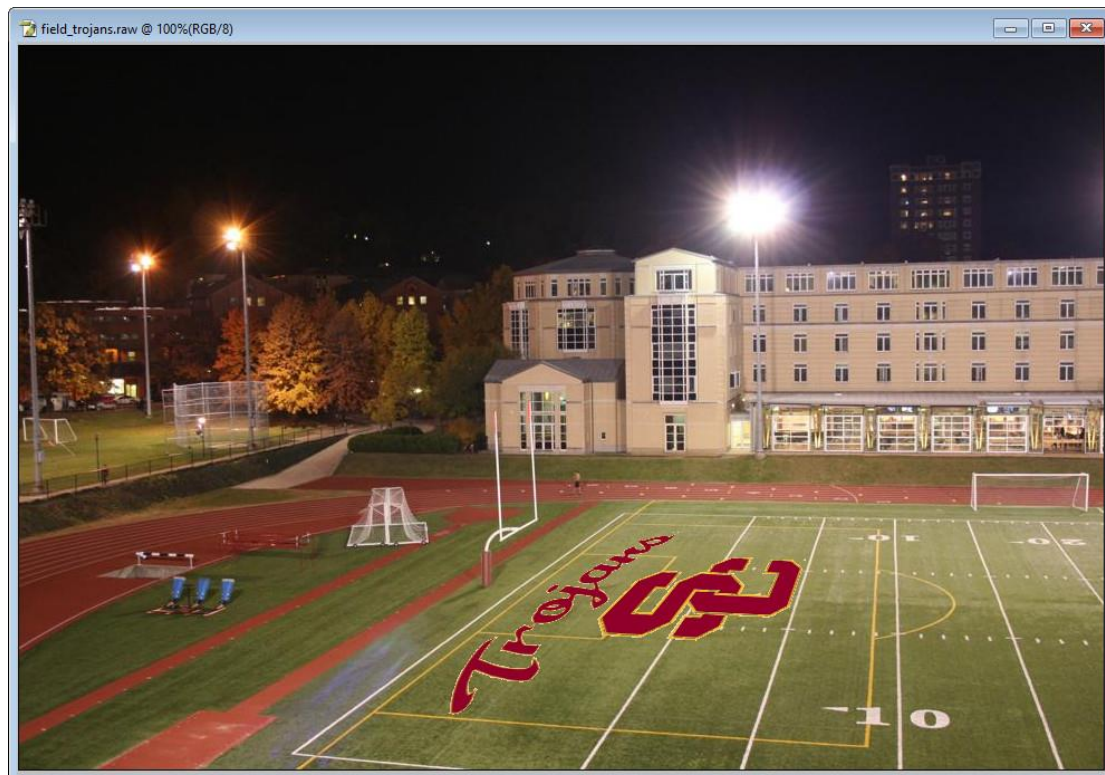
$$y_{out} = \frac{y'_{out}}{w} = \frac{dx_{in} + ey_{in} + f}{gx_{in} + hy_{in} + 1} \rightarrow y_{out} = dx_{in} + ey_{in} + f - gx_{in} - hy_{in}$$

For one control point, there are two functions. Therefore, for the four control points, there are eight functions, which is adequate for us to solve the eight parameters.

Having got the matrix, we can warp the patch image into the background easily. However, we should be careful because the background in the patch is not purely white, which means that we have to set the threshold value to leave out the white background.

### Experimental Results

We can get the field image after adding the Trojans patch



The combination of Trojans and Field

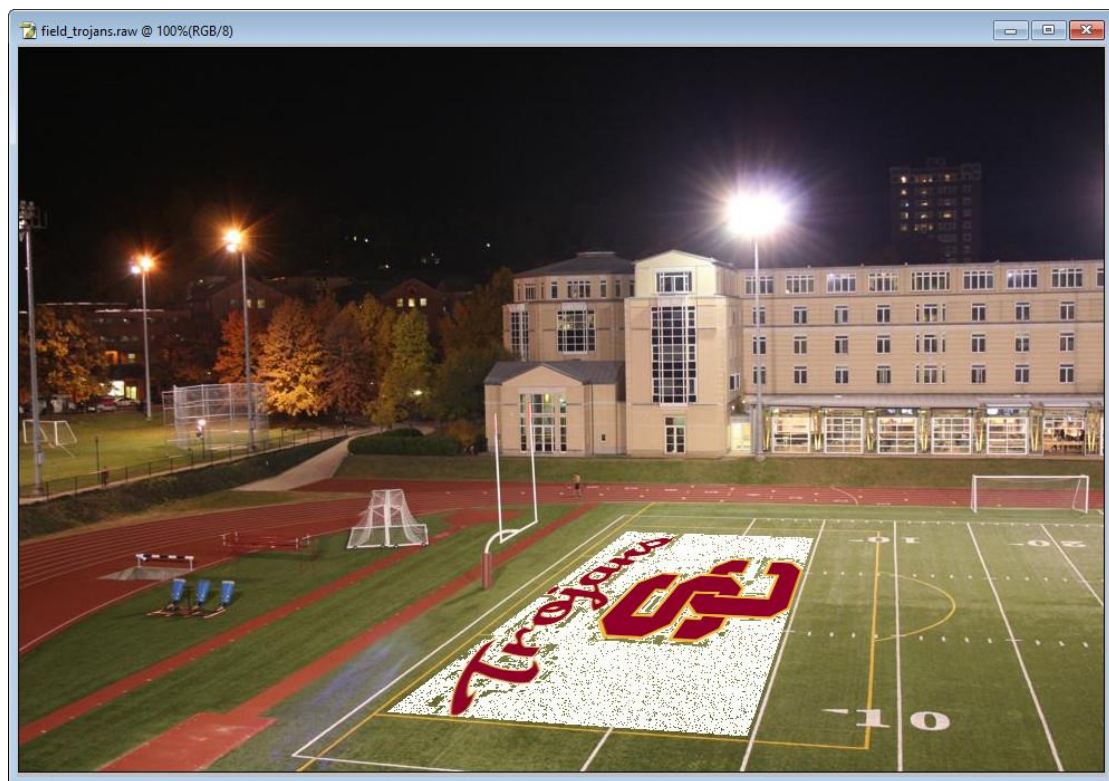
When conducting the operation, we can just input the desired output pixel point and use the MATLAB to calculate the transformation matrix as the procedures above. Then, we can put our patch into the field background at once.

## Discussion

In this question, there are some issues to discuss.

First, we have to deal with the rectangle in one operation. As a result, it will be complicated if I use the vector angle procedure in the Problem 1(a) to classify whether the point is in the area we should add the patch. So I propose another method to classify each points. We can do the transformation for all the points in the background; then, we can do a judgement: whether the transformed pixel is located in the area of patch. If so, we can say the pixel is the one we want to transform; if not, we keep the value of the pixel unchanged.

Another question is the threshold value that can be used to leave out the background part in the patch image. Because the background is not pure white, we should not decide the 255 as the threshold value, otherwise we can get the result as follow:

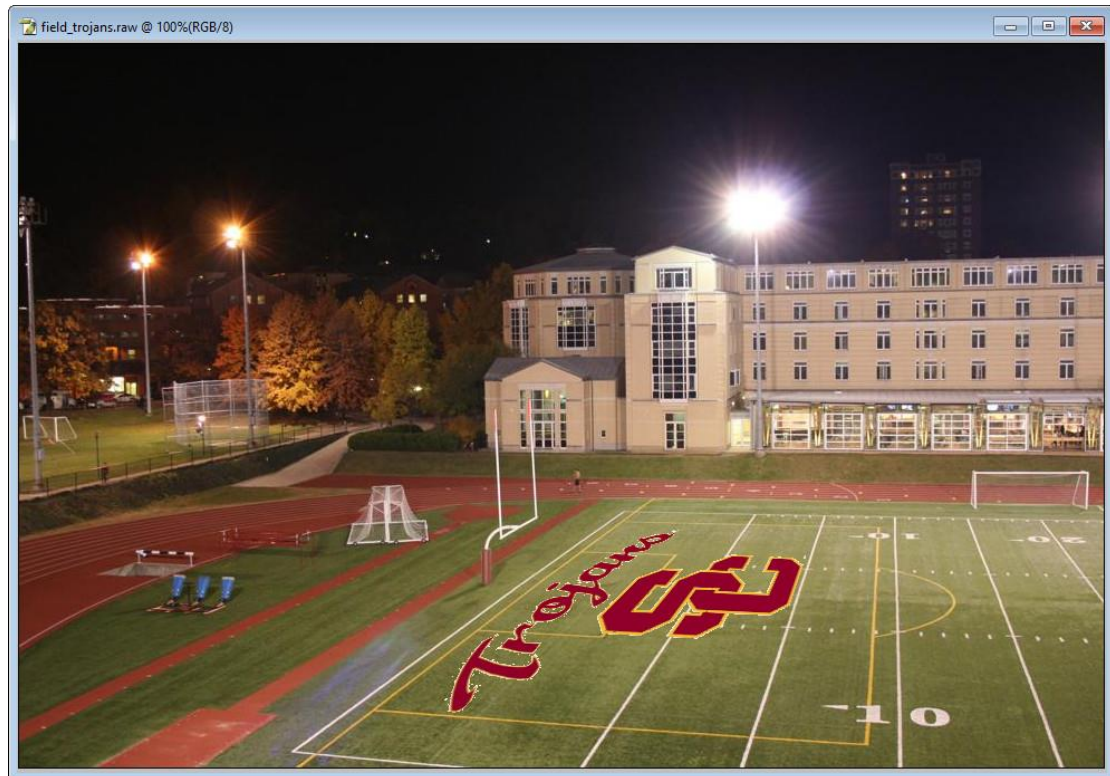


Use the 255 as the threshold value

We can find that most of the background in the patch has not been deleted.

If we use the 250 as the threshold value, the result can be shown like this:

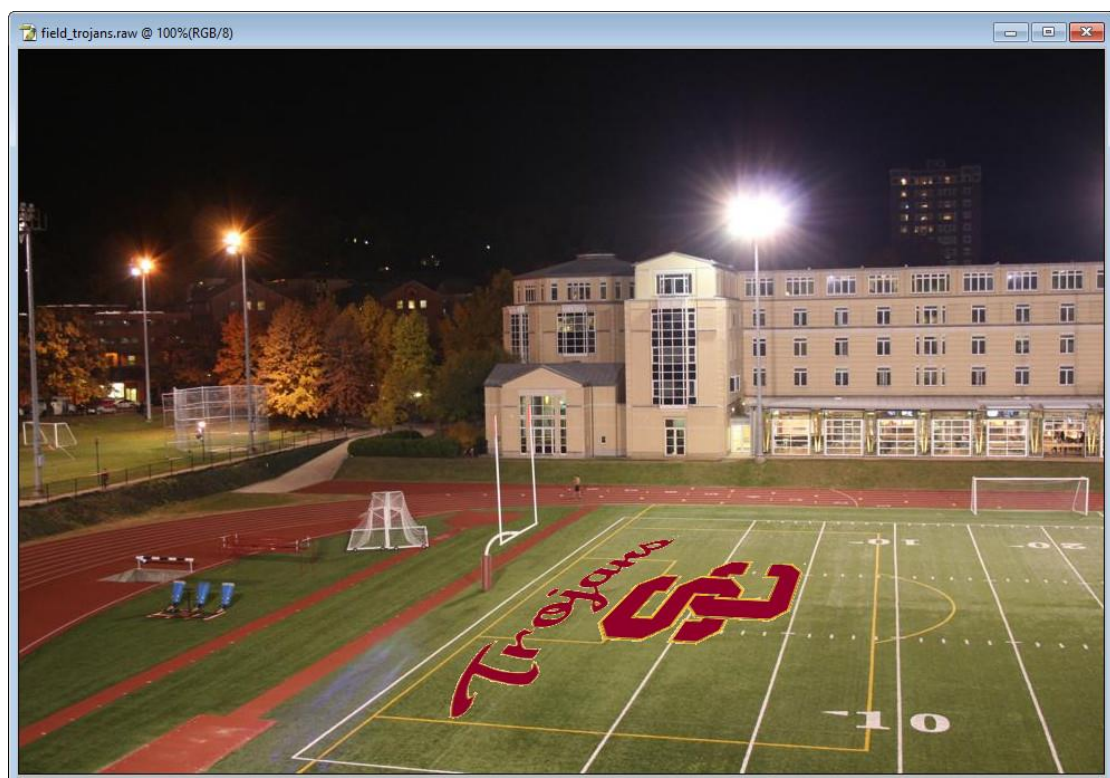




Use the 250 as the threshold value

Even though there are some white speckles relevant to the background of the patch, we can delete most of the background successfully.

Then, we use 245 as the threshold value, we can get the image:



Use the 245 as the threshold value

From the image, we can find that the white speckle has been removed completely. However, there are

some margin message has been deleted.

## Problem 2: Digital Halftoning (30 %)

### (a) Dithering Matrix (Basic: 15%)

#### Motivation

Halftone can convert a gray image, which is represented by 256 kinds of pixel values, into another one, which is represented by only two pixel values: 0 and 255. In this question, we will use the Bayer matrix to do the halftone processing.

A basic format of Bayer matrix can be shown as follow:

$$I_2(i, j) = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}$$

The less the number is, the more likely the pixel will have a white dot. For example, if the region has the pixel less than 127 but more than 56 or close to the 255, we can set the pixel value as follow:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

If we want to build larger Bayer matrix, we can use the formula:

$$I_{2n}(i, j) = \begin{bmatrix} 4I_n(i, j) & 4I_n(i, j) + 2 \\ 4I_n(i, j) + 3 & 4I_n(i, j) + 1 \end{bmatrix}$$

Then, we can use the formula below to define the output of a specific pixel point:

$$G(i, j) = \begin{cases} 1 & \text{if } F(i, j) > T(i \bmod N, j \bmod N) \\ 0 & \text{otherwise} \end{cases}$$

In the equation, the  $N$  is the size of the Bayer matrix,  $F(i, j)$  and  $G(i, j)$  is the input and the output, and  $T(x, y)$  can be defined as follow:

$$T(x, y) = \frac{I(x, y) + 0.5}{N^2} \times 255$$

#### Approach and Procedures

The key element in the question is to build the Bayer matrix. In this question, I use the irritation method to build the matrix we want. Because the Size of the Bayer matrix is  $2^{(n)} \times 2^{(n)} = 2^{2n}$ , I will use  $n$  to represent the Bayer matrix  $I_{2n}(i, j)$ . Then, starting from the  $I_2(i, j)$ , I will build any size of Bayer matrix we want.

The code for the Bayer Matrix can be shown as follow:

```
void Bayer(int N, double *a)
{
    double N0, N1;
    N1 = 2 * N;
    N0 = 2 << N;
    a[0] = 0;
    double temp[1024] = { 0 };
    for (int n = 0; n < N; n++)
    {
        int Size = sqrt(1 << (2*n));
        cout << Size << endl;
        for (int s = 0; s < Size; s++)
        {
```

```

        for (int t = 0; t < Size;t++)
        {
            a[Size * 2 * s + t] = 4 * temp[Size * s + t];
        }
    }
    for (int s = 0; s < Size;s++)
    {
        for (int t = 0; t < Size;t++)
        {
            a[Size * 2 * s + (t + Size)] = 4 * temp[Size * s + t] + 2;
        }
    }
    for (int s = 0; s < Size;s++)
    {
        for (int t = 0; t < Size;t++)
        {
            a[Size * 2 * (s + Size) + (t)] = 4 * temp[Size * s + t] + 3;
        }
    }
    for (int s = 0; s < Size;s++)
    {
        for (int t = 0; t < Size;t++)
        {
            a[Size * 2 * (s + Size) + (t + Size)] = 4 * temp[Size * s + t] + 1;
        }
    }
    for (int j = 0; j < 2 * 2 * Size * Size; j++)
    {
        temp[j] = a[j];
    }
}

```

Then, use the formula in the motivation, we can finally reconstruct the image represented by only two pixel values: 0 and 255.

### Experimental Results

The result by  $I_2(i, j)$  and  $I_8(i, j)$  matrix can be shown as follow:





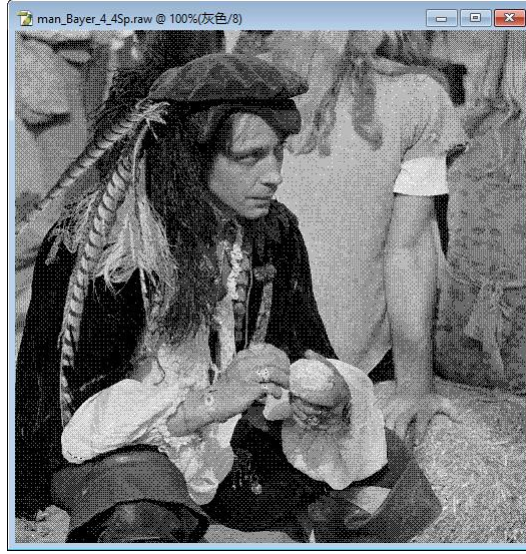
Use the  $I_2(i, j)$  (Left) and  $I_8(i, j)$  (Right) to conduct the halftone processing.

The result by Bayer Matrix  $I_4(i, j)$  and different Bayer Matrix  $4 \times 4$   $A_4(i, j)$  can be shown below:



Use the  $I_4(i, j)$  (Left) and  $A_4(i, j)$  (Right) to conduct the halftone processing.

The result of four intensity levels can be shown below (I use the  $I_4(i, j)$  Bayer matrix):



**Result of four intensity levels (use the  $I_4(i, j)$  Bayer matrix)**

### Discussion

From the result of the Different Bayer Matrix  $I_2(i, j)$  and  $I_8(i, j)$ , we can find that the larger the size of the Bayer matrix is, the more details the output will preserve because we can use more pattern to represent the change of the pixel value in the origin image.

The Different Bayer Matrix can be shown below:

$$A_4(i, j) = \begin{bmatrix} 14 & 10 & 11 & 15 \\ 9 & 3 & 0 & 4 \\ 8 & 2 & 1 & 5 \\ 13 & 7 & 6 & 12 \end{bmatrix}$$

Comparing with two results conducted by  $I_4(i, j)$  and  $A_4(i, j)$  matrix, we can find that the image processed by the  $I_4(i, j)$  is a little darker than the one processed by the  $A_4(i, j)$ . In addition, the detail presented by both matrix is similar. What is more, from my perspective, the image processed by the  $I_4(i, j)$  is more natural than the one processed by the  $A_4(i, j)$  since the right image has some artifacts where the color does not change gradually. The reason is that the location most possible to place the light doe is really condensing, which will make such inconsistent place.

For the new question, because the output of the pixel value has four kinds, I decide to modify the method used in the former questions and do it. That is, we can use the Dithering matrix to calculate the output for different gray-scale intervals; then, we will add those output values together for each pixel value.

First, according to the four-gray levels: 0, 85, 170 and 255, we can divide input gray level into four groups: 0; 1 – 85; 86 – 170; 171 – 255. In addition, we will prepare for three image matrix A, B and C.

Then, we can discuss the situation:

If the gray value of input (we can define it as  $F(i, j)$ ) is 0, we can define  $A(i, j) = 0$ ,  $B(i, j) = 0$ ,  $C(i, j) = 0$

If the value of input is between the 1 and 85, we can define  $A(i, j) = F(i, j)$ ,  $B(i, j) = 0$ ,  $C(i, j) = 0$

If the value of input is between the 86 and 170, we can define  $A(i, j) = 85$ ,  $B(i, j) = F(i, j) - 85$ ,



$$C(i, j) = 0$$

If the value of input is between the 86 and 170, we can define  $A(i, j) = 85$ ,  $B(i, j) = 85$ ,  $C(i, j) = F(i, j) - 85 \times 2$

Then, for each matrix ABC, we can use Bayer Matrix and the formula below to get the output for each matrix:

$$G(i, j) = \begin{cases} 1 & \text{if } F(i, j) > T(i \bmod N, j \bmod N) \\ 0 & \text{otherwise} \end{cases}$$

In the equation, the  $N$  is the size of the Bayer matrix,  $F(i, j)$  and  $G(i, j)$  is the input and the output for each ABC matrix, and  $T(x, y)$  can be defined as follow:

$$T(x, y) = \frac{I(x, y) + 0.5}{N^2} \times 85$$

Use the method above, we can get the  $A_{out}(i, j)$ ,  $B_{out}(i, j)$ ,  $C_{out}(i, j)$  and we can get the final output for each pixel  $(i, j)$ :  $A_{out}(i, j) + B_{out}(i, j) + C_{out}(i, j)$ . Therefore, we can get the dithered image with only four gray values: 0, 85, 170 and 255. Compared with the other halftone image, we can find that this output image keeps much more details than other image.

The code can be shown below:

```
c = 2;
c0 = 4;
for (int i = 0; i < Size2; i++)
{
    for (int j = 0; j < Size1; j++)
    {
        if (Imagedata[1 * (Size1 * i + j) + 0] == 0)
        {
            Imagedatasub1[3 * (Size1 * i + j) + 0] = 0;
            Imagedatasub1[3 * (Size1 * i + j) + 1] = 0;
            Imagedatasub1[3 * (Size1 * i + j) + 2] = 0;
        }
        else if (Imagedata[1 * (Size1 * i + j) + 0] >= 1 && Imagedata[1 * (Size1 * i + j) + 0] <= 85)
        {
            Imagedatasub1[3 * (Size1 * i + j) + 0] = Imagedata[1 * (Size1 * i + j) + 0];
            Imagedatasub1[3 * (Size1 * i + j) + 1] = 0;
            Imagedatasub1[3 * (Size1 * i + j) + 2] = 0;
        }
        else if (Imagedata[1 * (Size1 * i + j) + 0] >= 86 && Imagedata[1 * (Size1 * i + j) + 0] <= 170)
        {
            Imagedatasub1[3 * (Size1 * i + j) + 0] = 85;
            Imagedatasub1[3 * (Size1 * i + j) + 1] = Imagedata[1 * (Size1 * i + j) + 0] - 85;
            Imagedatasub1[3 * (Size1 * i + j) + 2] = 0;
        }
        else if (Imagedata[1 * (Size1 * i + j) + 0] >= 171 && Imagedata[1 * (Size1 * i + j) + 0] <= 255)
        {
            Imagedatasub1[3 * (Size1 * i + j) + 0] = 85;
```

```

        Imagedatasub1[3 * (Size1 * i + j) + 1] = 85;

        Imagedatasub1[3 * (Size1 * i + j) + 2] = Imagedata[1 * (Size1 * i + j) + 0] - 85 - 85;

    }

}

Bayer(c, a); //construct the Bayer Matrix
for (int i = 0; i < c0; i++)
{
    for (int j = 0; j < c0; j++)
    {
        cout << a[c0 * i + j] << '\t';

    }

    cout << endl;
}

for (int i = 0; i < (1 << (2 * c)); i++)
{
    a[i] = a[i] / (1 << (2 * c)) * 85;
}

for (int i = 0; i < Size2; i++)
{
    for (int j = 0; j < Size1; j++)
    {
        int a0 = i % (2 * c), b0 = j % (2 * c);

        if (Imagedatasub1[3 * (Size1 * i + j) + 0] > a[c0 * a0 + b0])
        {
            Imagedatas1[3 * (Size1 * i + j) + 0] = 85;
        }
        else
        {
            Imagedatas1[3 * (Size1 * i + j) + 0] = 0;
        }
    }
}

for (int i = 0; i < Size2; i++)
{
    for (int j = 0; j < Size1; j++)
    {
        int a0 = i % (2 * c), b0 = j % (2 * c);

        if (Imagedatasub1[3 * (Size1 * i + j) + 1] > a[c0 * a0 + b0])
        {
            Imagedatas1[3 * (Size1 * i + j) + 1] = 85;
        }
    }
}

```

```

        else
        {
            Imagedatas1[3 * (Size1 * i + j) + 1] = 0;
        }
    }
}

for (int i = 0; i < Size2; i++)
{
    for (int j = 0; j < Size1; j++)
    {
        int a0 = i % (2 * c), b0 = j % (2 * c);
        if (Imagedatasub1[3 * (Size1 * i + j) + 2] > a[c0 * a0 + b0])
        {
            Imagedatas1[3 * (Size1 * i + j) + 2] = 85;
        }
        else
        {
            Imagedatas1[3 * (Size1 * i + j) + 2] = 0;
        }
    }
}

for (int i = 0; i < Size2; i++)
{
    for (int j = 0; j < Size1; j++)
    {
        Imagedata[1 * (Size1 * i + j) + 0] = Imagedatas1[3 * (Size1 * i + j) + 0] + Imagedatas1[3 * (Size1 * i + j) + 1] +
        Imagedatas1[3 * (Size1 * i + j) + 2];
    }
}

```

## (b) Error Diffusion

### Motivation

In this part, we will use the error diffusion method to obtain the half-toned image. For each pixel, we will convert it to be 0 or 255, then, we can get the deviation between the output value and input value. In addition, we will add the deviation to the pixel that will be processed in the future according to a specific ratio. As a result, we will move to the next pixel to do the same things.

In this part, we will use three kinds of error diffusion matrices: Floyd-Steinberg, JJN and Stucki, which can be shown as follow:

Floyd-Steinberg

$$\frac{1}{16} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix}$$

JJN

$$\frac{1}{48} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$$

Stucki

$$\frac{1}{48} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

### Approach and Procedures

For the three diffusion matrices, the basic method is similar. First, we should extend the edge  $(n - 1)/2$  times ( $n$  is the size of the error diffusion matrix). Then, for each scanned pixels, we will decide the relationship between the pixel value and 128, if larger, we decide the output as 255; else, we decide the output as 0. Then, we do the subtraction which is input pixel value minus output pixel value. Next, we will regard the pixel point above as the center, we add the proportion deviation for the local pixel points according to the relative position in the picture and the proportion number at the position of matrix. We will keep on doing the processing until all pixels has been dealt with.

### Experimental Results

Use the Floyd-Steinberg method:



**The halftone result by Floyd-Steinberg method**

Use the JJN method:





**The halftone result by JJN method**

Use the Stucki method:



**The halftone result by Stucki method**

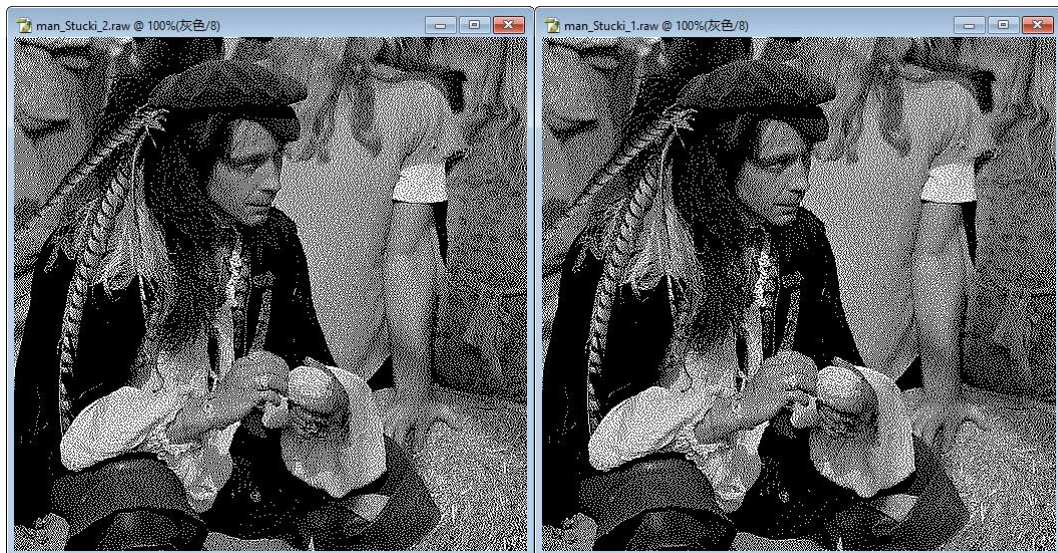
**Discussion**



From the result above, we can find that the JJN and the Stucki method can preserve much more detail than the Floyd-Steinberg. In addition, we can compare those image with the image gained from the dithering matrix, we can find that the image conducted by the error diffusion is much natural (few artifacts) than the one by dithering matrix and have less artifacts and it has more details than the image by dithering matrix. The reason is that the error diffusion will decide the next pixel according to the error before it automatically, while the output pixel of dithering matrix method can only be decided by the current pixels.

Also, we can discuss the affection of the limitation on the pixel value. As a matter of fact, when we add the deviation of the pixel value from the previous pixel points, I will make a justification whether it is more than 255 or less than 0. If it is more than 255, we define the value as 255. If it is less than 0, we define the value as 0. This is called limitation. If we do not use it, we will lose some important details since we use the unsigned char format, which can only represent the number from 0 to 255.

If I want to cancel the limitation, we have to use the short integer format to represent the pixels. As a result, I want to see whether the limitation has affection on the halftone image. The result of the limitation and no limitation can be shown below:



**The result of Stucki method with limitation (Left) and the one without limitation (Right)**

Therefore, we can find that there is little affection when canceling the limitation.

In addition, we can discuss some methods to improve the quality of the image. First method is to implement the Gaussian function to be the error diffusion matrix because from the Stucki and JJN matrix the larger the distance between the center and the neighbor point, the less the error weight is, and those parameters are not strict Gaussian function. Therefore, if we use the Gaussian function and set the value of  $\sigma$  carefully, we can get better error weight relevant to their distance. We can obtain  $\sigma$  by testing some values:





**The output image when  $\sigma = 0.5$  (Top Left),  $\sigma = 1$  (Top Right),  $\sigma = 1.5$  (Bottom Left),  $\sigma = 5$  (Bottom Right)**

From the image, we can conclude that the  $\sigma$  should not be too small because it will make the error diffusion matrix be close to the impulse matrix, which means that it will get the grayscale ramp image;  $\sigma$  should also not be too much since it will filter more details and make the image so smooth and get more artifacts. Then, we can find that is the  $\sigma$  is around 1.5, we can preserve much details and get less artifacts.

Another solution is to use four-level representation instead of two-level representation to represent the image. That is, we can use the output (0, 85, 170, 255) to represent the image of man like the last question of Problem 2 (a). If we implement the method (I use the Floyd-Steinberg method as the error diffusion method), we can get the image as follow (compared with old Floyd-Steinberg method):





**The result of Four-level pixel value method (Left) and Floyd-Steinberg method (Right)**

From the comparison, we can find that we can get image with better quality when using the same dithering method but 4-pixel value representation method. The reason is that it can preserve much more details than the origin Floyd-Steinberg method.

In addition, we can use the multiscale error diffusion to finish the halftoning process because it can preserve the contrast in the original image. In addition, it cannot oversmooth the image and the sharpness of the output image can be controlled by changing the size of diffusion filter.

### **Problem 3: Morphological Processing (30%)**

#### **(a) Shrinking (Basic: 7%)**

##### **Motivation**

In this question, we should finish three questions:

- Find a method to make the shrinking filter.
- Use the image passed through shrinking filter to count the number of the squares in the image.
- Use the Shrinking filter to decide the frequency of square sizes.

##### **Approach and Procedures**

For the first question, we should do two steps: choose the candidate to delete and final decision about the candidate points. For the primary step, if the pixel value is zero, we should define the output as zero, else, we should check if it hits the conditional patterns. If it hits, we regard it as the candidate; else, we define the output as 255; For the next step, we should check if each candidate point satisfies the unconditional pattern. If it hits, we define the final output as 255; else, we define the final output as zero.

For the second question, the answer is easy. We can shrink the image until there is no change any more. Since the final image for the squares is a series of single points, we can just get the total number of squares by counting the number of the single points.

For the frequency of each size, it will be not easy. At first, I will count the number of the single points before processing the shrinking operation. Then, after shrinking the image one time, I will count the exact number of the single points. When the shrinking finished, I just subtract the number with the previous

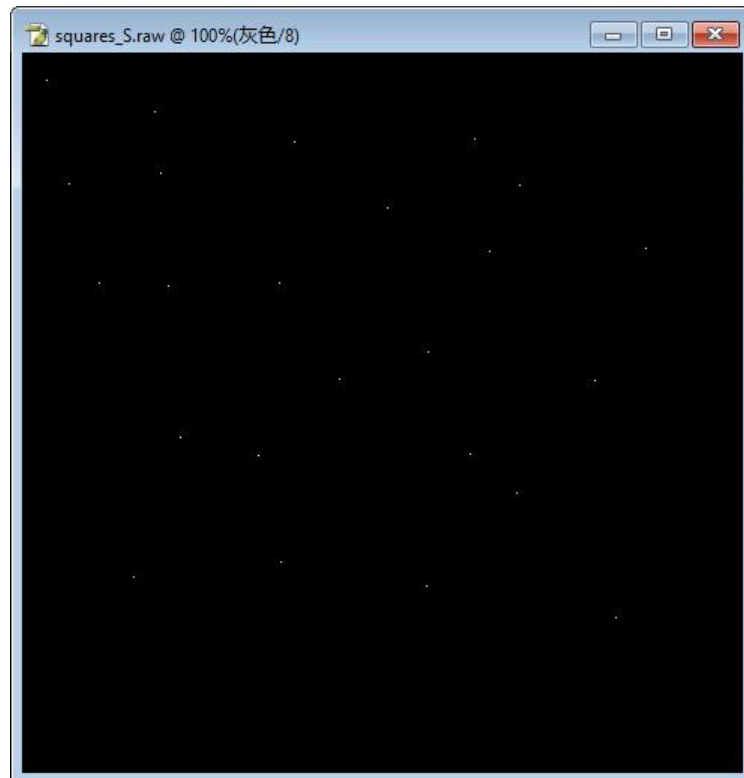
number of the single points, Then, we can get the size for each  $n$ th time by  $2 \times n + 1$  (if we define the primary count is zero time).

However, this method has a problem: Have not consider the presence of even-size squares. For the square with even size  $2 \times n$ , we have to do  $n$  time shrink to turn it to be a single point, and the  $n - 1$  time shrink will be a  $2 \times 2$  square. As a result, we should consider the number of even-size square. As a result, we should do some revision. The revision of method can be shown below:

- Count the number of the single points and the  $2 \times 2$  squares before processing the shrinking
- When shrinking the image one time, I will count the exact number of the single points and the  $2 \times 2$  squares
- If the number of single point in  $n$ th shrink operation is  $x$ , and the number of  $2 \times 2$  square in  $(n - 1)th$  time is  $y$ , we can get that: The number of  $2n \times 2n$  squares is  $y$ ; The number of  $(2n + 1) \times (2n + 1)$  squares is  $x - y$ .

### Experimental Results

The completed shrinking image will be shown as follow:



**The Shrinking image of Squares.raw**

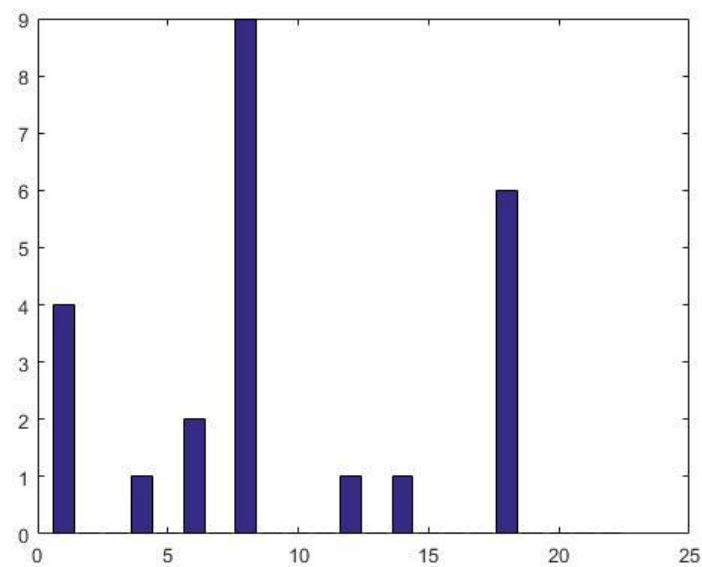
Use the method referred in the approach and procedure, I can get the total number of squares is 24 and the frequency of each size square can be shown as follow: (Use the result of program and histogram)

```

C:\Windows\system32\cmd.exe
D:\EE569_Assignment\1\C++\Image_Processing_HW2\Debug>D:\EE569_Assignment\1\C++\Image_Processing_HW2\Debug\Image_Processing_HW2.exe D:\EE569_Assignment\2\squares.raw D:\EE569_Assignment\2\squares_S.raw 1 480 480
Solve which problems: 3
solve which part: a
The total number is 24
The number of 1x1 size is 4
The number of 2x2 size is 0
The number of 3x3 size is 0
The number of 4x4 size is 1
The number of 5x5 size is 0
The number of 6x6 size is 2
The number of 7x7 size is 0
The number of 8x8 size is 9
The number of 9x9 size is 0
The number of 10x10 size is 0
The number of 11x11 size is 0
The number of 12x12 size is 1
The number of 13x13 size is 0
The number of 14x14 size is 1
The number of 15x15 size is 0
The number of 16x16 size is 0
The number of 17x17 size is 0
The number of 18x18 size is 6
The number of 19x19 size is 0
The number of 20x20 size is 0
The number of 21x21 size is 0
The number of 22x22 size is 0
Press any key to continue . . .

```

**The Result of the program for the Square.raw**



**The frequency of the size of squares**

## Discussion

For this part, I have some issues to discuss.

The first is the method to represent the conditional matrix and unconditional matrix. For the conditional matrix, I just use the binaries to represent the matrix since the elements in the matrix is confirmed. If the pixel value we want to clarify is 255, the representation can be shown as follow (the left is the pixel values around the pixel point we want to solve, and the right is the converted binaries):

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & \text{Center} & 5 \\ 6 & 7 & 8 \end{bmatrix} \rightarrow [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8]$$

For example, we have such patterns

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

We can convert it to be the binaries  $(10011010)_2$

However, for the unconditional matrix used in the identification of the candidate points, it will be hard for us to use only the binaries since there are a series of uncertain points in the unconditional matrix. As a result, I will use both the binaries and logical representations to do the justification.

If the unconditional matrix has no uncertainty, we can use the binaries just as the example below:

$$\begin{bmatrix} 0 & 0 & M \\ 0 & M & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow (00100000)_2$$

If there are some uncertain points, we can use the logical representation to present it just as the example below (in the question, each D can be 0 or 1; at least one of A, B and C is 1):

$$\begin{bmatrix} M & D & M \\ D & M & D \\ A & B & C \end{bmatrix}$$

If we use the representation below to present the location, we can get that

$$\begin{bmatrix} F(-1,-1) & F(-1,0) & F(-1,1) \\ F(0,-1) & M & F(0,1) \\ F(1,-1) & F(1,0) & F(1,1) \end{bmatrix}$$

Then, we can use the code below to represent the logical relationship:

$$if(F(-1,-1) = 1 \ \&\& \ F(-1,1) = 1 \ \&\& \ (F(1,-1) = 1 \ or \ F(1,0) = 1 \ or \ F(1,1) = 1))$$

The another question is the identification of the single point and  $2 \times 2$  squares. In this question, I decide to use the hit and miss matrix above to identify it. If we hit the matrix below:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

We can regard the pixel as a single point. Also, if the pixel hit the matrix:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

And the pixel 2-pixel south east of the hit another matrix:

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

We can regard that there is a  $2 \times 2$  squares.

The function of shirking: to shrink the image to be a single pixel if the image has no holes or a connected ringing structure where it has linking lines between each pair of holes. Therefore, standard squares can be shrunk as single points.

### (b) Thinning (Basic: 7%)

#### Motivation

In this question, we should finish two questions:

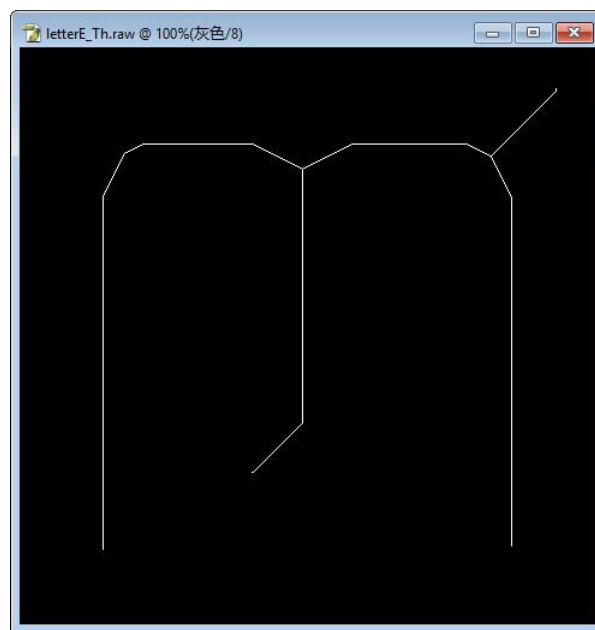
- Find a method to build the Thinning filter.

- Use the Thinning filter to get the final result of the Letter E.

### Approach and Procedures

For the question, we should do the procedure similar with the shrinking process. We will do two steps: choose the candidate to delete and final decision about the candidate points. For the primary step, if the pixel value is zero, we should define the output as zero, else, we should check if it hits the conditional patterns. If it hits, we regard it as the candidate; else, we define the output as 255; For the next step, we should check if each candidate point satisfies the unconditional pattern. If it hits, we define the final output as 255; else, we define the final output as zero. Please note that the conditional matrices and unconditional matrices in Thinning are different from the ones in Shrinking.

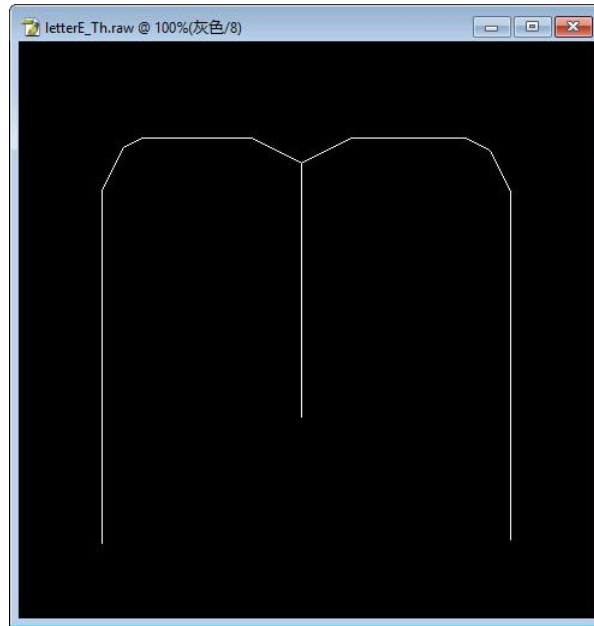
### Experimental Results



**The result of thinning of Letter E**

### Discussion

From the definition of the Thinning, we can get that the pixel will be preserved to be a stroke equidistant from its nearest outer boundaries. Theoretically, the output of E will have no strange fold lines at the end of vertical line shown in the picture. However, there are two fold lines. The reason is that there is an inconsistent point in that corner, which changes the outer boundaries. As a result, there will be two fold lines in the output image. If we just filter those inconsistent points in the corner, we can get the perfect E as follow:



**The result of thinning of Letter E after filtering the inconsistent point**

**(c) Skeletonizing (Basic: 8%)**

**Motivation**

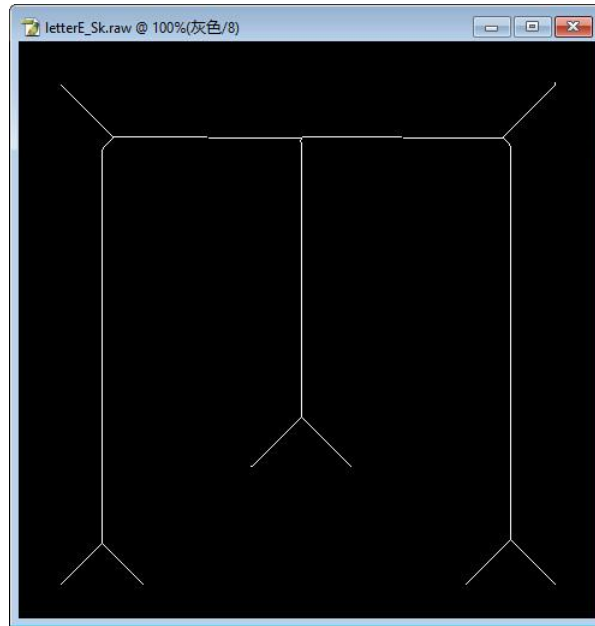
In this question, we should finish two questions:

- Find a method to build the Skeletonizing filter.
- Use the Skeletonizing filter to get the final result of the Letter E.

**Approach and Procedures**

For the question, we should do the procedure similar with the shrinking process. We will do two steps: choose the candidate to delete and final decision about the candidate points. For the primary step, if the pixel value is zero, we should define the output as zero, else, we should check if it hits the conditional patterns. If it hits, we regard it as the candidate; else, we define the output as 255; For the next step, we should check if each candidate point satisfies the unconditional pattern. If it hits, we define the final output as 255; else, we define the final output as zero. Please note that the conditional matrices and unconditional matrices in Skeletonizing are different from the ones in Shrinking.

**Experimental Results**



**The result of Skeletonizing of Letter E**

### **Discussion**

From the definition of Skeletonizing, we can get that the Skeletonizing image can be regarded as the bone of the image. That is, we can know not only the shape of image, but the width of the origin image. In the question, we can know the width of each edge and the shape of the letter E from the output image.

### **(d) Counting game (Advanced: 8%)**

#### **Motivation**

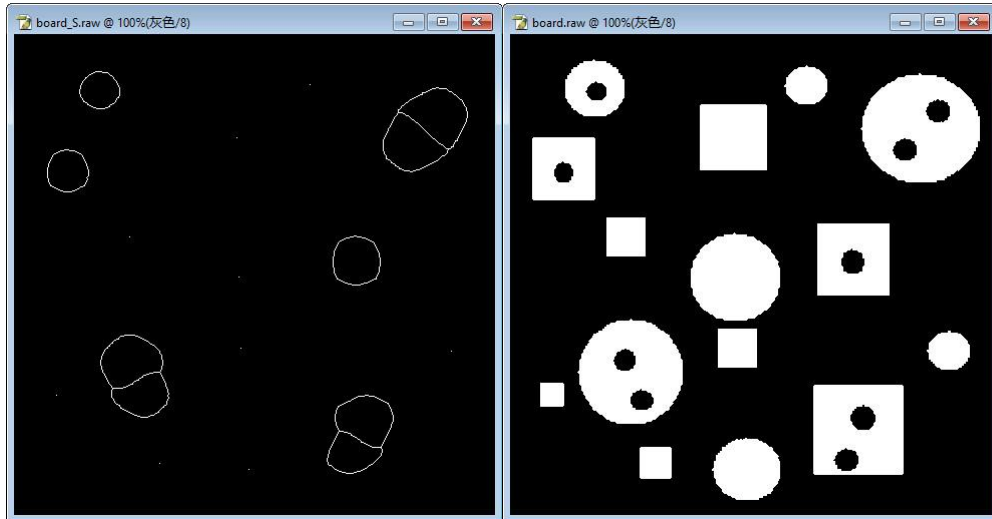
In this question, we should do four questions:

- Find the number of all white objects in the image;
- Find the number of holes in the image;
- Find the number of all white square holes (with or without holes);
- Find the number of all white circle holes (with or without holes);

#### **Approach and Procedures**

For counting the number of objects, we can get the idea of shrinking which can be used to make the image into single point and make the counting easier. However, there are some patterns with some holes. According to the definition of the shrinking, the hole can make the origin image be a circle, which can be shown below:





**The shrinking image (Left) and Origin one (Right) for the board.raw**

As a result, we should do some procedure to clear the holes in the image. Then, I make an important **assumption**:

**The diameter of the holes in the image is less than 30 pixels.**

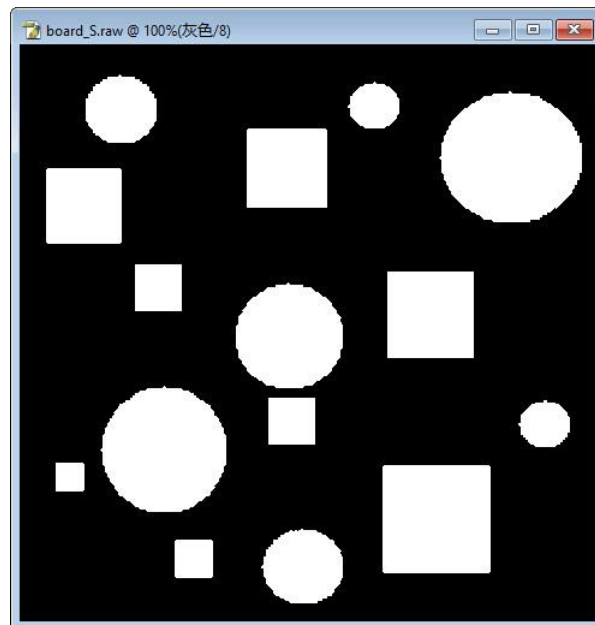
Then, I design a method:

When scanning pixel by pixel and get to the point  $(i, j)$ , we make a justification: if such five points has such properties:  $F(i, j)$  is the pixel value at the point)

$$F(i, j) = 1 \text{ and } F(i + 15, j + 15) = 1 \text{ and } F(i, j + 30) = 1 \text{ and } F(i - 15, j + 15) = 1 \text{ and } F(i, j + 15) = 0$$

If so, we will do such operation:

Scan through this area: square  $(i - 15, j)$ ,  $(i - 15, j + 30)$ ,  $(i + 15, j)$ ,  $(i + 15, j + 30)$  row by row. For each row, if the two endpoints of the row are not all 1, we skip to the next row; else, when there are pixels whose value is 0, we convert it to be 1 and change the value in both the input and the output image. When the operation completed, we can get the image as follow:



**The board image filtered the holes**

Then, we can use the shrinking filter to get the total number of the white objects successfully.

For searching the holes, we can use the method as follows:

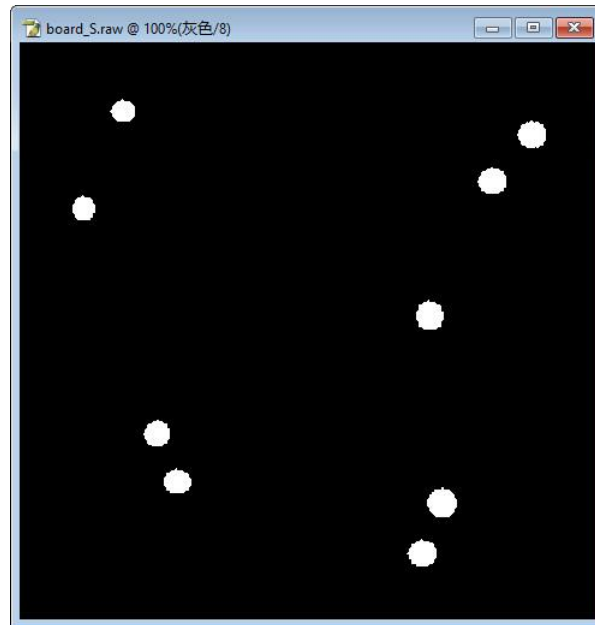
When scanning pixel by pixel and get to the point  $(i, j)$ , we make a justification: if such five points has such properties: ( $F(i, j)$  is the pixel value at the point)

$$F(i, j) = 1 \text{ and } F(i + 15, j + 15) = 1 \text{ and } F(i, j + 30) = 1 \text{ and } F(i - 15, j + 15) = 1 \text{ and } F(i, j + 15) = 0$$

If so, we will do such operation:

Scan through this area: square  $(i - 15, j)$ ,  $(i - 15, j + 30)$ ,  $(i + 15, j)$ ,  $(i + 15, j + 30)$  row by row. For each row, if the two endpoints of the row are not all 1, we skip to the next row; else, when there are pixels whose value is 0, we convert it to be 0.5 and change the value in both the input and the output image.

When this operation completed, should do the transformation: if the pixel value is 0.5, we convert it to be 1; else, we convert it to be 0. The result is follow:



**The hole image in the board.raw**

For the third and fourth question, I will use the Circularity C to identify, and the formula is

$$C = \frac{4\pi A}{P^2}$$

In the equation, A is the area of an object in the image; the P is the length of the perimeter of the object. For example, the area of a circle is  $\pi r^2$ , and the length of the perimeter of the object is  $2\pi r$ . Therefore, the C is  $4\pi \times \pi r^2 / (2\pi r)^2 = 1$

Therefore, I will use the Circularity to justify the shape of an object. In this question, because the circle is not perfect. For such question, I use another **assumption**:

**The only shape in the picture is the circle and square.**

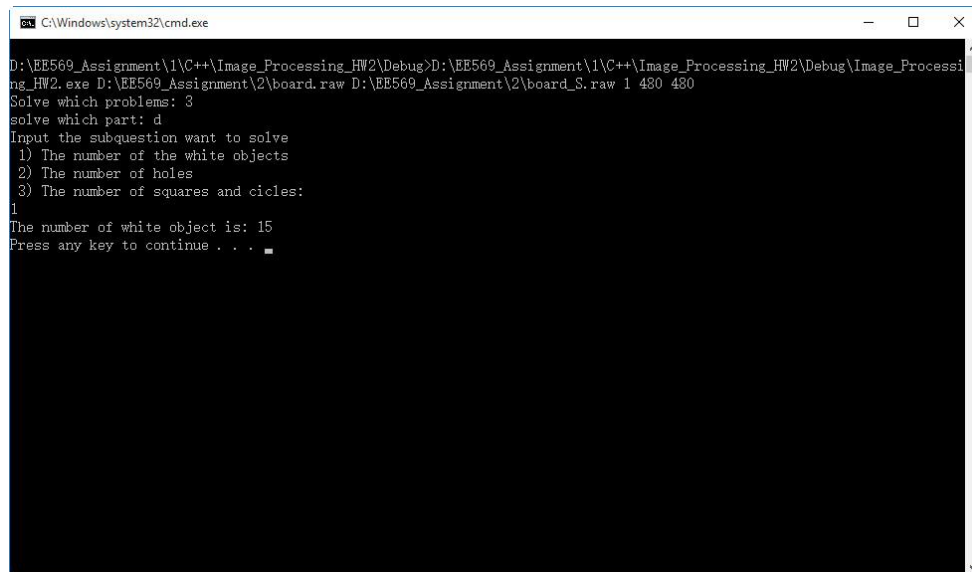
Therefore, I will use the square Circularity, which is less than 0.8, to justify if it is square. If it is less than 0.8, we can regard it as square; else, we can regard it as circle.

The method to get the area, length of the perimeter and Circularity will be discussed in the discussion

section.

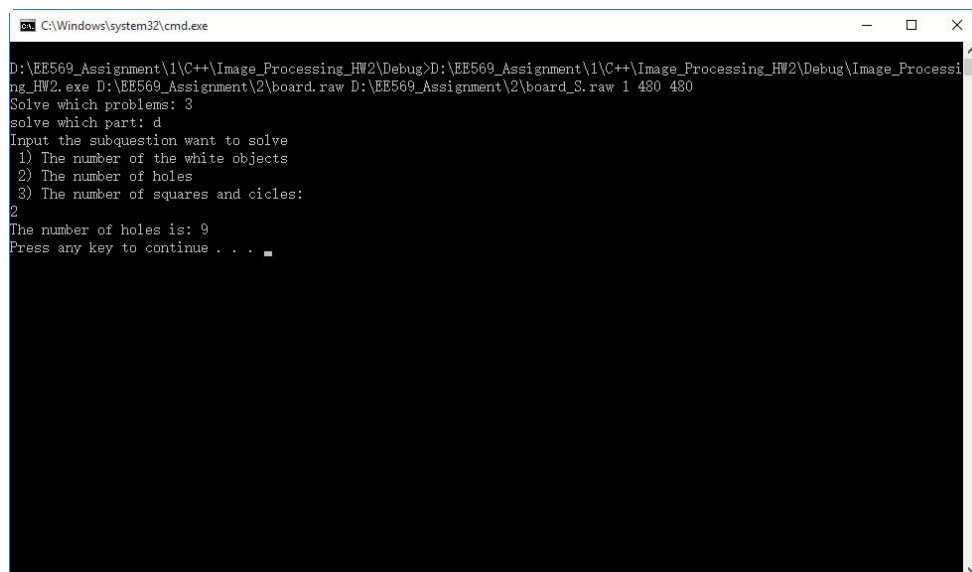
## Experimental Results

The result of problem 1 is



```
C:\Windows\system32\cmd.exe
D:\EE569_Assignment\1\C++\Image_Processing_HW2\Debug>D:\EE569_Assignment\1\C++\Image_Processing_HW2\Debug\Image_Processing_HW2.exe D:\EE569_Assignment\2\board.raw D:\EE569_Assignment\2\board_S.raw 1 480 480
Solve which problems: 3
solve which part: d
Input the subquestion want to solve
1) The number of the white objects
2) The number of holes
3) The number of squares and circles:
1
The number of white object is: 15
Press any key to continue . . .
```

The result of problem 2 is



```
C:\Windows\system32\cmd.exe
D:\EE569_Assignment\1\C++\Image_Processing_HW2\Debug>D:\EE569_Assignment\1\C++\Image_Processing_HW2\Debug\Image_Processing_HW2.exe D:\EE569_Assignment\2\board.raw D:\EE569_Assignment\2\board_S.raw 1 480 480
Solve which problems: 3
solve which part: d
Input the subquestion want to solve
1) The number of the white objects
2) The number of holes
3) The number of squares and circles:
2
The number of holes is: 9
Press any key to continue . . .
```

The result of problem 3 and 4 is

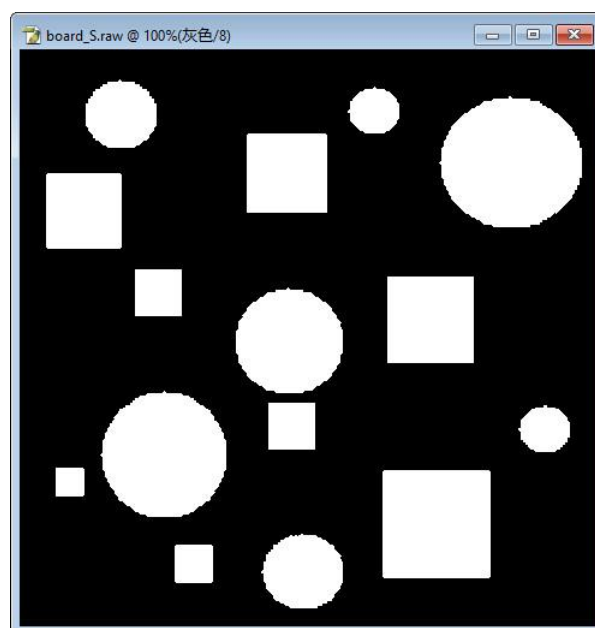
```

C:\Windows\system32\cmd.exe
D:\EE569_Assignment\1\C++\Image_Processing_HW2\Debug>D:\EE569_Assignment\1\C++\Image_Processing_HW2\Debug\
ng_HW2.exe D:\EE569_Assignment\2\board.raw D:\EE569_Assignment\2\board_S.raw 1 480 480
Solve which problems: 3
solve which part: d
Input the subquestion want to solve
1) The number of the white objects
2) The number of holes
3) The number of squares and circles:
3
The number of white object is: 15
The number of white circles is: 7
The number of white squares is: 8
Press any key to continue . . .

```

## Discussion

In this section, I will talk about the method for question 3 and 4 in detail. I will use the image below to solve the question.



The most important problem is to extract each image pattern from the board image separately. As a result, I use a series of steps to solve the problem.

First, I will use the hit and miss matrix to identify and get the edge image. In an image, if an edge pixel is connected with both the side edge pixels and corner edge pixels, I will regard the corner pixels as the edge. That is, if we get the image which is

|   |   |   |   |   |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

We will get the edge picture which will be

```

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

```

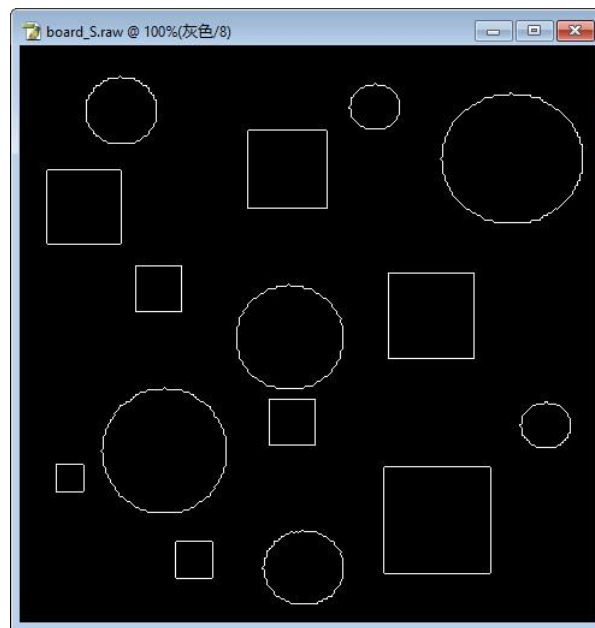
Therefore, the mask matrix is (if hit, remove the pixel value)

```

D 1 D
1 1 1
D 1 D

```

Use this method, I will get the edge element of the image:



Second, I will identify which pixel value is 1. If I get such pixel, I will stop to do another operation.

In this part, I will have another assumption:

**The first scanned point whose pixel value is 1 is on the top of the image pattern.**

- Save the pixel coordinate in an array and make the pixel value in the image as zero.
- Check the neighbor pixel from one to eight in the ascending order. If one situation is satisfied, we will stop at this pixel and move to this pixel, the justify order is shown below (the number n in the matrix is the nth pixel we will check, so it is an order of clockwise):

```

6   7   8
5  Center  1
4   3   2

```

- Do the operation until there is no satisfying pixel point.

Then, we can scan the image again and do the same thing in the second section (use another array to save the coordinates of the edge). When we cannot find any white points in the picture, we can stop the action and do the next section.

Third, for each pixel coordinate in the array, I will scan the array to get the coordinate pairs with the same row number, Then, we can get the maximum and minimum column number for the given row

number. Then, we can do the calculation:  $j_{max_i} - j_{min_i} + 1$  for the specific row  $i$ . When we do the calculation for all the row in the picture, we can get the result below:

$$\sum_{for\ the\ all\ i\ row} j_{max_i} - j_{min_i} + 1$$

This is the area of the image.

Forth, I will use the edge coordinates saved in the arrays to check the image which is the one at the beginning of the discussion. For a pattern in the picture, if we use the coordinates to check the edge and find that it hits the matrices (D will be 0 or 1):

$$\begin{array}{cccccccccccc} D & 0 & D & 1 & 1 & D & 1 & 1 & 1 & D & 1 & 1 \\ 1 & 1(center) & 1 & 1 & 1(center) & 0 & 1 & 1(center) & 1 & 0 & 1(center) & 1 \\ 1 & 1 & 1 & 1 & 1 & D & D & 0 & D & D & 1 & 1 \end{array}$$

We will add 1 for the length of the perimeter.

If we use the coordinates to check the edge and find that it hits the matrices (D will be 0 or 1):

$$\begin{array}{cccccccccccc} D & 0 & 0 & 1 & 1 & D & D & 1 & 1 & 0 & 0 & D \\ 1 & 1(center) & 0 & 1 & 1(center) & 0 & 0 & 1(center) & 1 & 0 & 1(center) & 1 \\ 1 & 1 & D & D & 0 & 0 & 0 & 0 & D & D & 1 & 1 \end{array}$$

We will add 2 for the length of the perimeter.

If we use the coordinates to check the edge and find that it hits the matrices (D will be 0 or 1):

$$\begin{array}{cccccccccccc} 0 & 0 & 0 & D & 1 & D & D & 0 & 0 & 0 & 0 & D \\ 0 & 1(center) & 0 & 0 & 1(center) & 0 & 1 & 1(center) & 0 & 0 & 1(center) & 1 \\ D & 1 & D & 0 & 0 & 0 & D & 0 & 0 & 0 & 0 & D \end{array}$$

We will add 3 for the length of the perimeter.

Then, we can get the length of the perimeter (P) during operations above. Then, with the value of area (A), we can use the circularity C to clarify the shape of the image. If C is less than 0.8, we regard it as the rectangular; else, we regard it as circle. Therefore, we can finally get the exact total number of circles and squares.