EE 569 Homework #2

Prof. C.-C. Kuo

Hrishikesh Hippalgaonkar

hippalga@usc.edu

Table of Contents:

1. Problem 1

1.1 Abstract and Motivation:

Most of the image processing operations involve the change of the pixel intensity. But *Geometric image transformation* changes the pixel location. And this is implemented while preserving the pixel neighbourhood and the pixel intensity. Let (u, v) represent the image coordinate in the original image and (x, y) be the image coordinates in the warped image. We can use a function to relate the corresponding pixels in the two images:

$$\boldsymbol{Forward\ mapping}: \begin{cases} \boldsymbol{x} = \boldsymbol{f_1(u,v)} \\ \boldsymbol{y} = \boldsymbol{f_2(u,v)} \end{cases}$$

$$\boldsymbol{Inverse\ mapping}: \begin{cases} \boldsymbol{u} = \boldsymbol{g_1(x,y)} \\ \boldsymbol{v} = \boldsymbol{g_2(x,y)} \end{cases}$$

Given that we know the mapping function, we can easily warp the image

Homographic transformation is a general case of affine transformation. It relates the pixel co-ordinates in an image to the pixel coordinates in another image.

A homography can be defined as a transformation of plane which changes the perspective, i.e. it re-projects a plane viewed from one camera into a different camera view.

Two images are related by a homography iff:

- Both images are viewing the same plane from a different angle
- Both images are taken from the same camera but from a different angle (Camera is rotated about its centre of projection without any translation)

It should be noted that homography relationship is independent of the scene structure. It does not depend on what the cameras are looking at, or even what is seen in the image.

1.2 Approach and Procedure:

1.2.1 Geometric Image Modification:

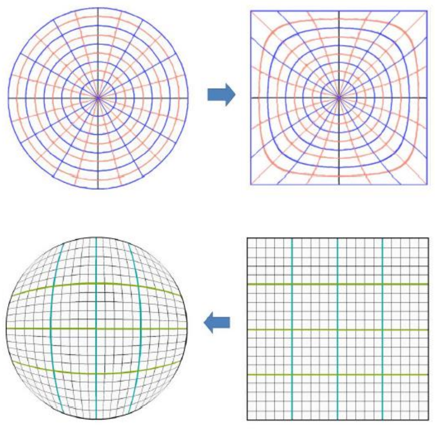


Figure 1. Warping a circle to square and warping a square to circle

If (x, y) are the pixel coordinates in the square and (u, v) are the pixel coordinates in the circle, then mapping function for warping a circle to a square is given by:

$$\boldsymbol{x = 0.5 * \sqrt{2 + u^2 - v^2 + 2\sqrt{2}u} - 0.5 * \sqrt{2 + u^2 - v^2 - 2\sqrt{2}u}}$$

$$\boldsymbol{y = 0.5 * \sqrt{2 - u^2 + v^2 + 2\sqrt{2}v} - 0.5 * \sqrt{2 - u^2 + v^2 - 2\sqrt{2}v}}$$

The mapping function for warping a square to a circle is given by:

$$\boldsymbol{u = x\sqrt{1 - \frac{y^2}{2}} \qquad v = y\sqrt{1 - \frac{x^2}{2}}}$$

Using bilinear interpolation and the above mapping function, we can easily warp a square image to circle image.

1.2.2 Homographic Transformation and Image Stitching:

Assume $\boldsymbol{F_1}$ is the original image and $\boldsymbol{F_2}$ is the transformed image. First, we need to locate four control points in both the images. Control points will be unique pixels which can be recognised as the same in both the images. The points in the image (the common points) will be related by a homographic transformation $\boldsymbol{H}$. This can be mathematically represented as

$$\boldsymbol{F_2 = HF_1}$$

The homography matrix can be represented by

$$\begin{bmatrix} x'_2 \\ y'_2 \\ w'_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \; and \; \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \frac{x'_2}{w'_2} \\ \frac{y'_2}{w'_2} \end{bmatrix}$$

Which can also be written as,

$$x_2 = \frac{H_{11}\,x_1 + H_{12}\,y_1 + H_{13}}{H_{31}\,x_1 + H_{32}\,y_1 + H_{33}} \;\; and \;\; y_2 = \frac{H_{21}\,x_1 + H_{22}\,y_1 + x_{23}}{H_{31}\,x_1 + H_{32}\,y_1 + H_{33}}$$

Where $\boldsymbol{x_1, y_1}$ are the pixel location of the control point in the original image and $\boldsymbol{x_2, y_2}$ are the pixel location in the transformed image

Assuming that we have selected the control points, we need to compute the $\boldsymbol{H}$ matrix. For simplicity we assume $\boldsymbol{H_{33} = 1}$. For each of the four control points pairs, we get a total of 8 linear equations. These linear equations can be solved to get the $\boldsymbol{H}$ matrix.

For the give problem of stitching the *left.raw, middle.raw* and *right.raw* images, I have initially placed the *middle.raw* image in the centre of a larger blank black image. For the control point that I have chosen for the *middle.raw* I have added a offset value which is equal to the offset of the *middle.raw* image from the left edge of the blank image in which it is placed. For the *left.raw* and *right.raw,* the control points are chosen as they are i.e. without adding any offset. The control points for *left.raw* and *middle.raw* are:

Control Points in *left.raw* image (column, row)	**Corresponding control points in *middle.raw* (column, row)**
473, 282	482, 822
424, 338	424, 883
362, 350	360, 895
204, 293	195, 831

Similarly, the control points for *right.raw* and *middle.raw* are:

Control Points in *right.raw* image (column, row)	**Corresponding control points in *middle.raw* (column, row)**
366, 258	372, 1264
525, 258	544, 1262
436, 125	436, 1119
224, 47	237, 1049

To get the $\boldsymbol{H}$ matrix we must solve the eight linear equations. This is represented mathematically as

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1X_1 & -y_1X_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_1X_1 & -y_1X_1 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_1X_1 & -y_1X_1 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_1X_1 & -y_1X_1 \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix}$$

Where lowercase variables indicate the control point locations in the original and uppercase variables indicate the control points in the new image. After substituting and solving the above equations, we can get the $\boldsymbol{H}$ matrix. But if we are using inverse warping algorithm, we will need the $\boldsymbol{H}^{-1}$ matrix. Therefore, the $\boldsymbol{H}^{-1}$ matrix for *left.raw* and *middle.raw* image is:

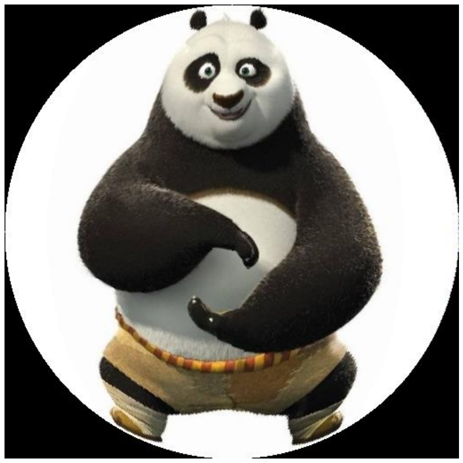
$$H^{-1} = \begin{bmatrix} 5.93574612e-01 & -1.18161093e-01 & 1.13810691e+02 \\ -1.46431303e-02 & 4.39553548e-01 & -1.73736135e+02 \\ -2.65245643e-05 & -4.22235087e-04 & 1 \end{bmatrix}$$

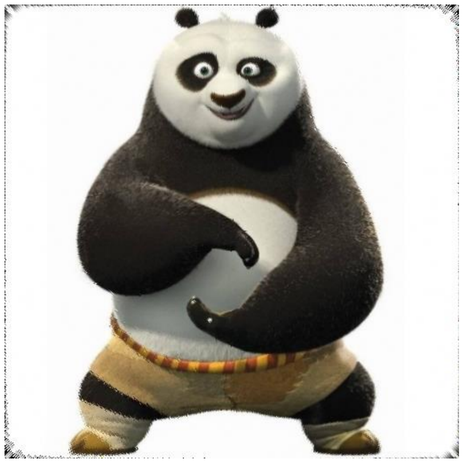
Similarly, the $\boldsymbol{H}^{-1}$ matrix for *right.raw* and *middle.raw* image is:

$$H^{-1} = \begin{bmatrix} 2.72080308e+01 & 7.84501789e+00 & -9.03502312e+03 \\ 2.47651339e-02 & 3.05513203e+01 & -3.08702441e+04 \\ -1.01013995e-03 & 2.32888902e-02 & 1 \end{bmatrix}$$

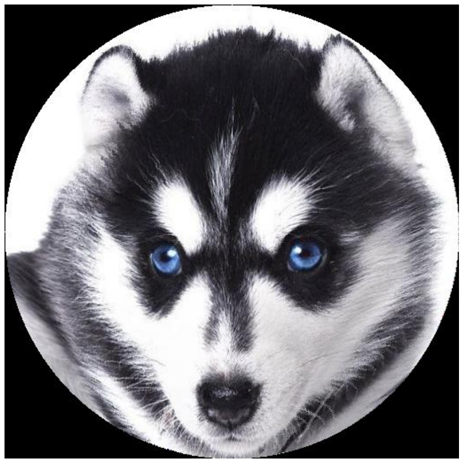
1.3 Experimental Results:

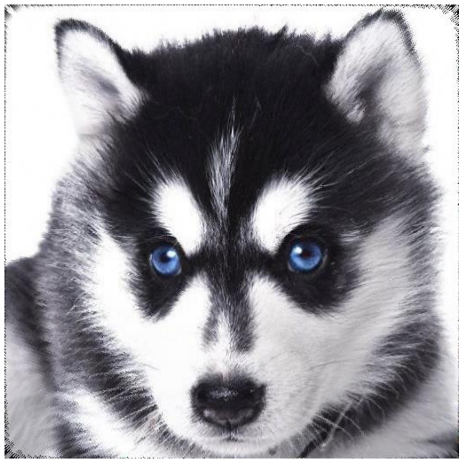
(a) Original *panda.raw* image

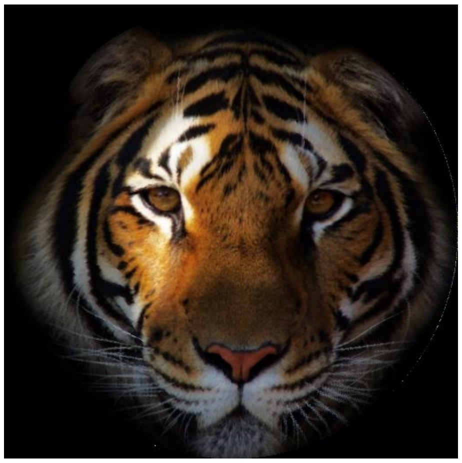
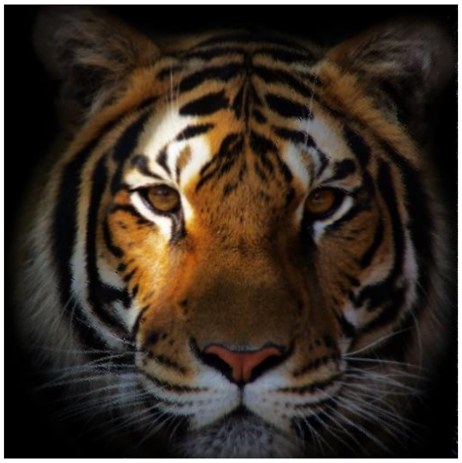

(b) Circularly warped image


(c) Reverse warped image

(d) Original *panda.raw* image


(e) Circularly warped image

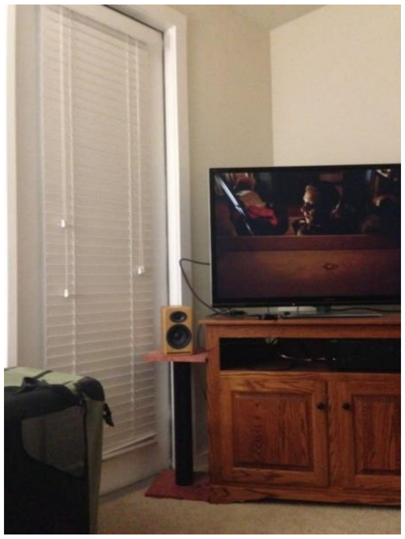
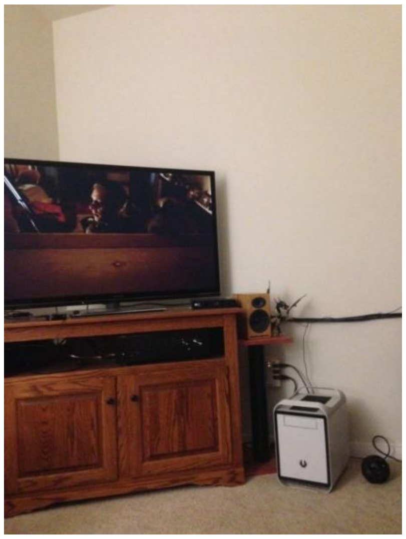
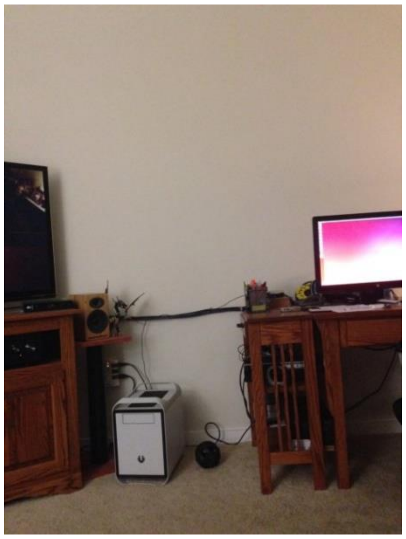

(f) Reverse warped image

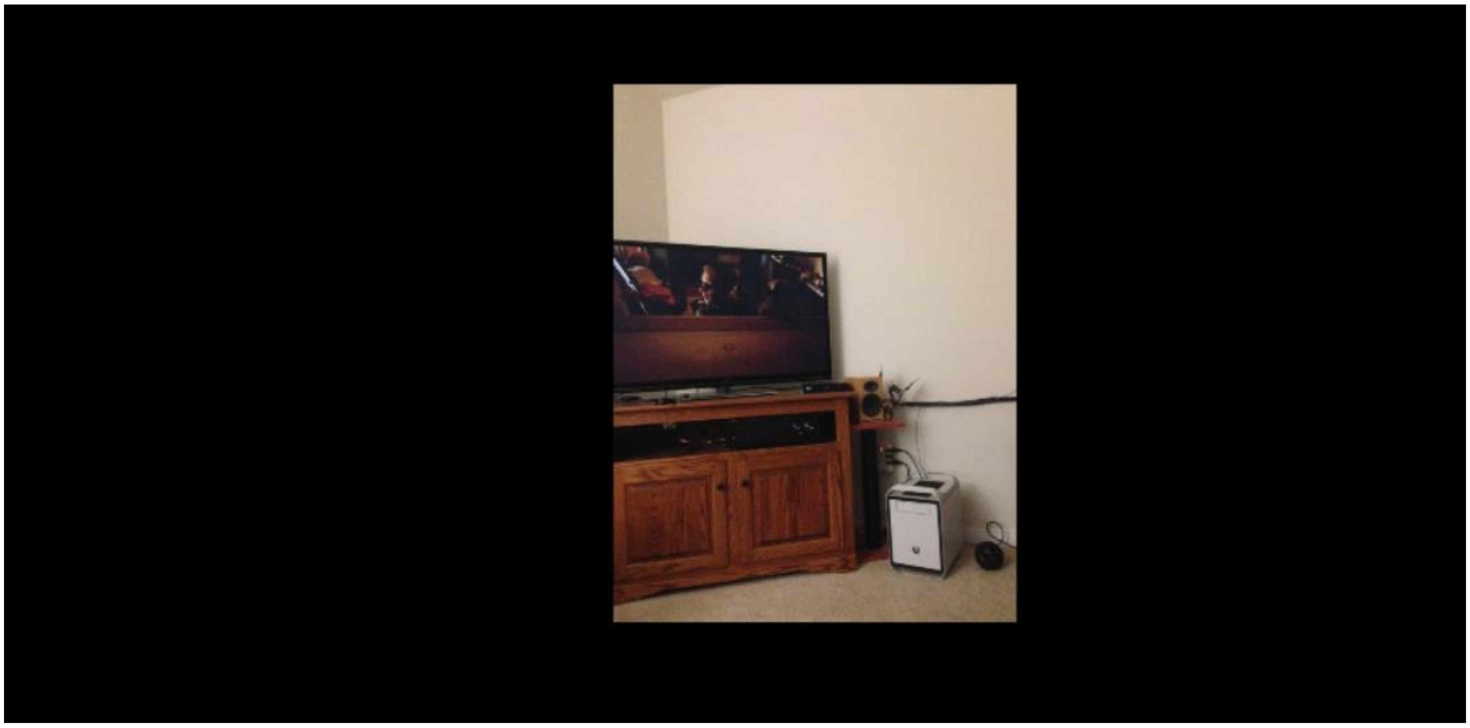
(g) Original *tiger.raw* image (h) Circularly warped image (i) Reverse warped image

Figure 2. Output for the *Circular warping*

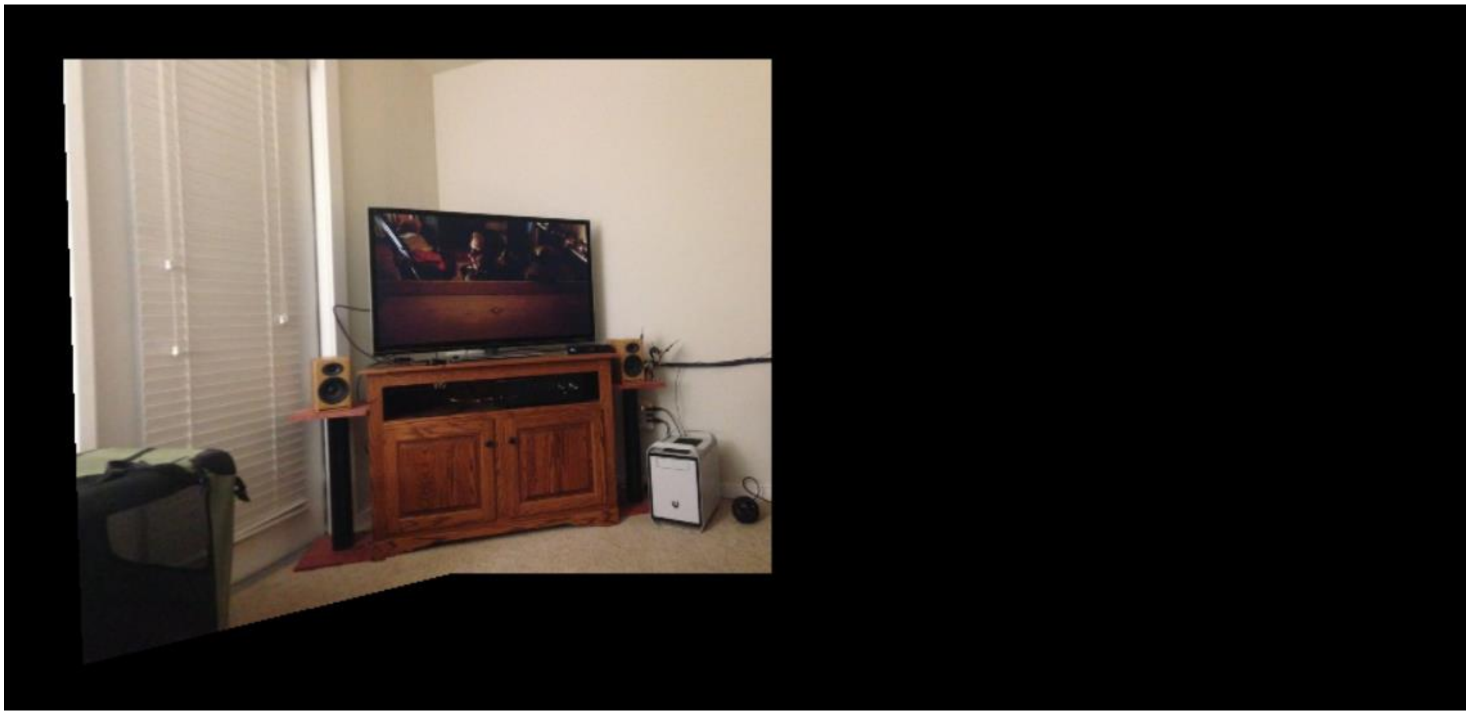
1.3.2 Homographic Transformation and Image Stitching:

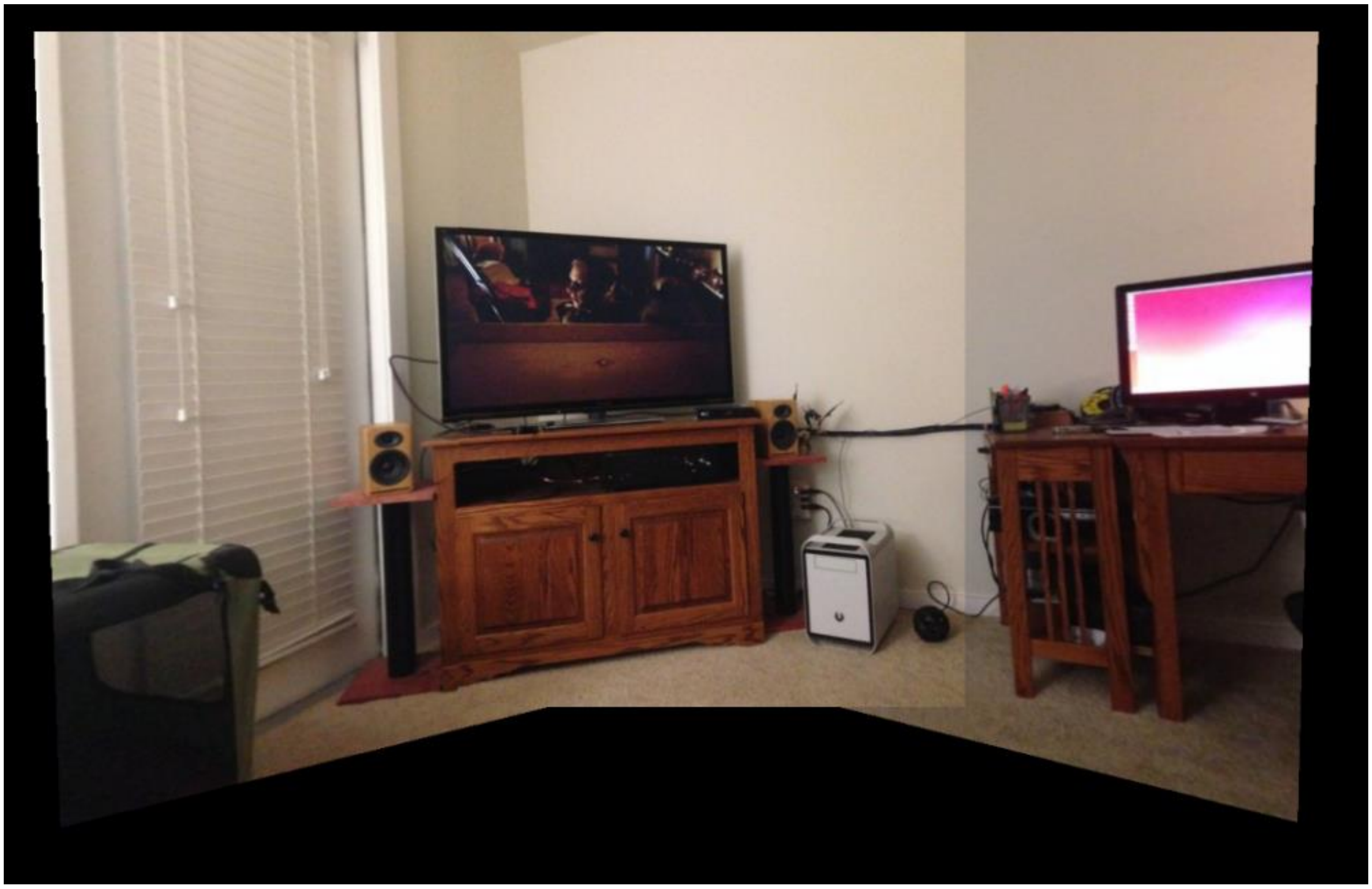
(a) Original *left.raw* image (b) Original *middle.raw* image (c) Original *right.raw* image



(d) *Middle.raw* image copied on a blank black image



(e) *Left.raw* projected on *Middle.raw* image and morphed together



(f) Final stitched image

Figure 3. Output for the ***Homographic Transformation and Image Stitching***

1.3 Discussion:

- **Geometrical Image Modification:** It can be observed the image does not perfectly warp to a circle. A white boundary can be observed in image (h). This is because the square shaped pixels cannot be used to create a smooth boundary. Furthermore, warping a circular image back to square also produces black coloured patterns at the corners and edges of the square image. The primary reason for this is, when converting a circle to a square, we are increasing the are of the image and because of this reason, many pixel locations in the square image don't have a mapping to a pixel in the circular image.

  There are several other approaches for warping a square image to circular image. One such approach would be to use circular coordinates. Every pixel in the square can be uniquely defined by the distance from the centre of the image and the angle at which the pixel is located. Keeping the angles same, and setting the distance of each pixel to a constant radius, square image can be warped to a circular image.

- **Homographic Transformation and Image Stitching:** In this problem I have only used just four control points to compute the homography matrix. I have chosen only those pixels as the control points which I could visually distinguish on both the source and target images. But in practice, more than four control points are used. This improves the homographic transformation and renders it more accurate. Furthermore, control points are never chosen on a visual basis. In practice, SIFT and SURF algorithms are used to find the right amount of and the right types of control points in an image. Scale Invariant Feature Transform (SIFT) is used to find key-points in an image which best describe the image and are robust to geometrical distortion and invariant to translation, rotation, scaling and even partial illumination. Thus, even if the target image is a scaled or rotated version of the source image, the SIFT algorithm will correctly find the best image descriptors.

2. Problem 2

2.1 Abstract and Motivation:

Halftoning is a process that uses dots of varying sizes and/or varying spacing to simulate continuous colour palette.

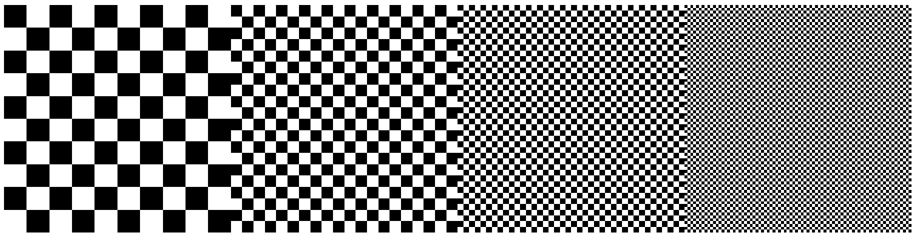


Figure 4. Example of a Halftone image[1]

Halftoning is widely used in the printing industry. This process helps to reproduce varying tones of colours with significantly fewer inks. The basic concept of halftoning can be explained using the above figure. The entire image is made up of just black and white square patches. But as the size of square patches decreases, one can observe a shade of grey colour in the image (which is, in fact, not even present in the image). This can be attributed to the fact that it becomes hard for the human eye to distinguish between two neighbouring square patches. Hence, the human eye blends fine details and records an overall intensity. Thus, halftoning exploits the fact the human eye cannot distinguish fine details to create an optical illusion of producing wide range of intensity levels using only a few intensities.

The concept of halftoning can also be use in an colour image. Dots of just a few colours placed together can fool the human eye in believing that a larger colour gamut exists. Halftoning can be achieved by a combination of *amplitude modulation* and *spatial modulation*. *Amplitude modulation* refers to the size of the dots. It can be observed in figure that dots with large size do not produce a very good halftone image. On the other hand, dots of small size produce a finely detailed image. *Spatial modulation* refers to spatial arrangement of the dots in the image. If the dots are placed far apart, the halftone image won't be a good reproduction of the original image. On the other hand, tightly packed dots will reproduce a very good image.

Dithering is a more general term for halftoning that refers to introducing noise and perturbations in the image to randomize the quantization error. There are two methods to generate a dithered image and they are:

1. Ordered Dithering
2. Error Diffusion

These dithering techniques will be discussed in the next section

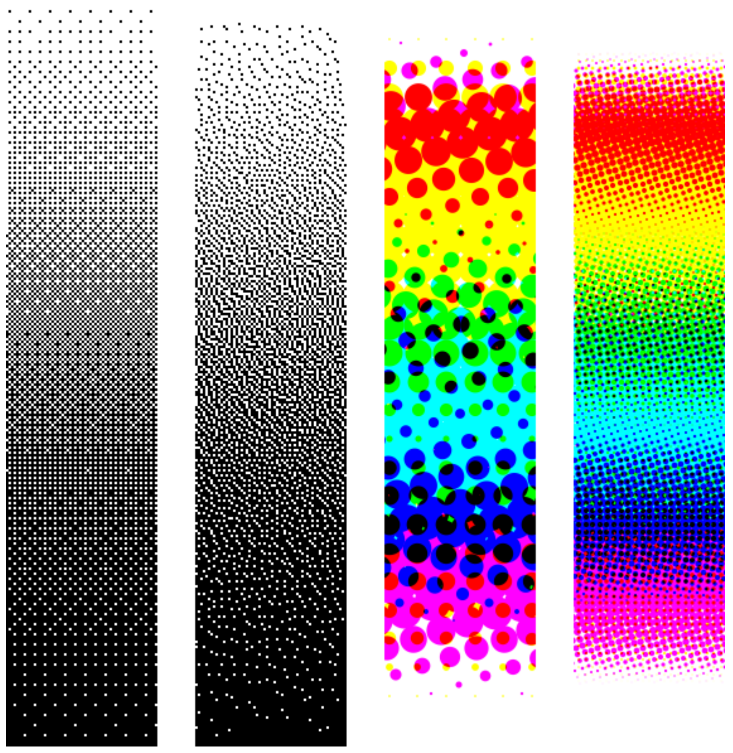


Figure 5. Left: two examples of monochrome dithering, firstly pattern dithering and secondly diffusion dithering. Right: two examples of colour halftoning by amplitude modulation, first with a very large dot size, the second with a smaller dot size that is more convincing

2.2 Approach and Procedure:

There are 3 parts in this question. This first part deals with create a halftone image using dithering, the second part deals with creating a halftone using error diffusion, and the last part concerns with creating halftone colour images.

2.2.1 Dithering:

This section is again divided in three subparts:

(i) Fixed Thresholding: The input image is quantized using a fixed threshold value T. Thus, any pixel having intensity level greater than 127 is quantized to 255 and if the intensity level is less than 127, the pixel is quantized to intensity 0. This can be mathematically represented as:

$$\boldsymbol{G(i,j) = \begin{cases} 0 & if\ 0 \leq F(i,j) < T \\ 255 & if\ T\ \leq F(i,j) < 256 \end{cases}}$$

Where, $G(i,j)$ is the output image and $F(i,j)$ is the input image. I have set the threshold intensity level as 127

(ii) Random Thresholding: Instead of quantizing the pixel intensities using a fixed threshold intensity level T , each pixel is quantized using a different threshold intensity, which is obtained using the $rand(i,j)$ function. This can be mathematically represented as:

$$\boldsymbol{G(i,j)} = \begin{cases} \mathbf{255} & \boldsymbol{if\ rand(i,j) \leq F(i,j)} \\ \mathbf{0} & \boldsymbol{if\ rand(i,j) > F(i,j)} \end{cases}$$

(iii) Ordered Dithering: In ordered dithering, Bayer matrix is used to create a halftone image. The Bayer matrix can be thought of as a matrix which defines an order in which the pixels are to be "turned on". For a given pixel in the image, Bayer matrix decides the threshold intensity for the pixel and for its nearest neighbours too. The general formula for the Bayer matrix is given by:

$$\boldsymbol{I_{2n}} = \begin{bmatrix} \boldsymbol{4 * I_n + 1} & \boldsymbol{4 * I_n + 2} \\ \boldsymbol{4 * I_n + 3} & \boldsymbol{4 * I_n} \end{bmatrix}$$

Thus, the 2x2, 4x4 and 8x8 Bayer matrix are:

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

$$\boldsymbol{I_4(i,j)} = \begin{bmatrix} \mathbf{5} & \mathbf{9} & \mathbf{6} & \mathbf{10} \\ \mathbf{13} & \mathbf{1} & \mathbf{14} & \mathbf{2} \\ \mathbf{7} & \mathbf{11} & \mathbf{4} & \mathbf{8} \\ \mathbf{15} & \mathbf{3} & \mathbf{12} & \mathbf{0} \end{bmatrix}$$

$$\boldsymbol{I_8(i,j)} = \begin{bmatrix} \mathbf{21} & \mathbf{37} & \mathbf{25} & \mathbf{41} & \mathbf{22} & \mathbf{38} & \mathbf{26} & \mathbf{42} \\ \mathbf{53} & \mathbf{5} & \mathbf{57} & \mathbf{9} & \mathbf{54} & \mathbf{6} & \mathbf{58} & \mathbf{10} \\ \mathbf{29} & \mathbf{45} & \mathbf{17} & \mathbf{33} & \mathbf{30} & \mathbf{46} & \mathbf{18} & \mathbf{34} \\ \mathbf{61} & \mathbf{13} & \mathbf{49} & \mathbf{1} & \mathbf{62} & \mathbf{14} & \mathbf{50} & \mathbf{2} \\ \mathbf{23} & \mathbf{39} & \mathbf{27} & \mathbf{43} & \mathbf{20} & \mathbf{36} & \mathbf{24} & \mathbf{40} \\ \mathbf{55} & \mathbf{7} & \mathbf{59} & \mathbf{11} & \mathbf{52} & \mathbf{4} & \mathbf{56} & \mathbf{8} \\ \mathbf{31} & \mathbf{47} & \mathbf{19} & \mathbf{35} & \mathbf{28} & \mathbf{44} & \mathbf{16} & \mathbf{32} \\ \mathbf{63} & \mathbf{15} & \mathbf{51} & \mathbf{3} & \mathbf{60} & \mathbf{12} & \mathbf{48} & \mathbf{0} \end{bmatrix}$$

These Bayer matrices can be transformed into threshold matrix T for an input grey-level image with normalized pixel values (i.e. with its dynamic range between 0 and 1) by the following formula:

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

Where, $\boldsymbol{N^2}$ the total number of pixels in the image. The above matrix is periodically repeated across the full image, and this is done by

$$\boldsymbol{G(i,j)} = \begin{cases} \mathbf{1} & \boldsymbol{F(i,j) > T(mod\ i, mod\ j)} \\ \mathbf{0} & \boldsymbol{otherwise} \end{cases}$$

Where, $G(i,j)$ is the normalized output binary image.

In the last part, we are expected to convert a 256-level grey image to an image having just four intensities 0, 85, 170, 255. This can be implemented using one of the above Bayer matrix and changing the threshold condition to:

$$\boldsymbol{G(i,j)} = \begin{cases} \mathbf{255} & \boldsymbol{f(i,j) > T} \\ \mathbf{170} & \boldsymbol{\frac{2T}{3} < f(i,j) \leq T} \\ \mathbf{85} & \boldsymbol{\frac{T}{3} < f(i,j) \leq \frac{2T}{3}} \\ \mathbf{0} & \boldsymbol{f(i,j) \leq \frac{T}{3}} \end{cases}$$

2.2.2 Error Diffusion:

In this technique, each pixel is quantized using a pre-determined threshold and the quantization error is propagated to the pixels in the immediate next row and next column.

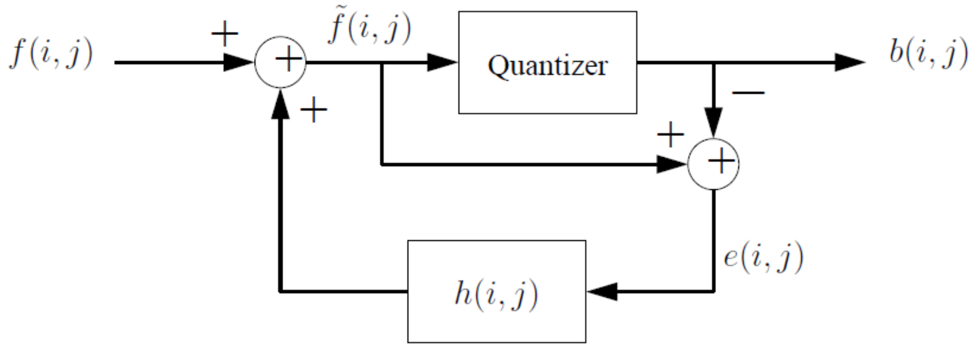


Figure 6. Error Diffusion Technique

The algorithm for error diffusion can be stated as:

1. $b(i,j) = \begin{cases} 255 & \hat{f}(i,j) > T \\ 0 & otherwise \end{cases}$

2. $e(i,j) = \hat{f}(i,j) - b(i,j)$

3. $\hat{f}(i,j) = f(i,j) + \sum_{k,l \in S} h(k,l)e(i-k, j-l)$

Where, T = 127 (typically) and *h(k, l)* are typically chosen as positive and sum to 1

The various error diffusion matrices are

1. **Floyd – Steinberg:**

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

2. **Jarvis, Judice and Ninke (JJN):**

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

3. **Stucki:**

$$\frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

2.2.3 Colour Halftoning with Error Diffusion:

- **Separable Error Diffusion:** In this method, each of the channels of the input colour image is subjected to one of the error diffusion techniques to quantize each of the channel separately. The error diffusion is carried out in the same way as explained in the preceding section. Then we will have an image with one of the following eight colours: White (255, 255, 255), Red (255, 0, 0), Blue (0, 255, 0), Green (0, 0, 255), Cyan (0, 255, 255), Magenta (255, 0, 255), Yellow (255, 255, 0) Black (0, 0, 0).

- **MBVQ – based Error Diffusion:** The *Minimal Brightness Variation Quadruples (MBVQ)* method considers the fact that all the colours are not equally bright It states that the use of just the normal error diffusion techniques won't reproduce a very a very good halftoned colour image. The algorithm for the MBVQ-based error diffusion is:
  1. Initially, for each of the pixel in an image one of the six tetrahedron is assigned to the RGB triplet of the pixel. The six tetrahedrons are:

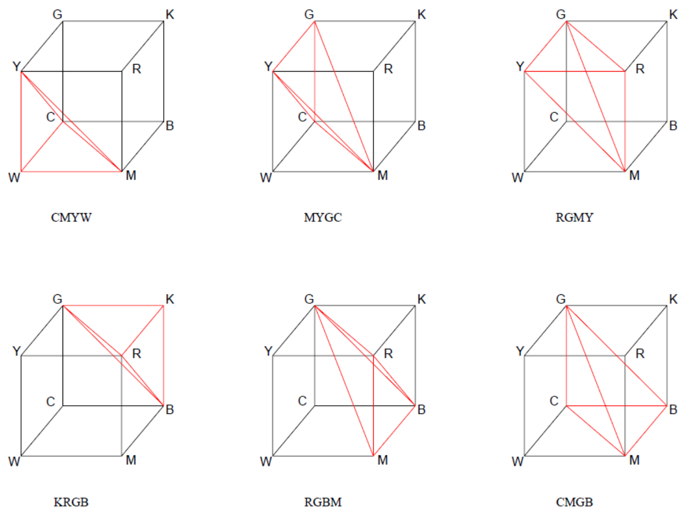


Figure 7. The partition of the RGB de tourbe to six tetrahedral volumes, each of which the convex hull of the MBVQ used to render colours in it

The RGB triplet can be assigned to one of the tetrahedrons using the following pseudocode:

```
if((R1 + G1) > 255) {
          if((G1 + B1) >255) {
               if((R1 + G1 + B1) > 510)
                    CMYW(R2, G2, B2);
               else
                    MYGC(R2, G2, B2);
          }
          else
                    RGMY(R2, G2, B2);
     }
     else {
          if((G1 + B1) <= 255){
               if((R1 + G1 + B1) <= 255)
                    KRGB(R2, G2, B2);
               else
                    RGBM(R2, G2, B2);
          }
          else
                    CMGB(R2, G2, B2);
     }
```

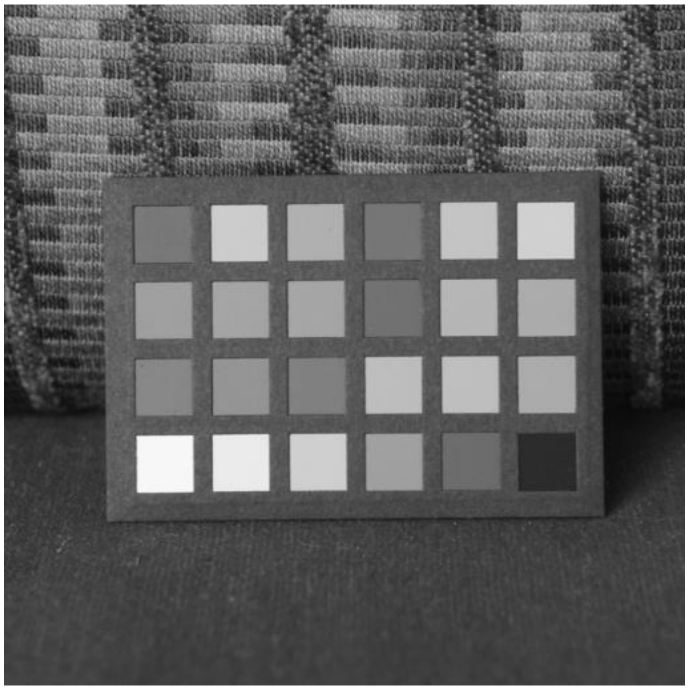
2. Once a RGB triplet is assigned is to one of the tetrahedrons, the vertex of the tetrahedrons closest to the RGB triplet is found using Euclidean distance.

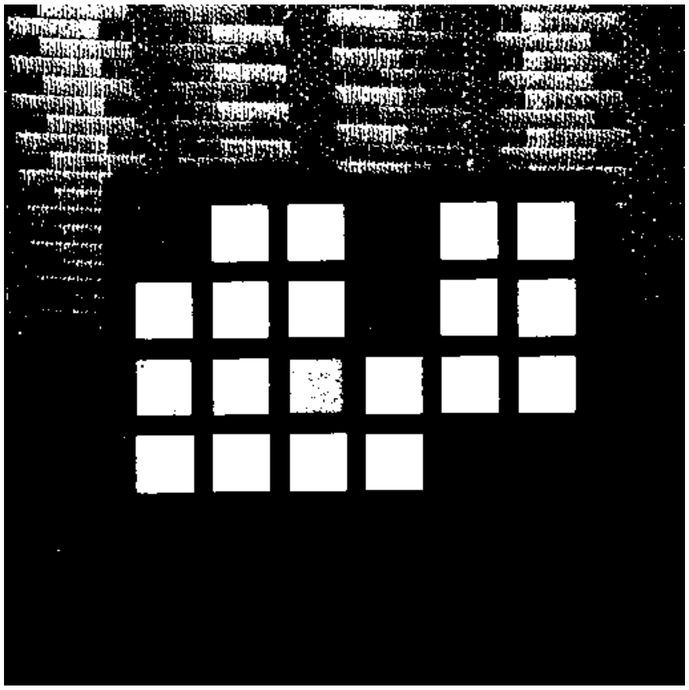
   The algorithm can be summarized in the following steps:
   For each pixel (i, j) in the image:
   i) Determine $MBVQ(RGB(i,j))$
   ii) Find the vertex $v \in MBVQ$ which is closest to $RGB(i,j) + e(i,j)$
   iii) Compute the quantization error $RGB(i,j) + e(i,j) - v$
   iv) Distribute the quantization error to the "future" pixel

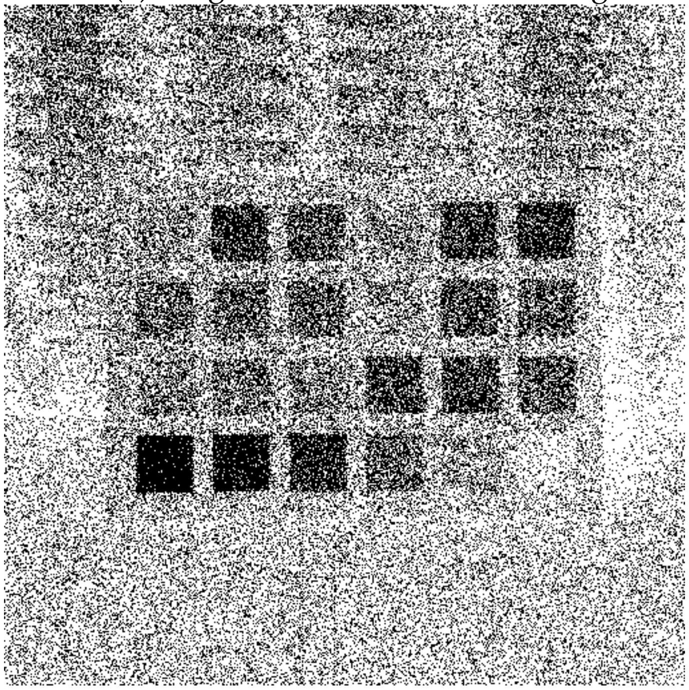
Here, $RGB(i,j)$ denotes the RGB value at the pixel (i,j) and $e(i,j)$ denotes the error accumulated by pixel at the (i,j) location.

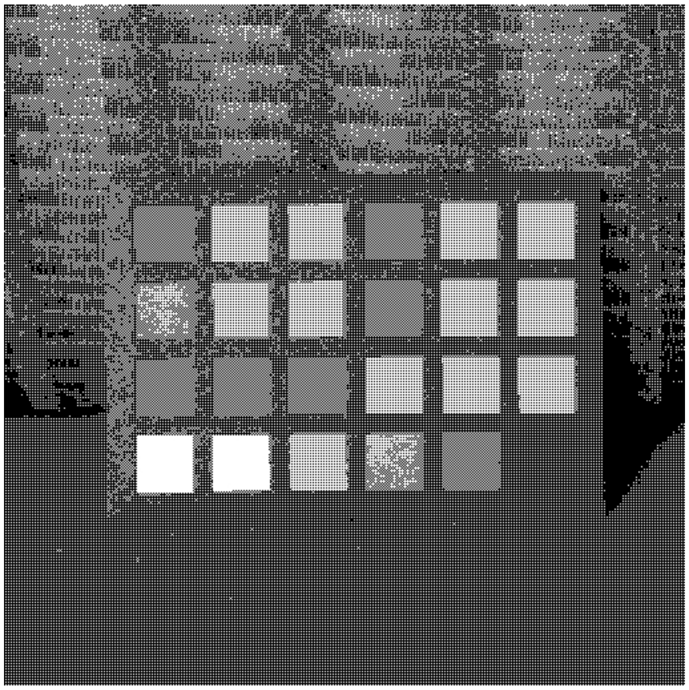
2.3 Experimental Results:

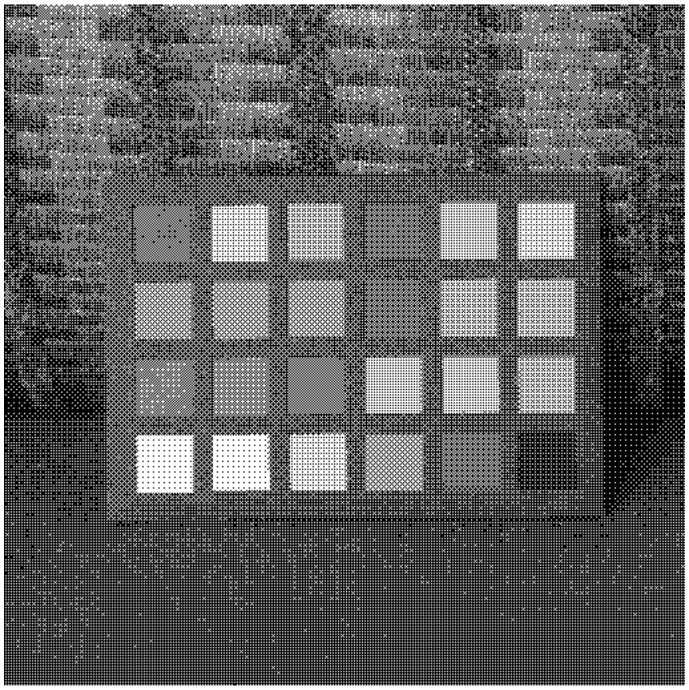
2.3.1 Dithering:


(a) Original colorchecker.raw image

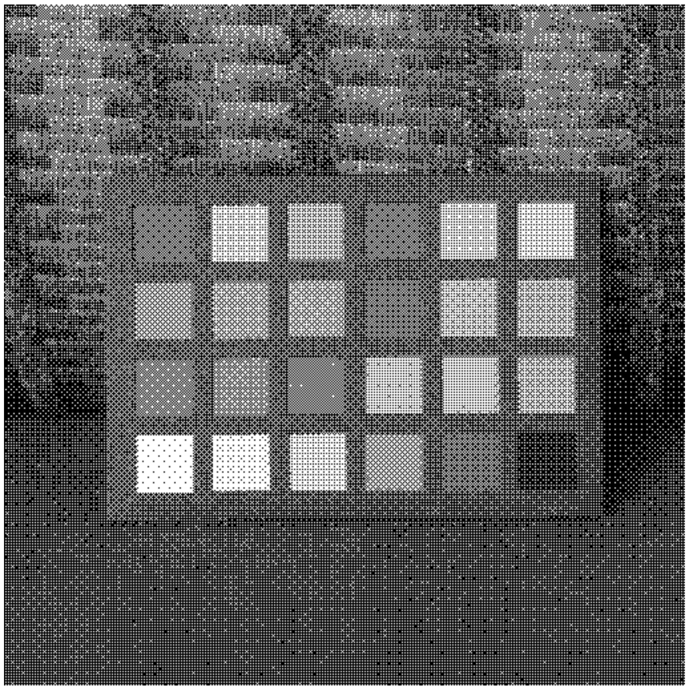

(b) Output image using *Fixed Thresholding*


(c) Output image using *Random Thresholding*


(d) Output image using Bayer dither matrix of 2x2 size



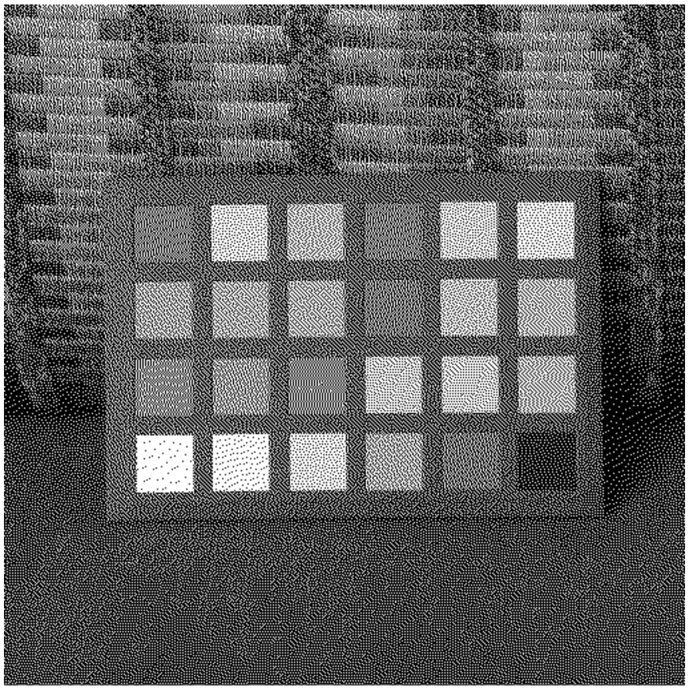
(e) Output image using Bayer dither matrix of 4x4 size



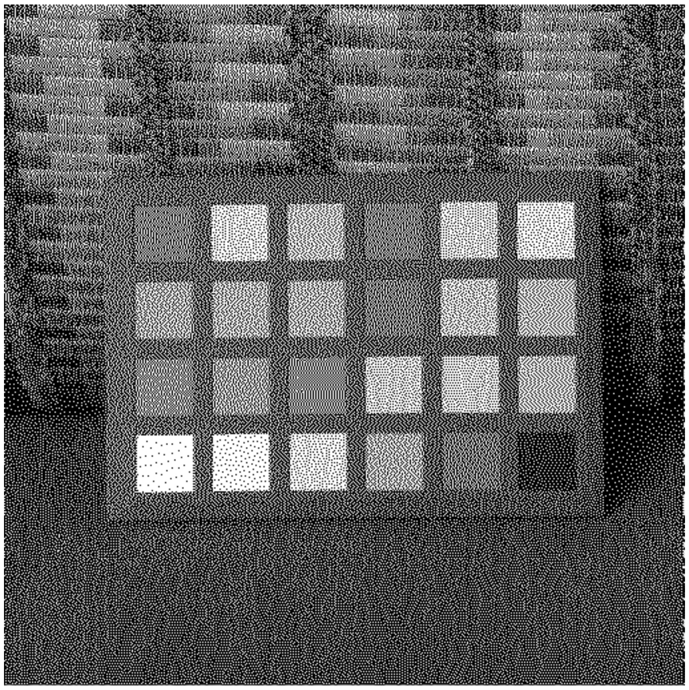
(f) Output image using Bayer dither matrix of 8x8 size

Figure 8. Output images for Part (A) of Problem 2

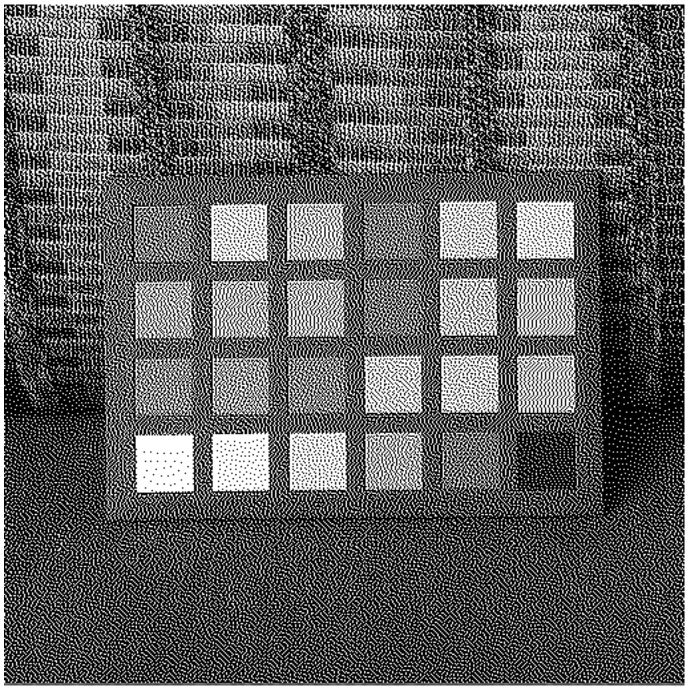
2.3.2 Error Diffusion:



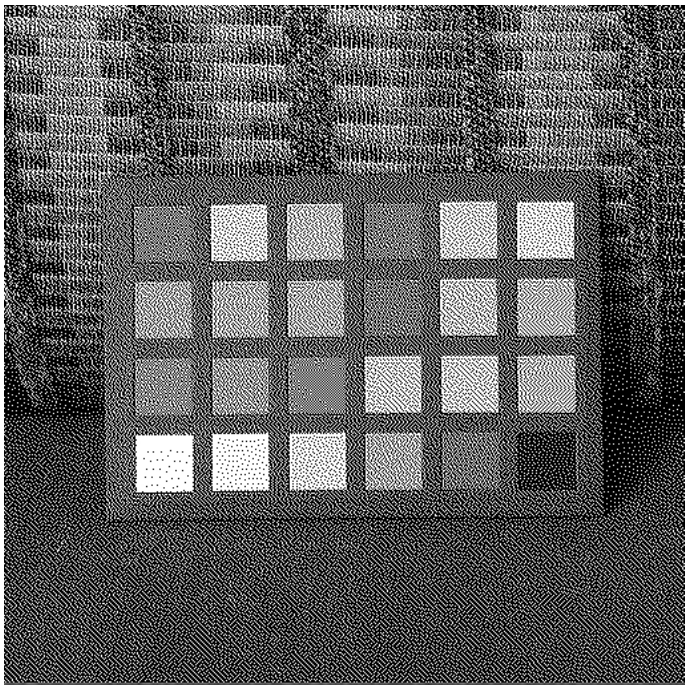
(a) Floyd – Steinberg error diffusion using normal scanning



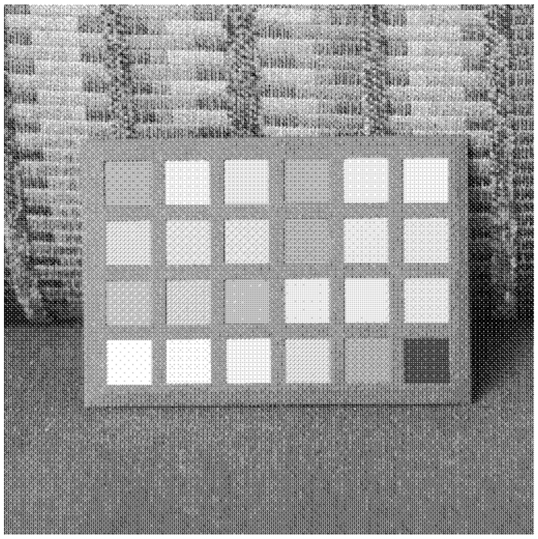
(b) Floyd – Steinberg error diffusion using serpentine scanning



(c) Jarvis, Judice and Ninke error diffusion technique



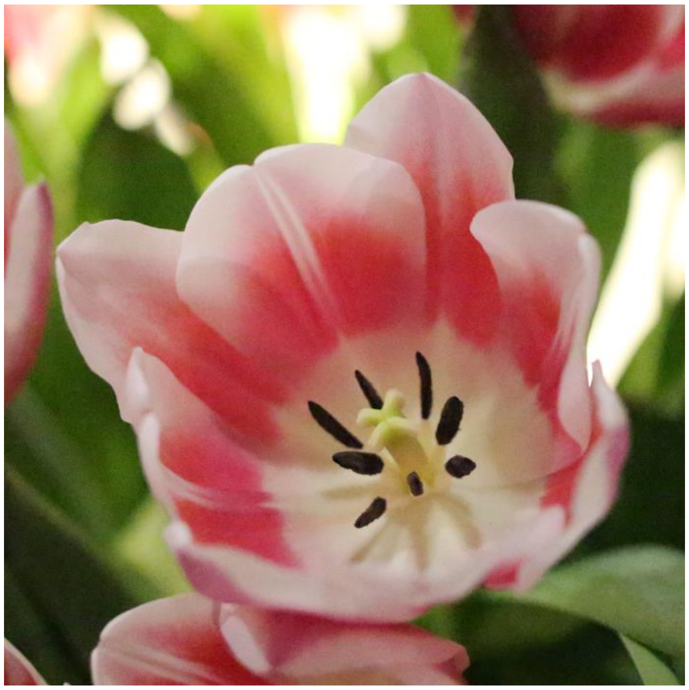
(d) Stucki error diffusion technique



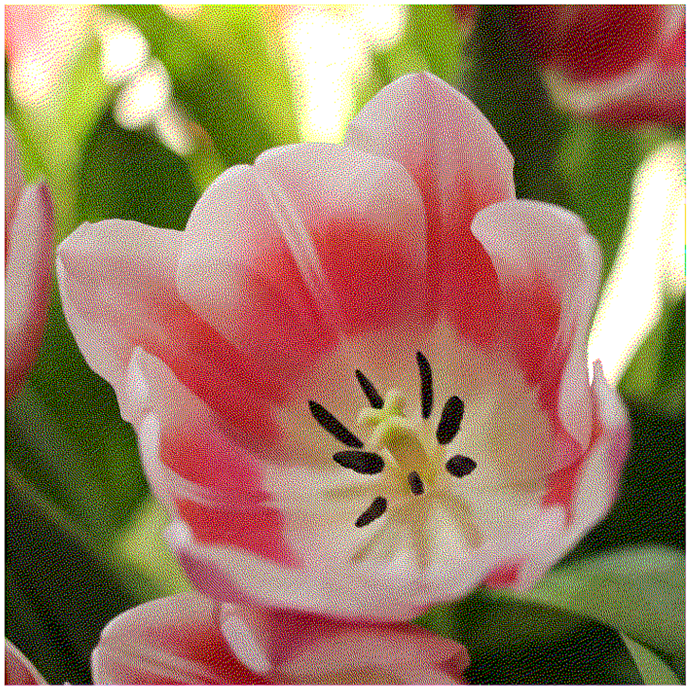
(e) Image dithered using four intensity levels

Figure 9. Output images using different error diffusion techniques

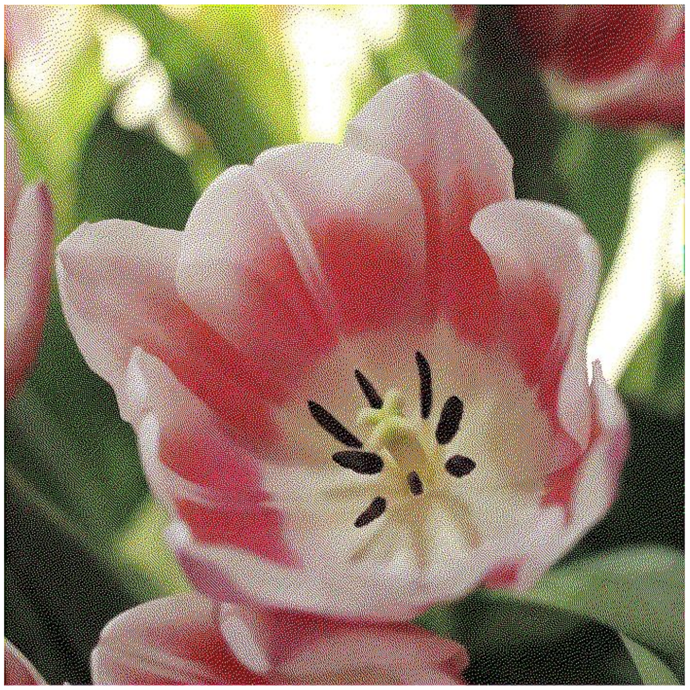
2.3.3 Colour Halftoning with Error Diffusion:



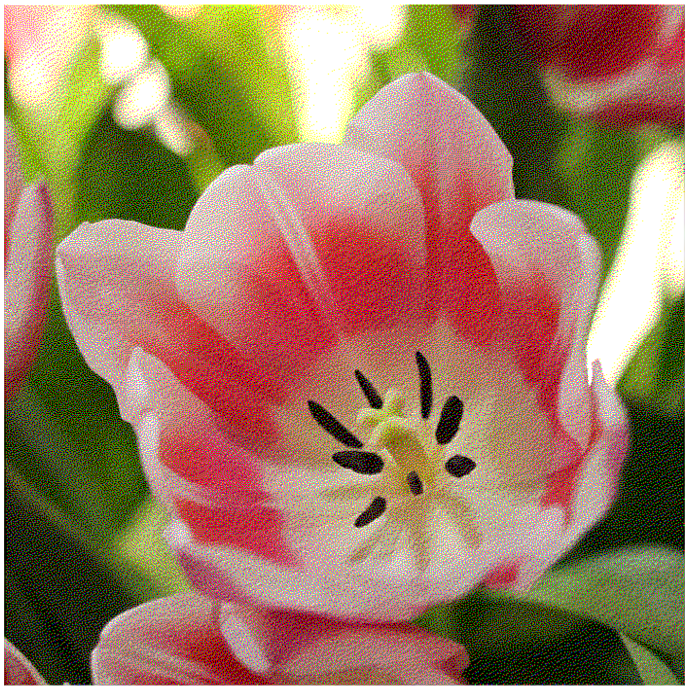
(a) Original Flower.raw image



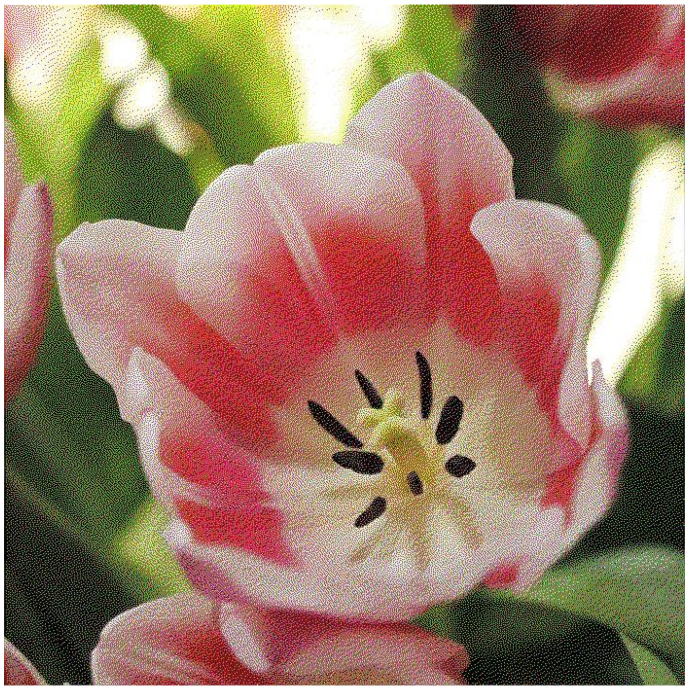
(b) Output image using *Separable Error Diffusion* (image displayed in *.png* lossless format)



(c) Output image using *Separable Error Diffusion* (image displayed in *.jpg* lossy format)



(d) Output image using *MBVQ-based Error Diffusion* (image displayed in *.png* lossless format)

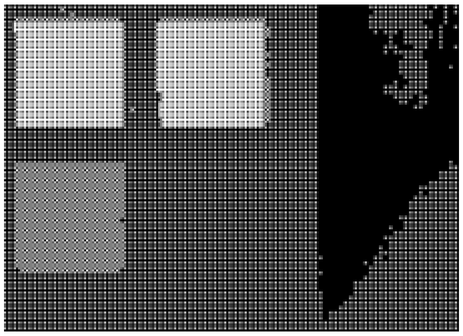


(e) Output image using *MBVQ-based Error Diffusion* (image displayed in *.jpg* lossy format)

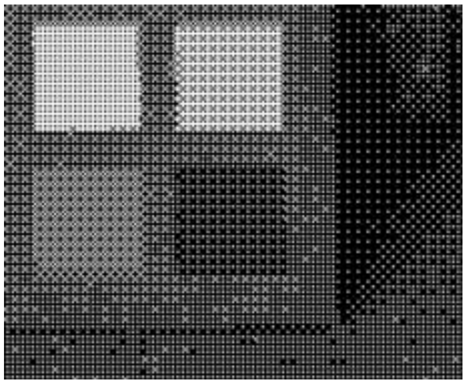
Figure 10. Colour Halftone images for part (c) of Problem 2

2.4 Discussion:

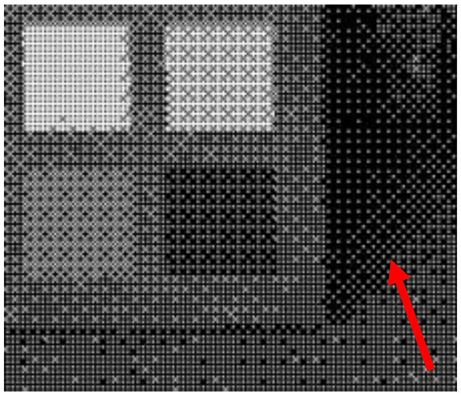
Ordered Dithering: We can observe that the *fixed thresholding* and *random thresholding* techniques gave rather poor results. Several squares were missing in the output images for both the techniques. Ordered dithering gave rather good results, since all the squares were present and discernible in the output images for each of the different Bayer matrix. Consider the bottom right part in the images. These are shown in the figure below for each of the different Bayer matrix



(a) Halftone image using 2x2 dither image



(b) Halftone image using 4x4 dither image



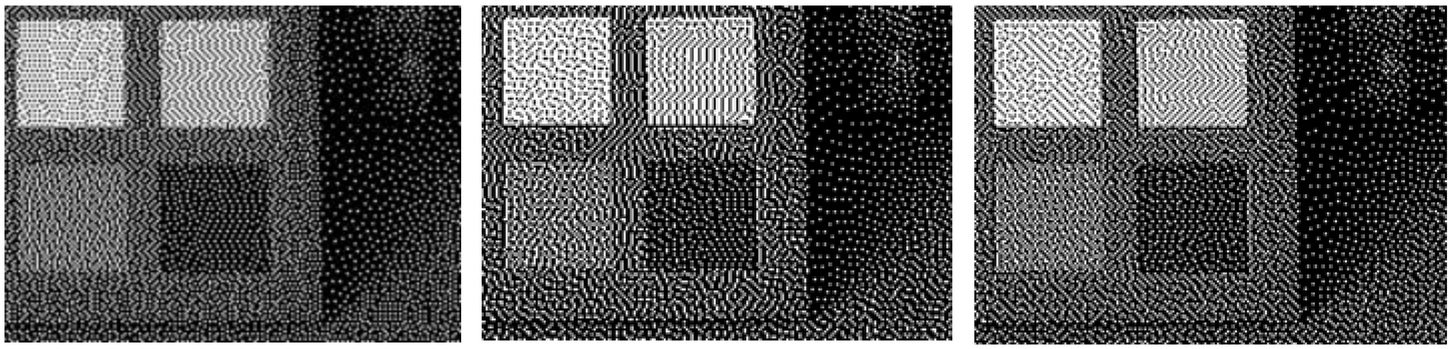
(c) Halftone image using 8x8 dither image

Figure 11. Difference in the output image for Bayer matrices different sizes

It can be observed that 2x2 Bayer matrix results in loss of information since it fails to reproduce the bottom corner square. Both the 4x4 and 8x8 Bayer matrix have reproduced that same square. Now consider the shadow behind the checkerboard (pointed by the red arrow). One can argue that the grey intensity levels of the shadow have been better represented by the 8x8 Bayer matrix than the 4x4 matrix.

One of the disadvantages of halftoned images obtained using ordered dithering is that one can observe repetitive patterns in the image. If the images are viewed from a relatively large distance, these patterns are not discernible. But when viewed up close, one can notice these patterns. And these patterns can be seen more prominently on a computer screen. A computer screen with a relatively high resolution can show these patterns in the images very clearly. But if the same image is zoomed out and viewed or if the image is saved in a lossy image format such as *.JPEG,* then the patterns are not quite discernible.

Error diffusion: Again, consider the bottom right part in the images. These are shown in the figure below for each of the different error diffusion techniques.



(a) Halftone image using Floyd – Steinberg Algorithm

(b) Halftone image using Jarvis, Judice and Ninke Algorithm

(c) Halftone image using Stucki Algorithm

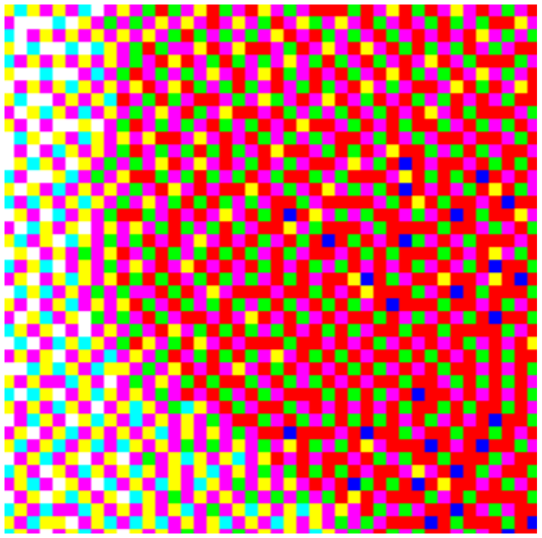
Figure 12. Difference in the output image for different error diffusion techniques

All the error diffusion techniques have been able to reproduce the corner square. And even the shadow behind the checkerboard is well reproduced by all the three techniques. It's quite difficult to choose any one technique as the best technique among the three techniques.

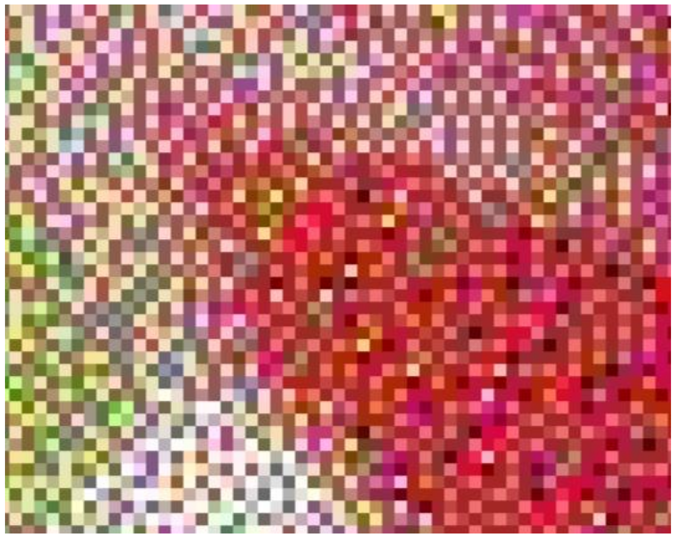
Repetitive patterns can also be observed in halftone images obtained using *error diffusion* techniques. But the patterns in these images are supposedly less frequent and less distracting. And once, if the image is zoomed out and viewed or if the image is saved in a lossy image format such as *.JPEG,* the patterns almost vanish.

Colour Halftoning: If the output images from both the techniques (*Separable error diffusion* and *MBVQ-based error diffusion)* are zoomed out and observed, the output image using the *MBVQ-based error diffusion* seems brighter that the other output image. It can be attributed the fact the *MBVQ* works on the principle of quantizing the pixels to intensity levels which better approximate the brightness of the original pixel intensity level.

The problem with saving halftone images in a lossy image format concerns with the fact that the lossy image format again introduces almost the whole colour gamut which we had approximated using just a few colours, bringing us back to square one! This can be better understood by observing the two images below



(a) Zoomed in part of the halftoned image saved in *.png format* (lossless)



(b) Zoomed in part of the halftoned image saved in *.jpg format* (lossy)

Figure 13. Difference between halftoned image saved in a lossless format and a lossy format

We can observe that in the right image, all the original colours we re-introduced, rendering the whole objective of converting an image to halftone useless.

I would like to propose the following matrix for error diffusion:

$$\frac{1}{32}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 3 \\ 2 & 4 & 5 & 4 & 2 \\ 1 & 1 & 3 & 1 & 1 \end{bmatrix}$$

3. Problem 3

3.1 Abstract and Motivation:

Morphology is a branch of science that refers to the structure of animals and plants. In the context of image processing, *mathematical morphology* can be described as a tool for extracting image components that describe a shape, such as boundaries, skeletons and convex hull. Morphological operations rely only on the relative ordering of pixels values and their spatial distribution. Hence, they are well suited for binary image. Some of the basic morphological operations are dilation, erosion, opening, closing etc. Binary image morphology is dependent on the connectivity of the neighbourhood of a given pixel. Consider a 3x3 neighbourhood for a pixel X in an image:

$$\begin{matrix} X_3 & X_2 & X_1 \\ X_4 & X & X_0 \\ X_5 & X_6 & X_7 \end{matrix}$$

The pixel X is said to be *four -connected* to a neighbour if X is a logical 1 and if anyone among $X_0,\ X_2,\ X_4,\ X_6$ is logical 1. Similarly, pixel X is said to be *eight-connected* to a neighbour if X is logical 1 and nay of its neighbours is also logical 1. Furthermore, morphological operations such as shrinking, thinning and skeletonizing can be implemented using *hit-or-miss transformation.* The procedure to implement these morphological techniques will be discussed in the next section.

Shrinking erases black pixels in an image such that an object without holes is eroded to a single pixel near or away from its centre of mass. Objects with holes are eroded to to a connected ring which is mid-way between hole and the nearest boundary.

Thinning erases black pixels such that an object without holes is eroded to a thin connected line which is equidistant from its nearest outer boundaries and an object with hole is eroded to a thin ring situated midway between each hole and the outer boundary

Skeletonizing reduces the image to a thin stick figure (skeleton) which best approximates the overall shape of the original image. The skeleton is equidistant from the edges of the object and has nodes in the corner.

3.2 Approach and Procedure:

There are four parts in this section. But since the approach for the first three parts is almost the same, they will be grouped together. The first part explains the implementation of *shrinking, thinning, skeletonizing* and the second part discusses an approach to implement the *counting game.*

3.2.1 Shrinking, Thinning and Skeletonizing:

Shrinking, thinning and skeletonizing are types of conditional erosion techniques in which the erosion is carried out in a controlled manner to prevent erasing the image completely and to maintain the connectivity of pixels in the image. It is not prudent to perform these operations using a single 3x3 filter for *hit-or-miss* because this could result in total erasure of the image and loss of information. Furthermore, it is difficult to implement a 5x5 filter because it would require the examination of almost 2^{25} patterns.

Hence such a controlled erosion is carried out using a two-stage *hit-or-miss transformation,* where two 3x3 filters are used in each stage.

- **Stage 1:**

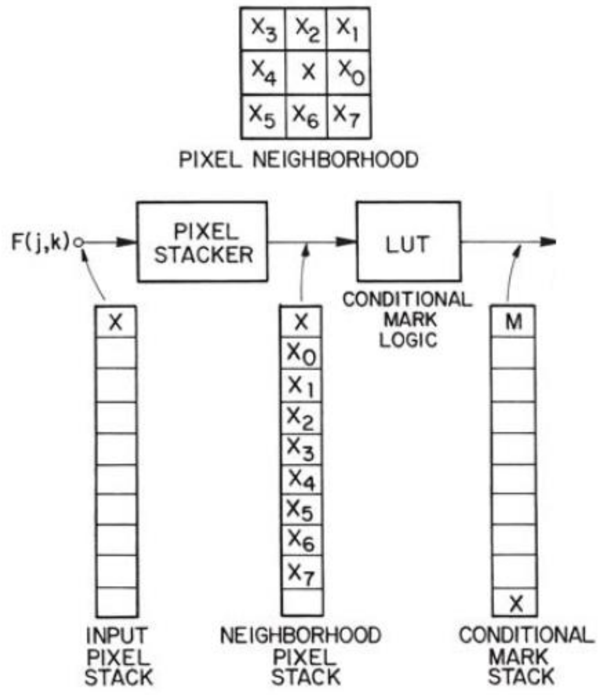


Figure 14. Stage 1 for *shrinking, thinning, skeletonizing*

1. Since morphological image operations are applied on binary images, the input image is initially quantized to produce a binary image $\boldsymbol{F(i,j)}$**.** If the object under consideration (foreground) is represented by intensity level 0 and the background is represented by intensity level 1, the

image intensities must be reversed. For our algorithm, the foreground i.e. the object under consideration must always be represented by intensity level 1.

2. Now, for each pixel of the image $\boldsymbol{F(i,j)}$, if the intensity of the pixel is 1, then the 3x3 neighbourhood of the image is noted and converted into a pattern as shown in the figure above. Consider the following 3x3 image grid.

$$\begin{matrix} X_3 & X_2 & X_1 \\ X_4 & X & X_0 \\ X_5 & X_6 & X_7 \end{matrix}$$

The neighbourhood pattern for the pixel X is given by $X_0\, X_1\, X_2\, X_3 X_4\, X_5 X_6\, X_7$

3. Now, this pattern matched with all the patterns in the *conditional patterns lookup-table* [LUT]. Therefore, if shrinking were to be performed, the pattern would be compared with all the patterns in the conditional LUT of shrinking process. Similarly, there are different LUT for thinning and skeletonizing.
4. An array of dimensions same as the image is set up. This array will hold the decision values of stage one for each corresponding pixel in the original image. This array, called the *conditional mark array*, is initially set to all zeros

5. If there is a pattern match, it is said to be a *hit,* and a logical 1 is saved in the corresponding location in the conditional mark array. If none of the patterns match, then nothing is changed in the conditional mark array.
6. This conditional mark array decides the extent to which an image is to be eroded. This is the basis for controlled erosion. Further, this conditional mark array is then passed to the second stage.

- **Stage 2:**

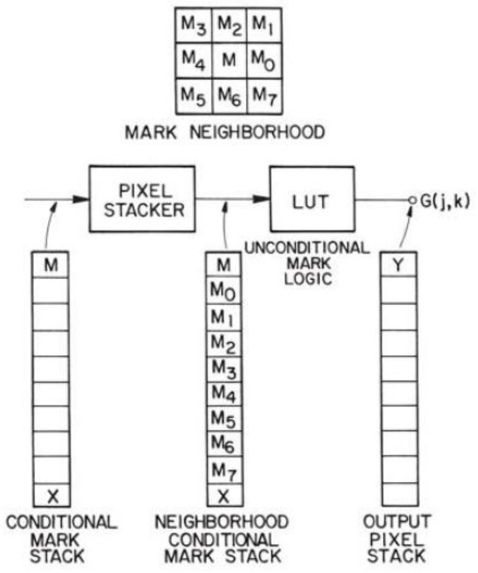


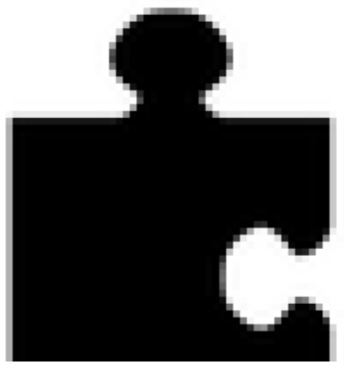
Figure 15. Stage 2 for *shrinking, thinning* and *skeletonizing*

1. Now, for each cell in the conditional mark array, if it is logical 1, then the neighbourhood of the cell is converted into a pattern (same as in stage 1).
2. This pattern is now compared with the patterns stored in the *unconditional pattern lookup table.* Each morphological process has a different unconditional pattern lookup table.
3. A copy of the original input image is saved in a temporary array (same dimensions as the input image).
4. For each cell in the conditional mark array with logical 1 intensity, the neighbourhood pattern is compared with all the patterns in the unconditional patterns in the unconditional LUT.
5. If the patterns match, it is a hit, then the intensity level at the corresponding location in the temporary array is changed. Else if it is miss, the cell in the temporary array is kept unchanged.
6. Now this temporary array is then sent back as the input to the first stage. This process loop is iterated until the expected eroded image (either shrinking, thinning, or skeletonizing) is obtained.

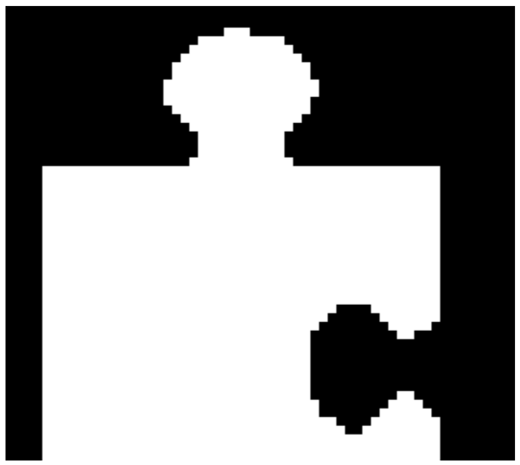
3.2.2 Counting Game:

There are two tasks in this part:

- **Counting total number of pieces:** This can be done using the *shrinking* procedure explained in section 3.2.1. The *shrinking* operation results in a single pixel centred at the centroid of each of the jigsaw piece. Furthermore, counting the number of such single pixels in the image will provide us with the total number of jigsaw pieces in the image
- **Counting total number of unique pieces:** This task has several different approaches. The approach that I used is as follows:
  1. Firstly, the input image is quantized to a binary image, with threshold intensity of 127. Thus, pixels with intensity level greater than 127 are set to 0 and pixels with intensity level less than or equal to 127 are quantized to 1. Thus, the quantized image looks something like:



(a) Part of the original image



(b) Binary image after quantization

Figure 16. Thresholding the input image

2. The quantized image becomes much sharper, and this can be observed in the above figure. This will help us to identify the corners of each of the jigsaw puzzle.
3. To locate the corners of the jigsaw pieces I used simple 3x3 pattern matching filters. Initially, the program looks for pixels with *intensity 1* (it must be noted that after quantizing the image, black is being represented as *intensity 1* and white is being represented as *intensity 0*). Thus, pixels with *intensity 1* will belong to object.
4. Once an *intensity 1* pixel is found, its neighbouring pixels are compared with one of the following 3x3 filters to determine whether it's a corner pixel, and if it is, then which corner.

$$P1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \qquad P2 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \qquad P3 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad P1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

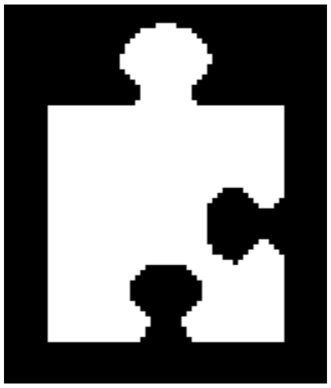
3x3 Matching Filter for *Top Right Corner*

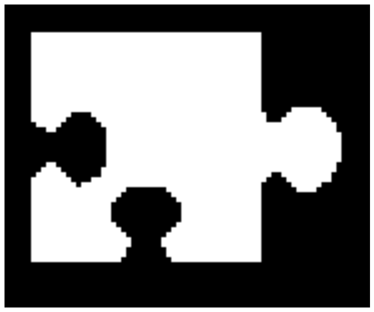
3x3 Matching Filter for *Top Left Corner*

3x3 Matching Filter for *Bottom Right Corner*

3x3 Matching Filter for *Bottom Left Corner*

5. The above step will give us the location of all the corners of all the jigsaw pieces. Furthermore, if the filters are convolved over the image in an iterative fashion (left to right, top to bottom), then we get the corner locations in a similar fashion too. Thus, the first corner locations that match the above filters will belong to the first jigsaw puzzle, second corner locations that match the above filters will belong to the second jigsaw puzzle, and so on.
6. Once the corner location for each jigsaw piece are located, it just a matter of traversing each edge of each of the jigsaw pieces to identify the jigsaw. Each jigsaw piece can be represented by a 4-digit pattern, each digit representing one edge of the jigsaw piece. If an edge has a *head*, it will be represented by digit '1'; if the edge has a hole, then it will be represented by digit '2'; else it will be represented by digit '0'.
7. Consider the jigsaw pieces shown below:


(a) Jigsaw piece # 2


(b) Jigsaw piece # 7

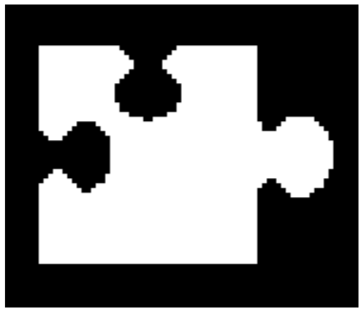
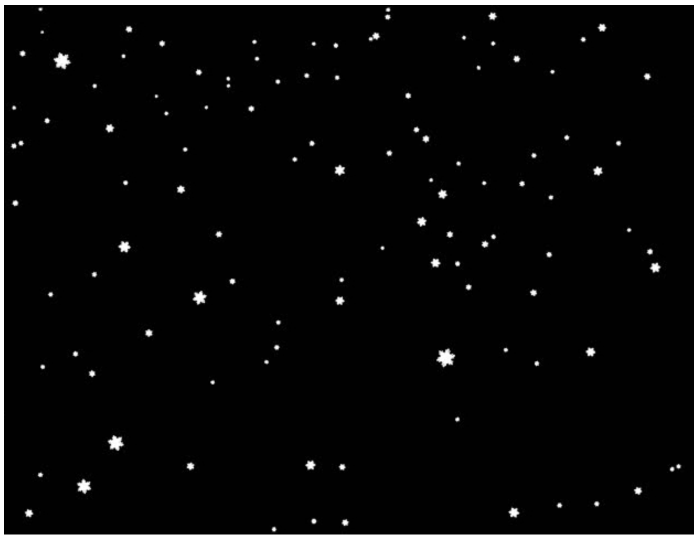

(c) Jigsaw piece # 11

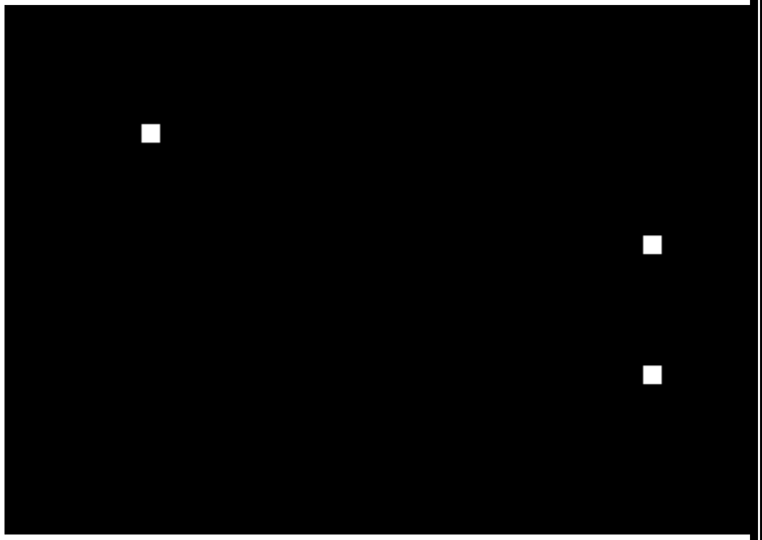
Figure 17. Similar Jigsaw Pieces

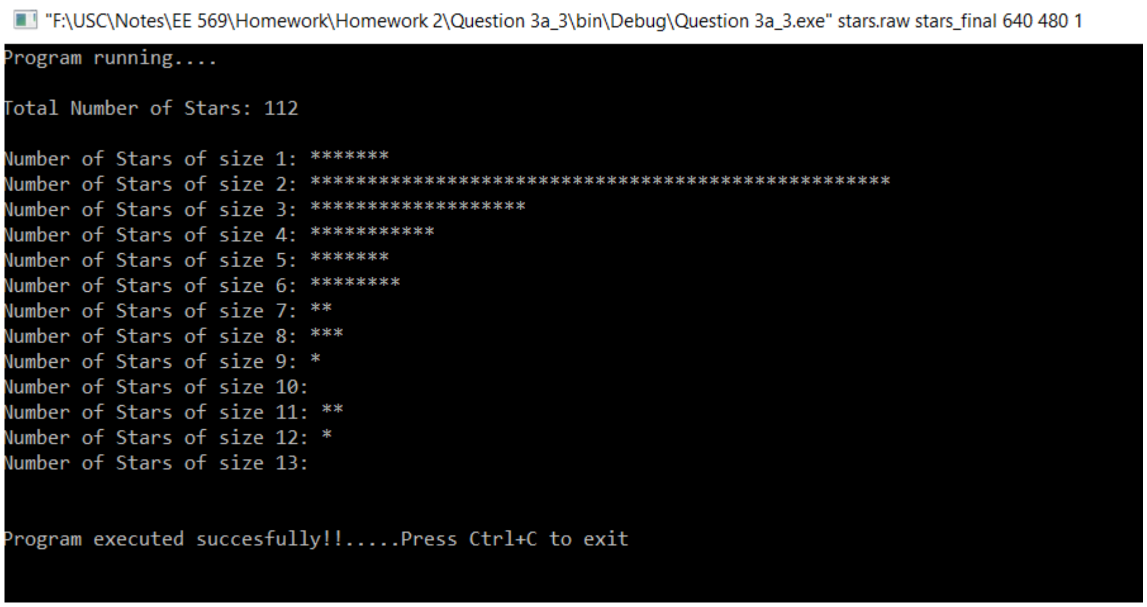
8. For this question, we are going to assume above jigsaw pieces to be the same. Jigsaw # 7 is a rotated version of jigsaw # 2 and jigsaw # 11 is a reflected version of the jigsaw # 7.
9. The 4-digit patterns for each of above jigsaw pieces is: (Considering we start looking for *heads* and *holes* from the top edge in a clockwise orientation)
    - Jigsaw # 2 = [1 2 2 0]
    - Jigsaw # 7 = [0 1 2 2]
    - Jigsaw # 11 = [2 1 0 2]
10. It can be observed that pattern for jigsaw #7 is cyclically rotated version of pattern for jigsaw #2. Similarly, the pattern for jigsaw #11 can be obtained be interchanging the first and third digits of the pattern for jigsaw #7 (which is nothing but reflection). In this way, we can compare patterns for each of the jigsaw puzzle by comparing with original patterns, rotated pattern, and the reflected patterns.
11. In this way we can find the unique jigsaw pieces

3.3 Experimental Results:

3.3.1 Shrinking:


(a) Original *stars.raw* image


(b) Output image after applying shrinking process (Zoomed 16x times)

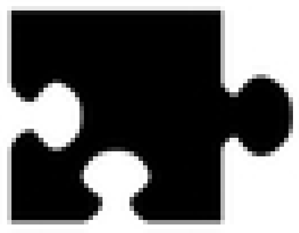


```
"F:\USC\Notes\EE 569\Homework\Homework 2\Question 3a_3\bin\Debug\Question 3a_3.exe" stars.raw stars_final 640 480 1
Program running....

Total Number of Stars: 112

Number of Stars of size 1: *******
Number of Stars of size 2: ****************************************************
Number of Stars of size 3: *******************
Number of Stars of size 4: ***********
Number of Stars of size 5: *******
Number of Stars of size 6: ********
Number of Stars of size 7: **
Number of Stars of size 8: ***
Number of Stars of size 9: *
Number of Stars of size 10:
Number of Stars of size 11: **
Number of Stars of size 12: *
Number of Stars of size 13:


Program executed succesfully!!.....Press Ctrl+C to exit
```

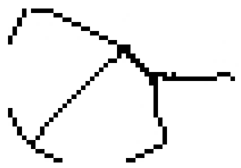
(c) Counting the number of stars and histogram of size of stars

Figure 18. Output for the part (a) of Problem 3

3.3.2 Thinning:



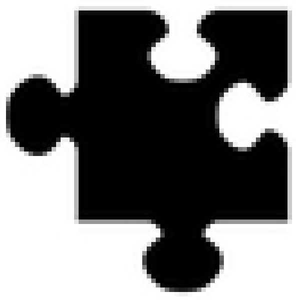
(a) Original *jigsaw_1.raw* image



(b) Output image after applying thinning morphological filter

Figure 19. Output for the part (b) of Problem 3

3.3.3 Skeletonizing:


(a) Original *jigsaw_2.raw* image

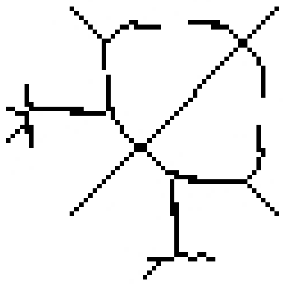
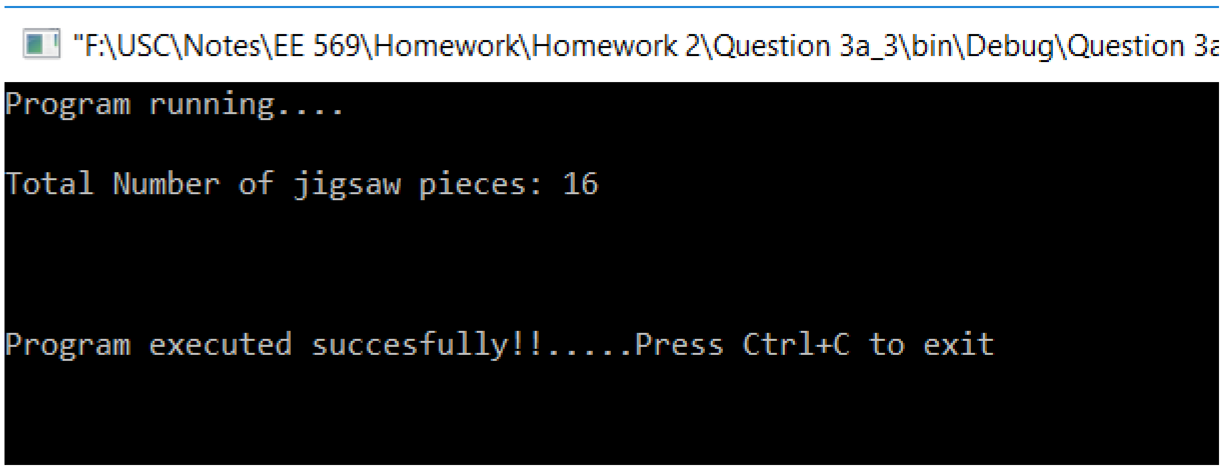

(b) Output image after applying skeletonizing morphological filter

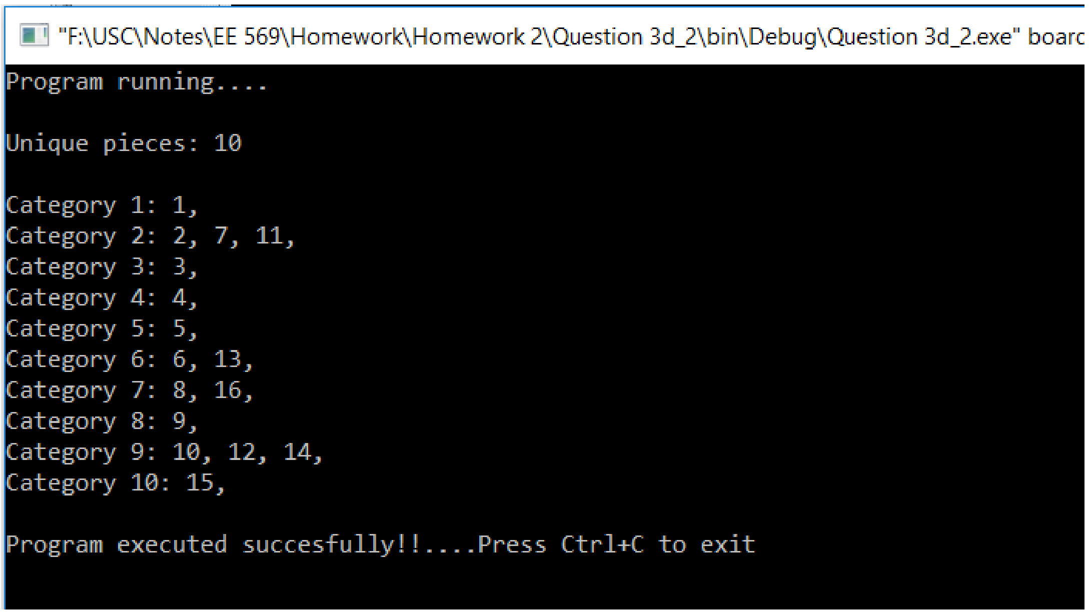
Figure 20. Output for the part (c) of Problem 3

3.3.4 Counting Game:



```
"F:\USC\Notes\EE 569\Homework\Homework 2\Question 3a_3\bin\Debug\Question 3a
Program running....

Total Number of jigsaw pieces: 16



Program executed succesfully!!.....Press Ctrl+C to exit
```

(a) Counting the total number of Jigsaw pieces in the board.raw image



```
"F:\USC\Notes\EE 569\Homework\Homework 2\Question 3d_2\bin\Debug\Question 3d_2.exe" boarc
Program running....

Unique pieces: 10

Category 1: 1,
Category 2: 2, 7, 11,
Category 3: 3,
Category 4: 4,
Category 5: 5,
Category 6: 6, 13,
Category 7: 8, 16,
Category 8: 9,
Category 9: 10, 12, 14,
Category 10: 15,

Program executed succesfully!!....Press Ctrl+C to exit
```

(b) Counting the total number of unique Jigsaw pieces in the board.raw image

Figure 21. Output for the Counting Game

3.4 Discussion:

- **Shrinking, Thinning and Skeletonizing:** For the first part of counting the stars in the image, shrinking has been used. I observed that changing the threshold value for creating a binary image affects the count. If the threshold is changed from 127 to 128, the count of stars changes. Thus, the value of threshold is very important when it comes to performing morphological operations.
  Morphological operations are used in several different applications like pruning, nose removal, boundary extraction etc. Morphological operations are also used in biomedical applications, recognition of handwritten digits, License plate detection, text extraction, detection of imperfections in printed circuit boards etc.

- **Counting Game:** There are different other approaches for finding the number of unique pieces. We can use *Harris Corner Detection* algorithm to detect the corners of the jigsaw pieces. Once the corners are detected, we can use the same approach discussed before to find the number.
  Another approach would be use the corners to find the location of each of the jigsaw pieces. Then calculate the *Hausdorff Distance* between each of the jigsaw pieces. *Hausdorff distance* can be used to find a given template in an arbitrary target image. It can also be used to find the similarity between two images.

References:

1. http://caca.zoy.org/study/part2.html
2. Digital Image Processing – William Pratt
3. Analytical Methods for Squaring the Disc
4. http://people.scs.carleton.ca/~c_shu/Courses/comp4900d/notes/homography.pdf
5. Colour Diffusion: Error-Diffusion for Colour Halftones – HP Labs