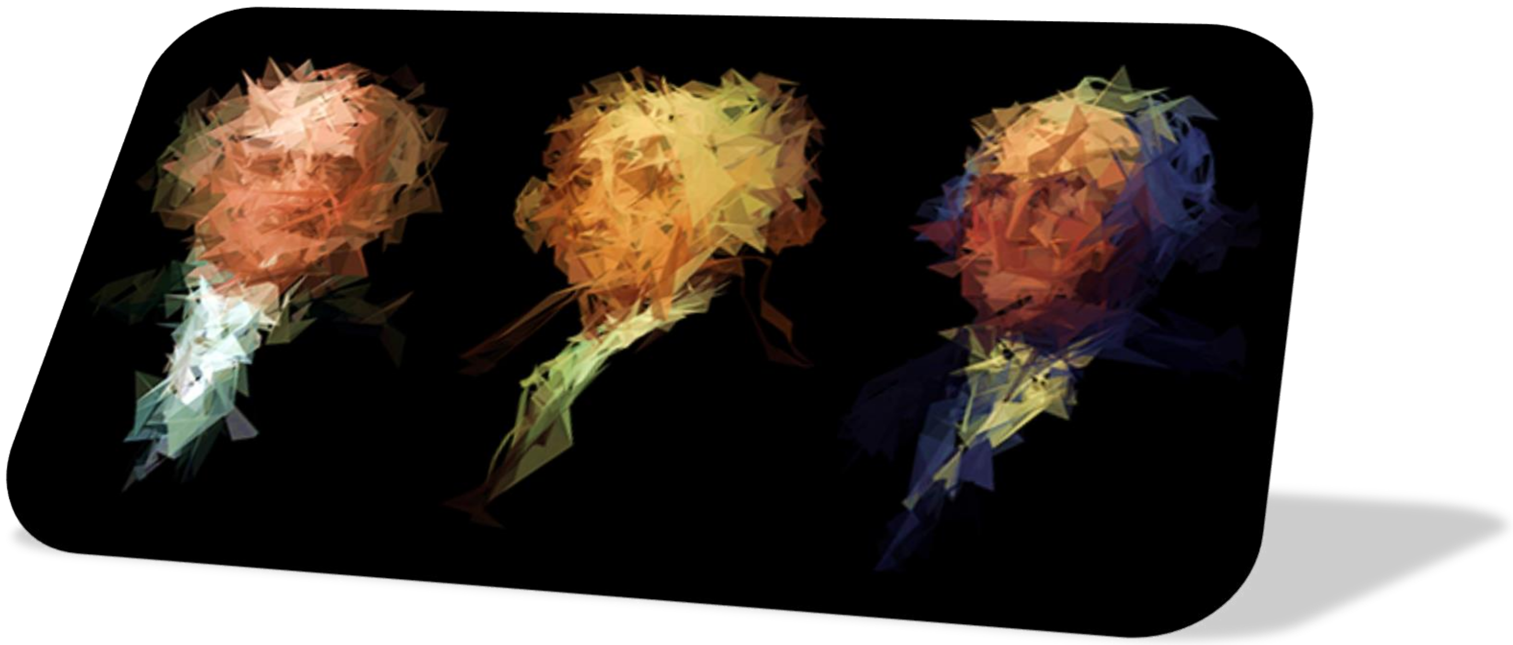
EE 569

HOMEWORK #2



SHIVA SHANMUGAPPA PATRE

spatre@usc.edu

6129918110

1. Problem 1

1.1 Motivation

Image Processing, as the name says is the processing or manipulation of the images in order to get a better quality image or an enhanced version of the same image. One of the most common operations in Image Processing is Geometrical manipulation. By changing the pixel location while keeping the colors unchanged, we can realize certain beautiful visual effects. This consists of translation, scaling, rotation etc. to achieve the desired effect. It is an important application of 3D and 2D Computer graphics.

In this homework, we must implement geometrical warping technique that transforms an input Square image into an output disk-shaped image. The pixels which lie on the boundaries of the square will lie on the boundaries of the circle in the disk image. The center of the square image will be mapped exactly to the center of the disk image. We also implement a reverse mapping, wherein we go from disk to square with very minimal loss in the pixel data.

Another interesting application of Image processing which is probably one of the most liked and used application is the Image Stitching or the classic Panorama. The user takes multiple shots to cover a wide angle and advanced softwares are used to composite this into a single image which overlap and create a stitched image effect. In this homework, we need to implement Homographic transformation and Image Stitching to achieve the Panorama effect.

So let us discuss the approach to be taken to implement the above method in detail. Let the fun begin.

1.2 Approach

There are two parts in this section. The Geometrical Warping is described in section 1.2.1, and the Homographic Transformation and Image Stitching is described in 1.2.2.

1.2.1 Geometrical Warping

Since the invention of the wheel, the circle and square are like the most important shapes ever used by mankind. Therefore it makes sense to study the transformations between the two. We implement the back and forth conversion of square to disk and vice versa in this problem. To manipulate the image for the above operation, we need to convert to the cartesian co-ordinates. Suppose the image co-ordinates are **(u,v)** and the Cartesian co-ordinates are **(x,y)**, then to convert the square to disc we use the following formula.

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

To convert the disk back to square we use the below formula.

$$x = \frac{1}{2}\sqrt{2+u^2-v^2+2\sqrt{2}\,u} - \frac{1}{2}\sqrt{2+u^2-v^2-2\sqrt{2}\,u}$$

$$y = \frac{1}{2}\sqrt{2-u^2+v^2+2\sqrt{2}\,v} - \frac{1}{2}\sqrt{2-u^2+v^2-2\sqrt{2}\,v}$$

To implement the above formulas we need to normalize the image to lie in the interval [-1,1]. So we left shift the image by 256 and divide by 256, to convert to the Cartesian co-ordinates. Then the formula is applied to convert from Square to Disk and we need to convert the Cartesian back to Image co-ordinates. So we multiply by 256 and right shift by 256. We repeat the same for all the three channels.

The reverse is also similar, and we use a different formula as stated above. But it is slightly different in the sense that we have to only map back the pixels inside and on the boundary of the circle to the square image (i.e, u*u + v*v <=1). But as we know there will be a lot of missing pixels and a circle has a lesser area than the square in this case. So to fix that I have used Bilinear interpolation and will discuss it in the discussing part of the problem.

1.2.2 Homographic Transformation and Image Stitching

The Homographic transformation and Image stitching give us the desired panorama effect. In this problem we are given three input images left, middle and right as Shown in Figure below. The approach is pretty intense with a lot of calculations. So here goes the approach. I first take the middle image and make it into a bigger image so as to accommodate the other images to get a stitched image. I basically pasted the 480x640 Image into a 1800x1400 all black image. This would form a big middle image. Then I found the control points manually using **cpselect()** command in MATLAB by passing the images in pairs. This is shown in **Figure 1b_CP**. The below matrix is used to find the missing points from the control points and thus the H inverse matrix.

$$\begin{vmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -y_2X_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3X_3 & -y_3X_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4X_4 & -y_4X_4 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1Y_1 & -y_1Y_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2Y_2 & -y_2Y_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3Y_3 & -y_3Y_3 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4Y_4 & -y_4Y_4 \end{vmatrix} \bullet \begin{vmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{vmatrix} = \begin{vmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{vmatrix}$$

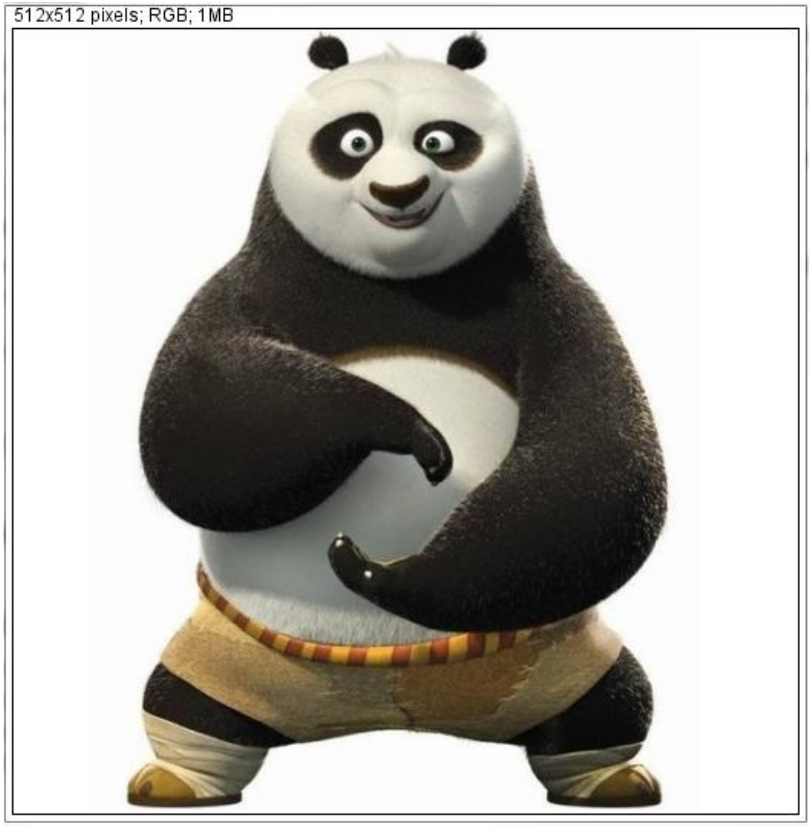
First the Left and BigMiddle and then the BigMiddle and Right. The control points were chosen meticulously so as to not form a distortion later on. I Chose **four** control points in each image and tried matching at the same locations on the other image. This will give 8 equations to solve, four equations from four control points. So using these equations I calculated H matrix. From this I calculated H inverse matrix.

$$\begin{bmatrix} x_2' \\ y_2' \\ w_2' \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \quad and \quad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \frac{x_2'}{w_2'} \\ \frac{y_2'}{w_2'} \end{bmatrix}$$

So now I got two H inverse matrices for left and middle and right and middle. Now the task is to stitch all three images together and I take the three inputs as the left image, Big Middle image and right image. Now I had to calculate the X and Y co-ordinates using the below formula given in the Homework doc.

Now I map the value of the left image to the big image only if X lies in (0,480) and Y lies in (0, 640). Now I repeat the same steps as above but with the right and Middle Big H inverse matrix and embed the right image onto the result from the previous step. So this will give us the desired Panoramic output as can be seen in **Figure 1b_Panorama.**

1.3 Results



Original Panda Image

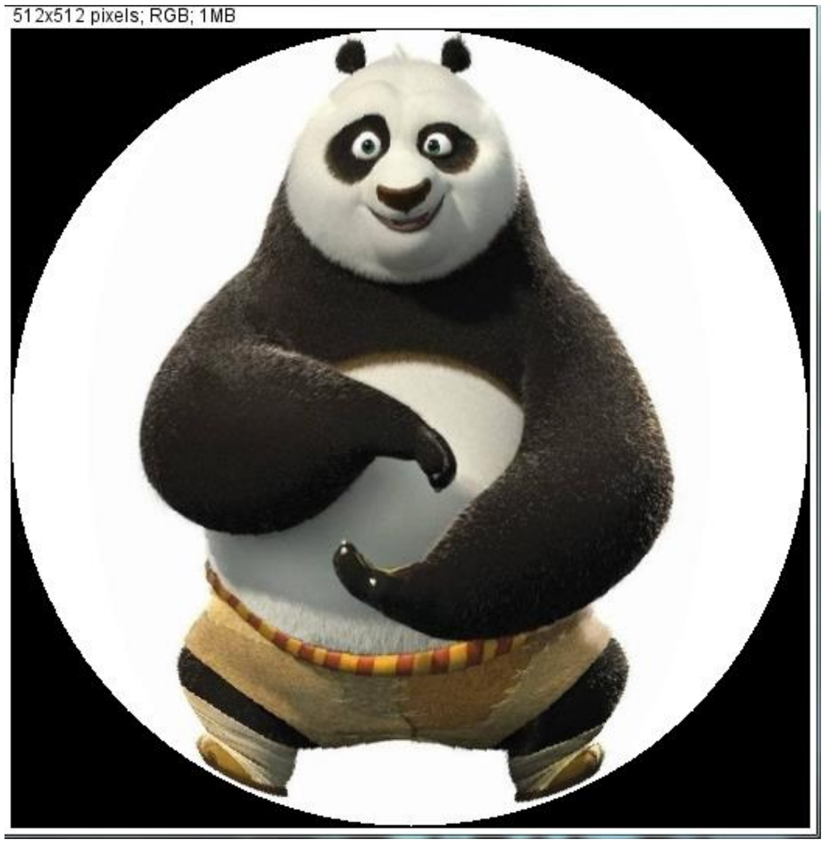


Figure 1a

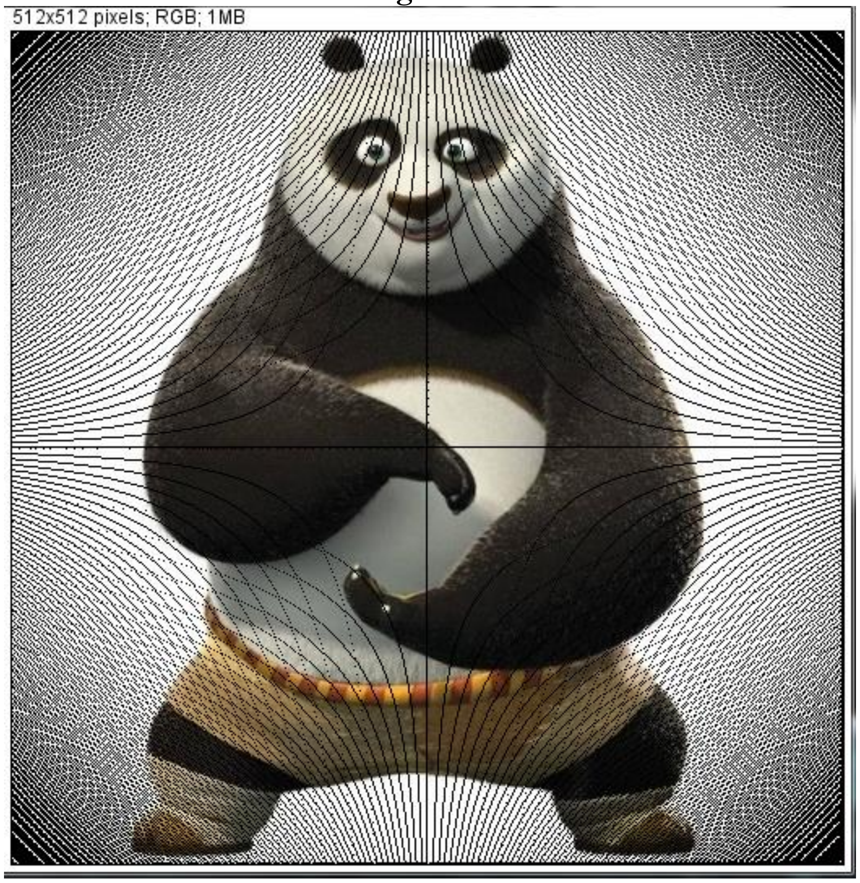


Figure 1b

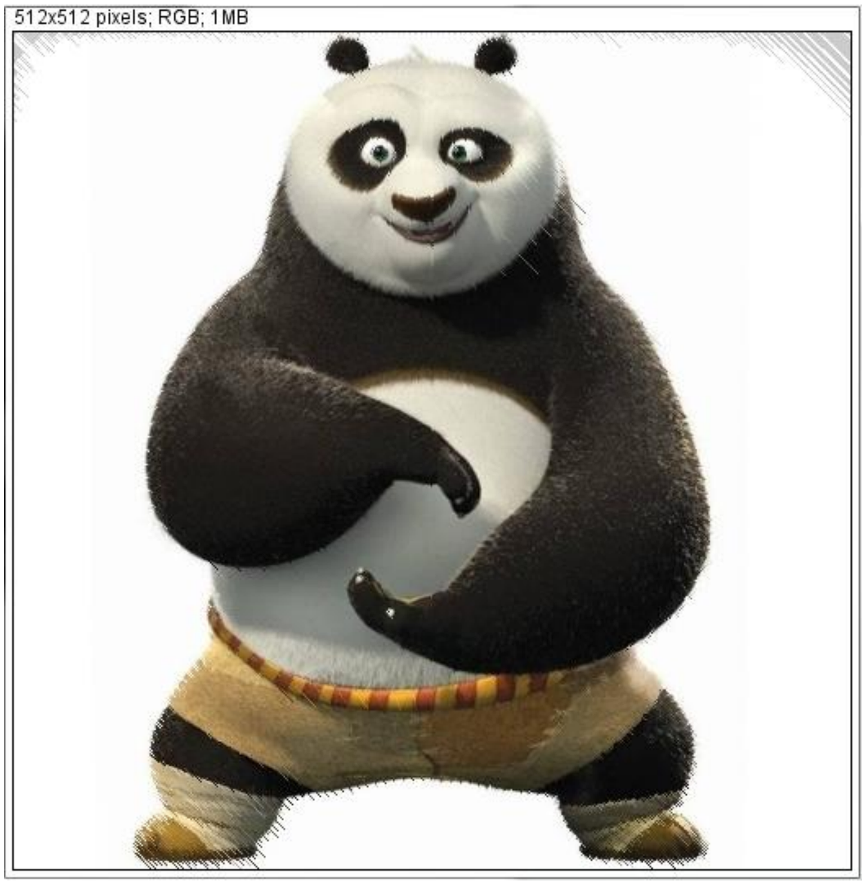
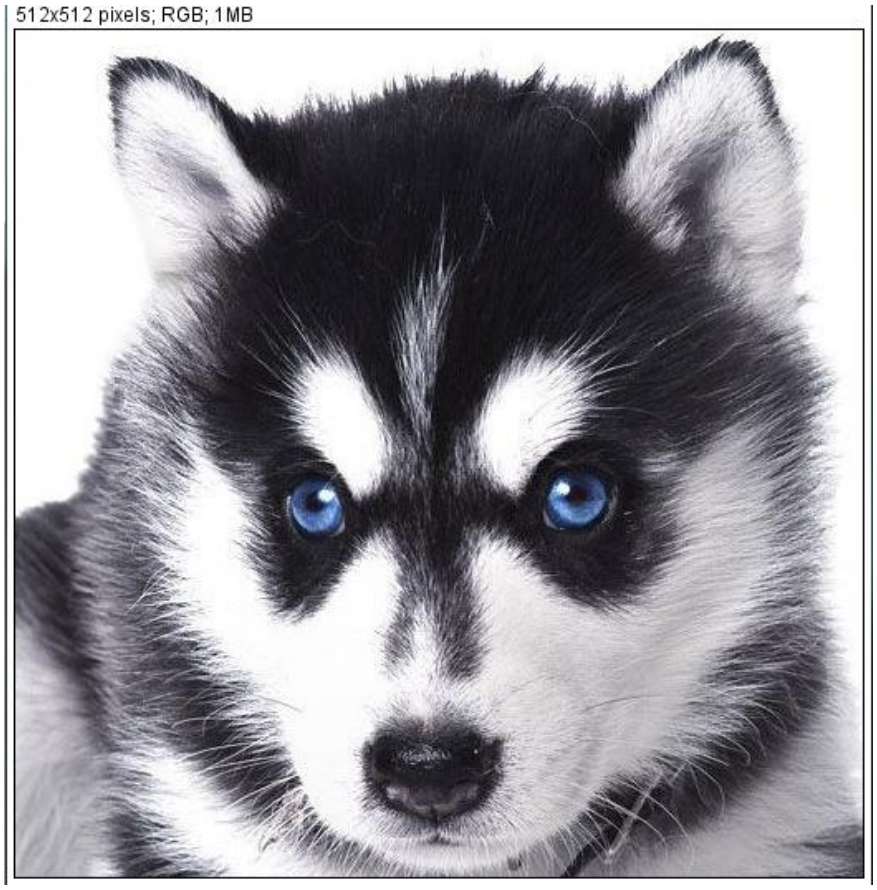


Figure 1c



Original Puppy Image

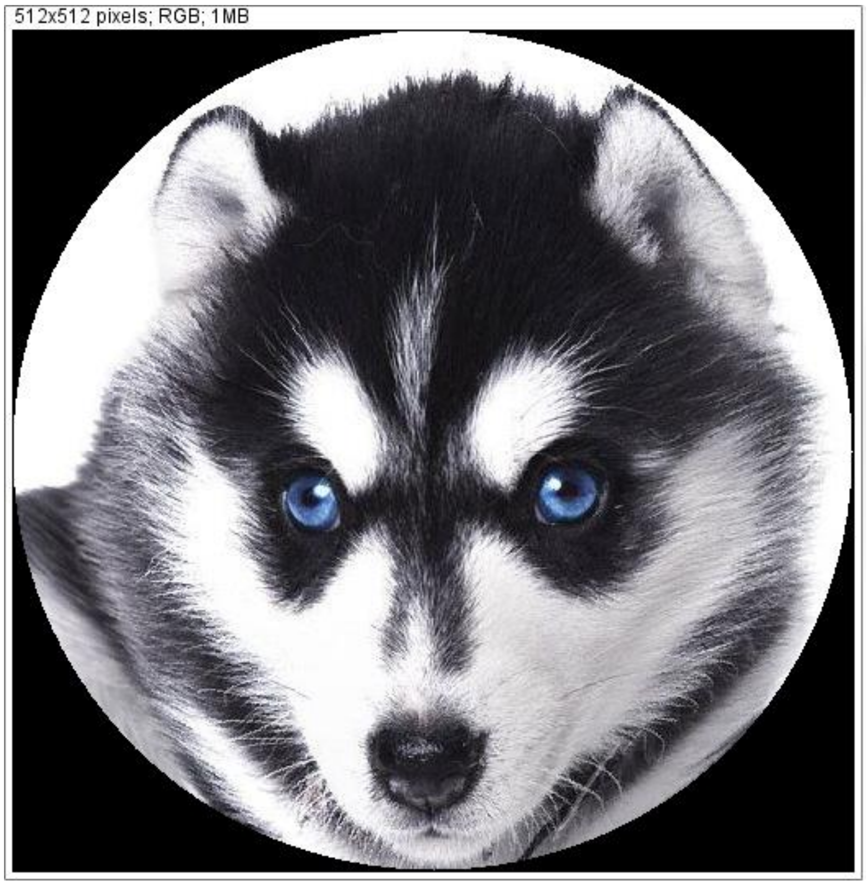


Figure 2a

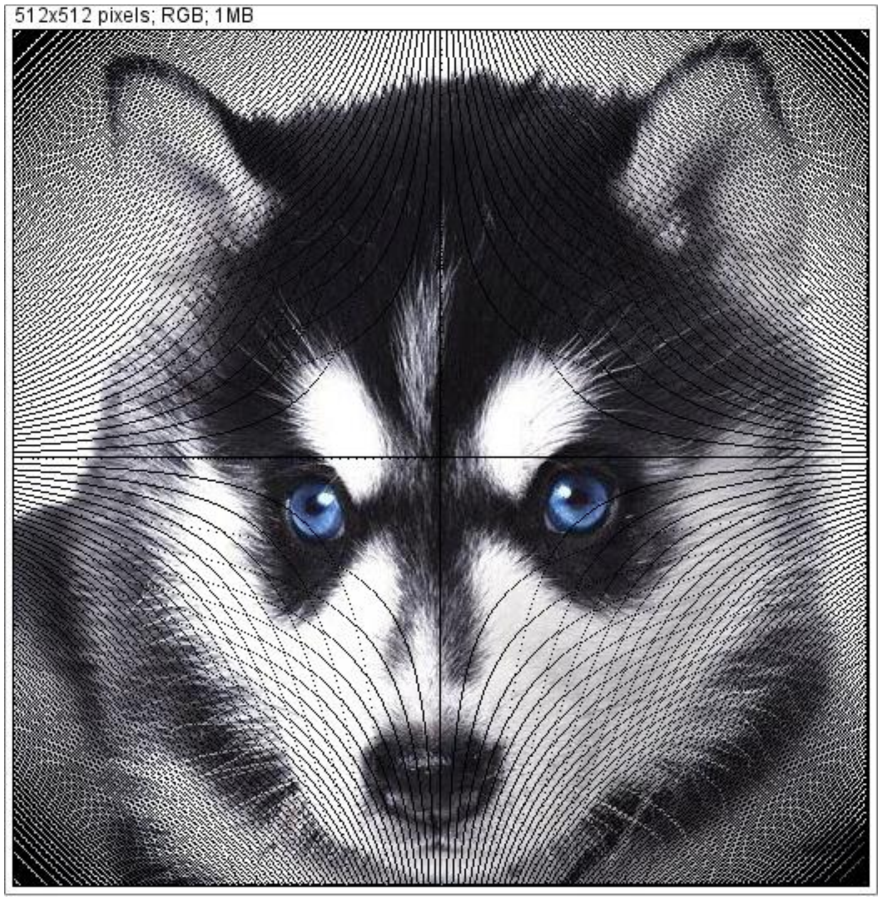


Figure 2b

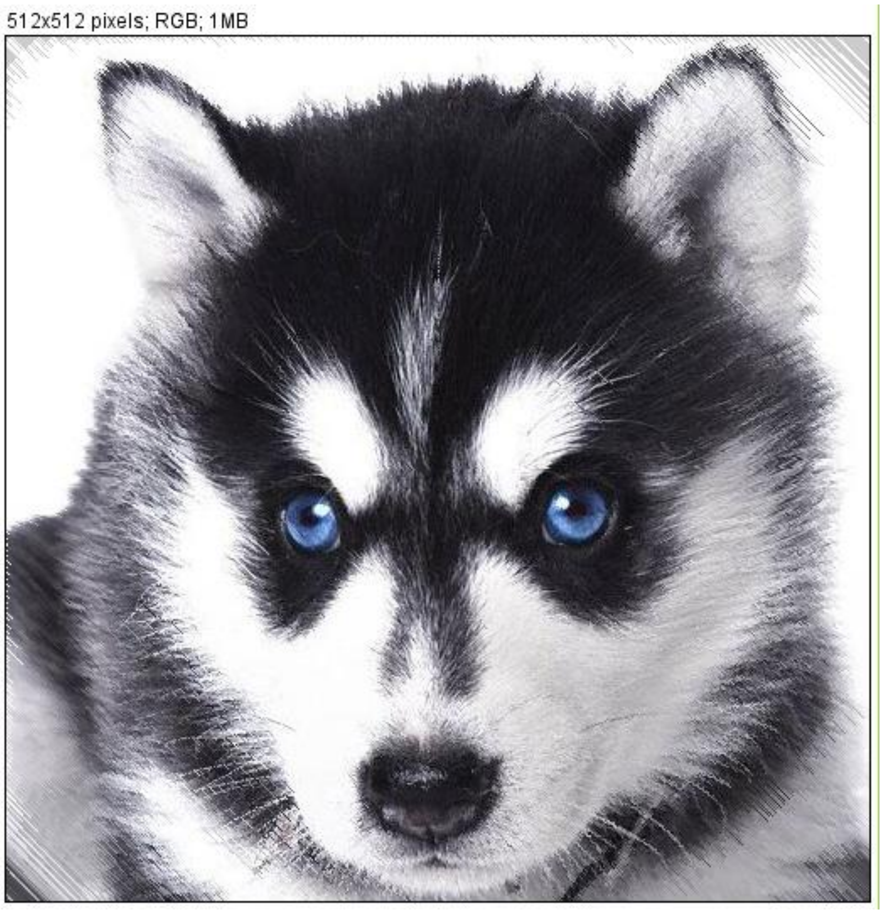
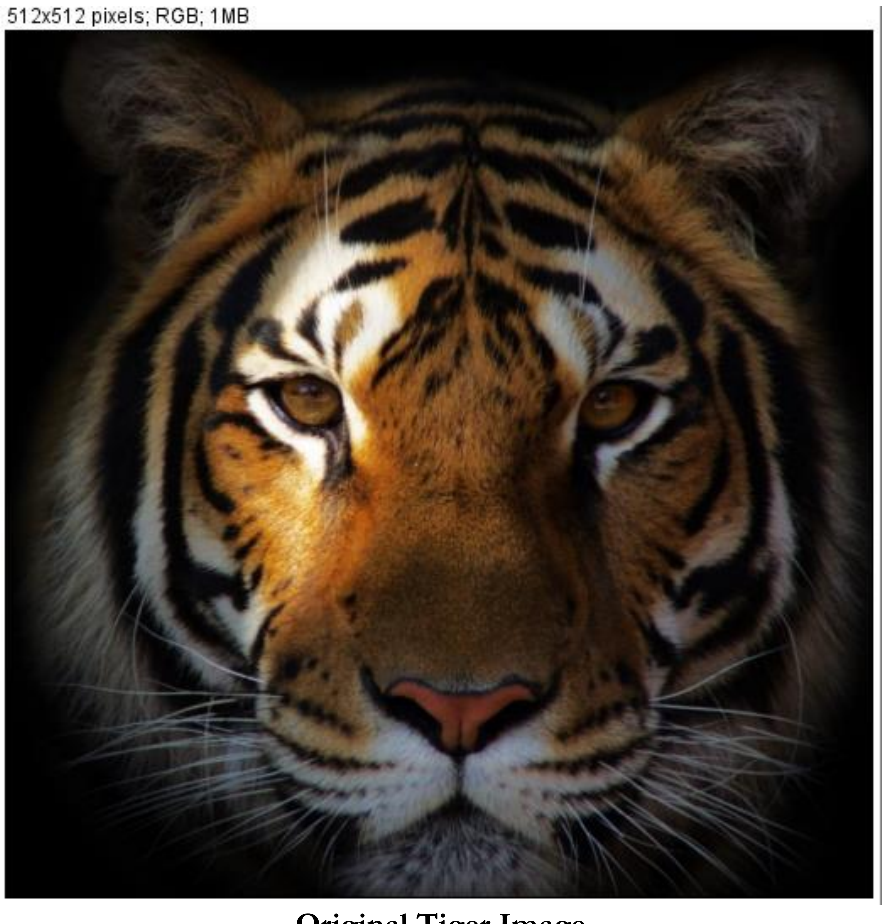


Figure 2c



Original Tiger Image

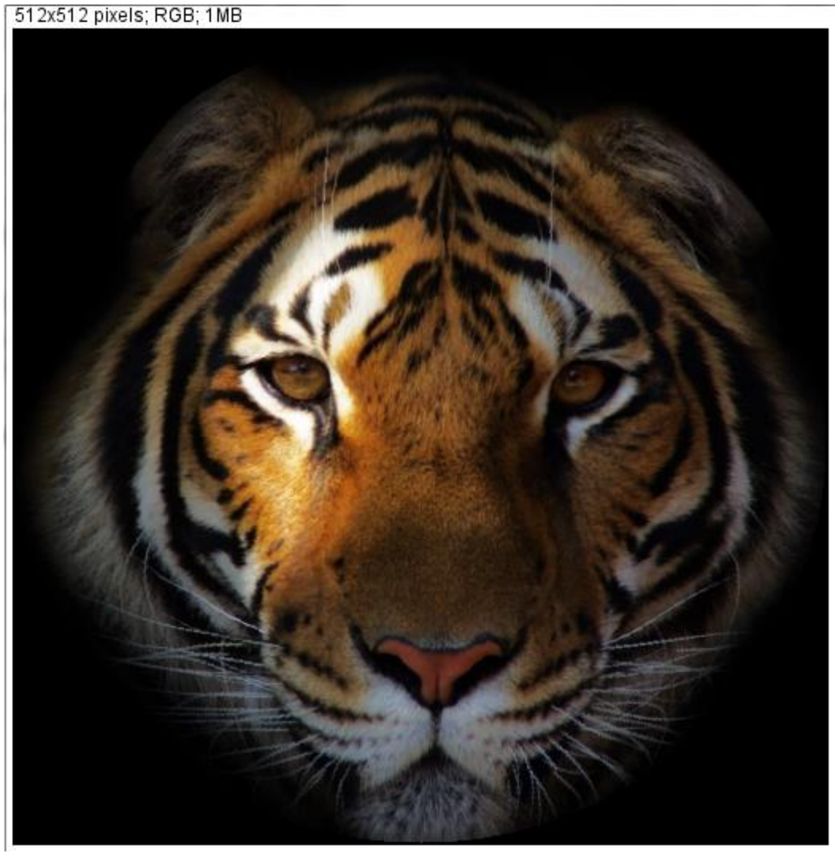


Figure 3a

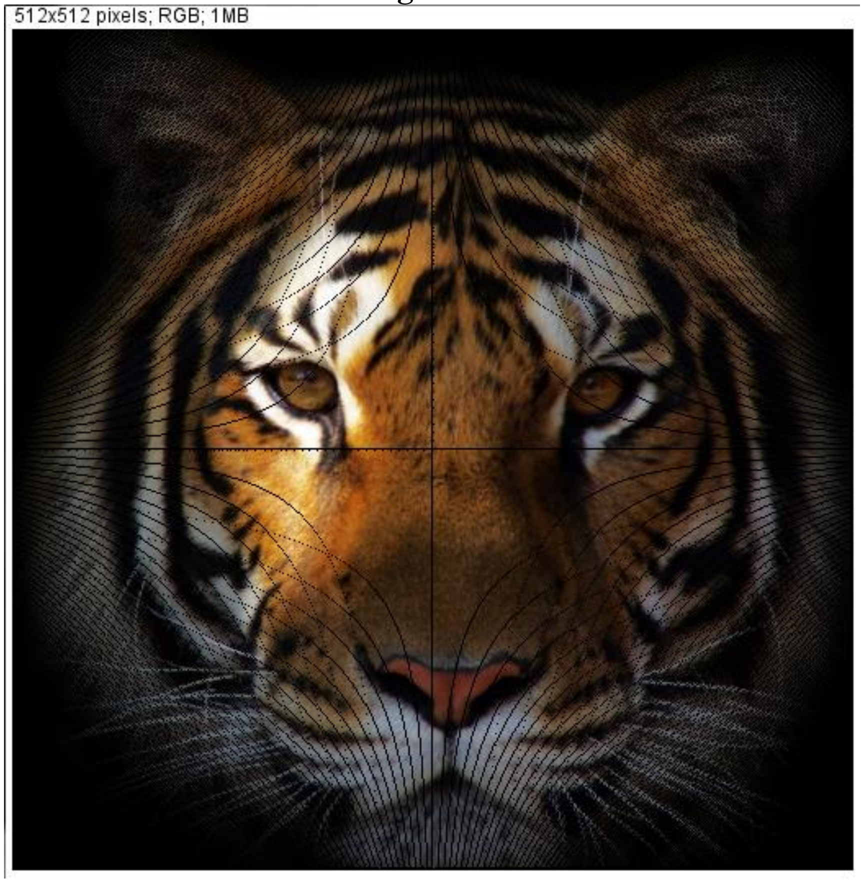


Figure 3b

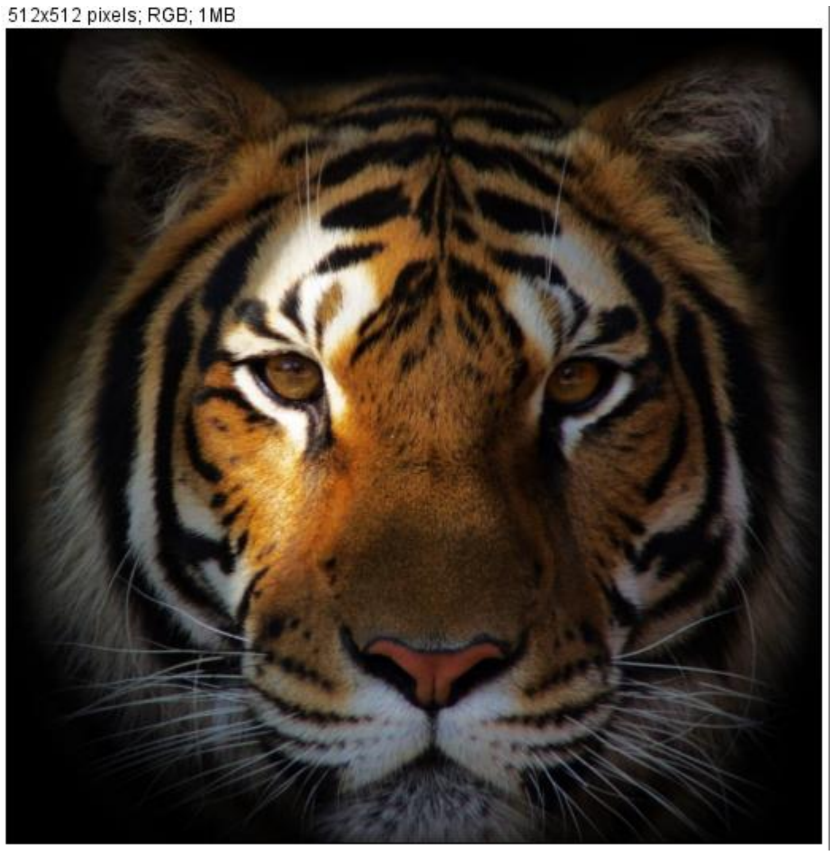


Figure 3c

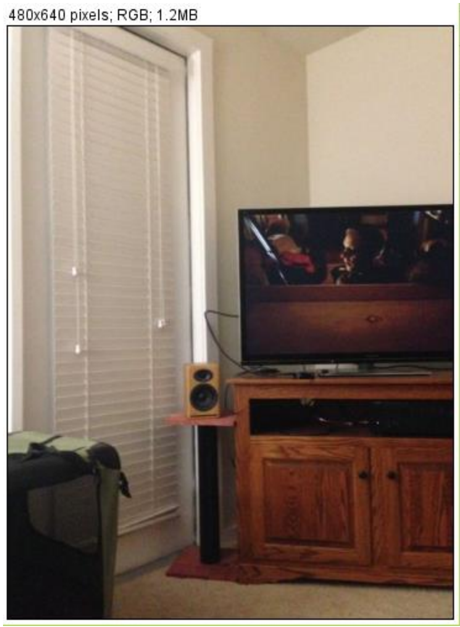


Figure 1b_left

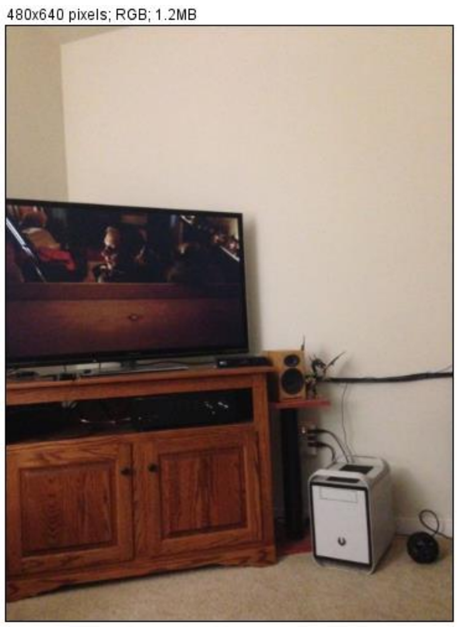


Figure 1b_middle

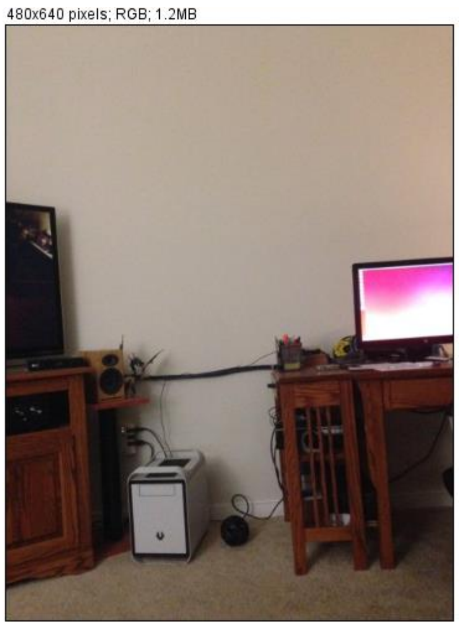


Figure 1b_right

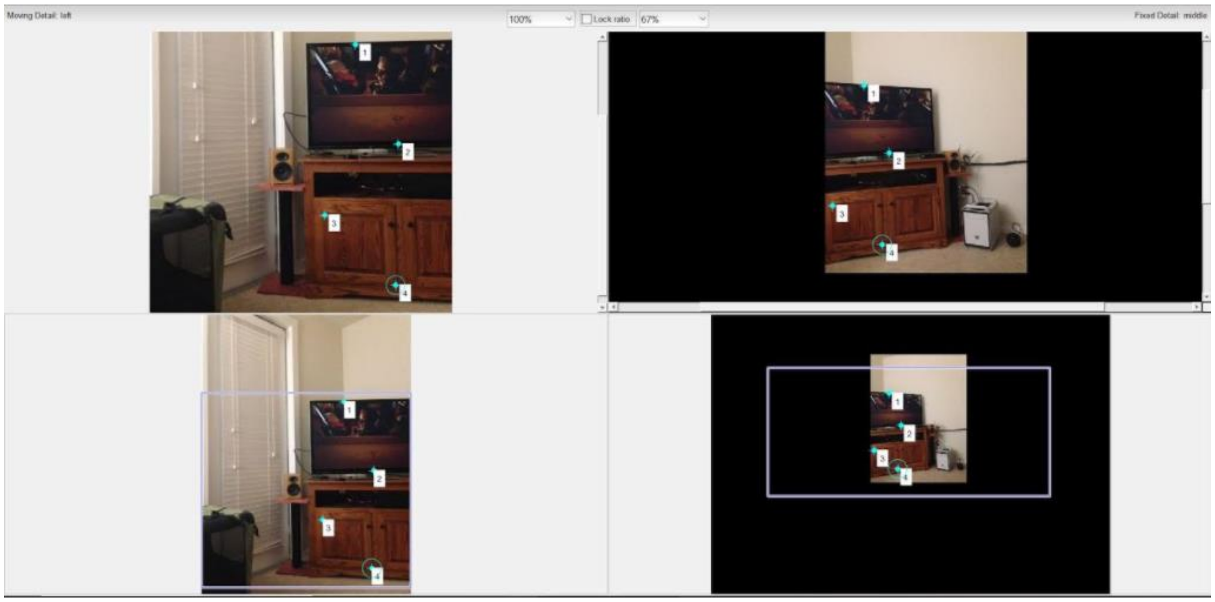


Figure 1b_CP

1800x1400 pixels; RGB; 9.6MB

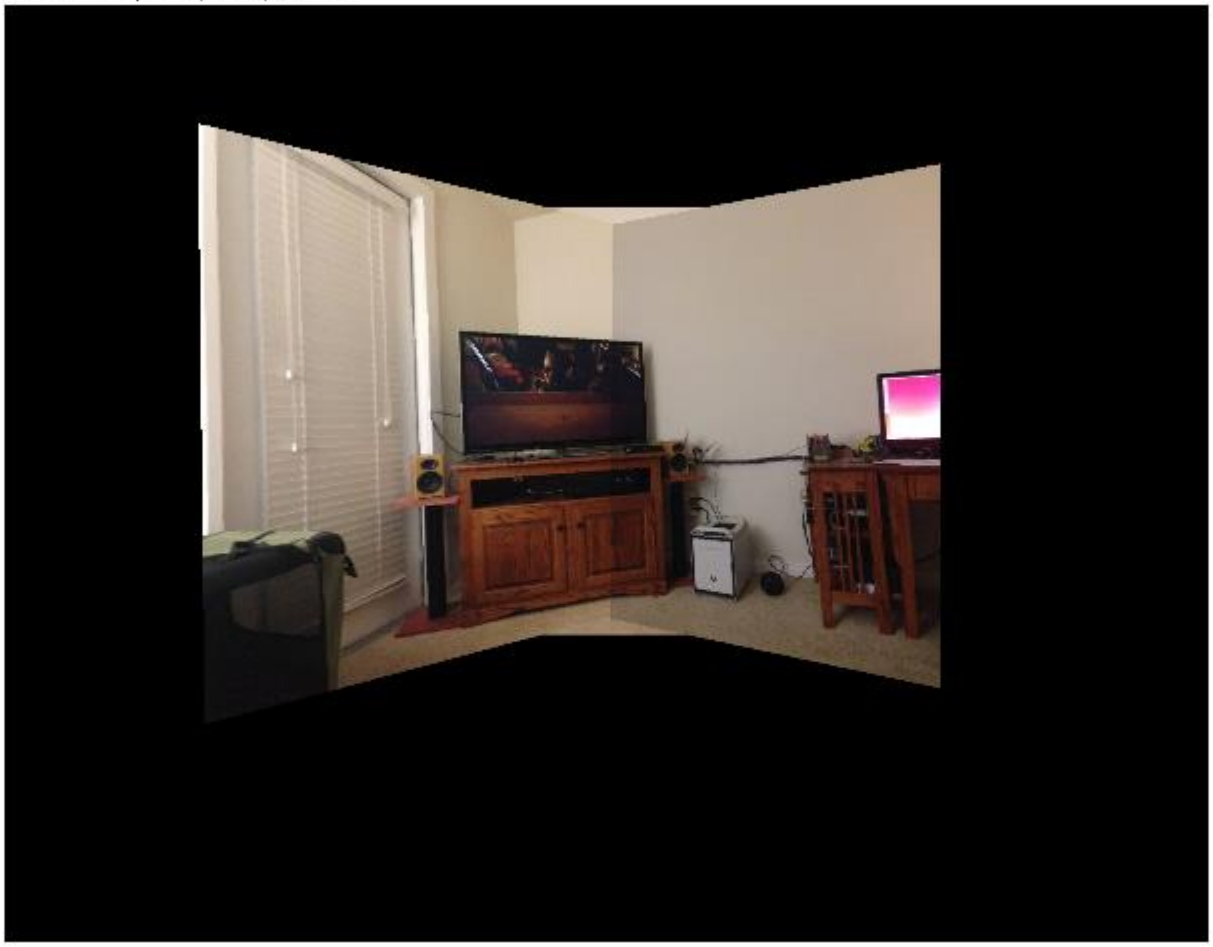


Figure 1b_Panorama

1.4 Discussion

- **Geometrical Warping**

  The results are as above. The input original images are the Panda, Puppy and the Tiger.

  Figure 1a, 2a, 3a give the required output as asked in the problem to convert the square image into a disk shaped image. The outputs are good enough as I have used Elliptical Grid mapping, and the outputs are as expected.

  So the Original image is now mapped into a circle and the area outside the circle is black as I have initialized the outer part to 0. But the real issue pops up when we invert the process, to transform the circle back to square. As the are of the circle is less than the area of the square in this case, there will be a lot of missing pixels and we can see black pixels in the reverted square image as shown in **Figure 1b, 2b, 3b**. This is not a desired result and is a **distortion** obtained due to the missing pixels. Hence the recovered Square image is not the same as the original square image.

  To fix this, I have applied Bilinear Transformation combined with a logic which will fill in a neighboring pixel if an empty or missing pixel is found. Thus the image can be made better and is almost comparable to the original images. The output of this step is as seen in **Figure 1c, 2c, 3c**.

  Therefore I have implemented geometrical warping and also inverted the transformation to get back a similar image to the original image. Hence, we can have fun trying out various warping methods and can make interesting apps for the real world for people to have fun with.

- **Homographic Transformation and Image Stitching**

  The output can be seen in **Figure 1b_Panorama** above and is found to be the desired output. It is a stitched image which is formed from three input files. The following are the discussion points for the questions in the homework.

  1) I used **Four** control points for each image in this problem. If we use more control points the preciseness or the accuracy of the stitched image will increase as feature matching will be accurate as the number of control points increase. But we need a minimum of 4 control points. The more the control points, the better is the result.
  2) I selected the control points manually using MATLAB as shown in **Figure 1b_CP.** This is the difficult part of the problem as I had to choose very carefully. The points shouldn't be too close or too far but should be distributed in between as shown. This is repeated for both left and right images and when we put the value of the H inverse matrix , we will be able to calculate the X and Y coordinates for the stitched image, Ultimately we get the desired stitched image as the output.

     Thus Homographic transformation and Image stitching got me to learn a lot about matrices and made me revisit the concepts of linear algebra. This can be used to create entertaining images and Funny Snapchat filters which are widely liked and used by millions of people all over the world.

2. Problem 2

2.1 Motivation

Digital Half-toning is one of the most widely used application of Image Processing. This is mainly used in the print industry. Since printing presses need to print a lot of copies with thousands of images, it makes sense to save the ink while also not taking away the image quality. This is where half toning comes into picture. So for a black and white image, by only having fewer pixels we can represent the image visually indifferent. When we view half toned images from a distance, the dots will blur together and creates an illusion of continuous shapes and lines and the observer will not feel the reduction in the pixel density. The density of the dots can be used to control the darkness or lightness of a region in the image. The main intent of half toning to reduce the cost incurred from the ink. Hence in this problem, I implement half toning by Dithering, Error Diffusion and Color half toning using certain techniques as explained below.

2.2 Approach

2.2.1 Dithering

In this there are three methods to be implemented. The Fixed thresholding, Random thresholding and the Dithering matrix.

Fixed Thresholding basically converts the given grayscale image into a binary image consisting of only two colors , black and white. The idea is simple. Any pixel value less than 127 is converted to 0(black) and any value above 127 is converted to 255(white) using the below condition.

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

Random Thresholding is similar to the above but doesn't use a fixed value but a random value. We generate a random value in the range 0-256 and compare the pixel value with this random value. If pixel value is less than this random value then we assign a 0 else 255 using the below condition.

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

Dithering Matrix is an important concept. Fixed and Random thresholding gives a flat image and isn't visually attractive. So another nice way to do it is by adding blue noise to the image and then quantizing it. But adding noise is a pain. So dithering technique uses multiple quantization thresholds. We can do this by setting different thresholds for different pixel values in a region to do binary quantization. The method we use is the Bayer index matrix. We use the following formula for the same.

$$I_{2n}(i,j) = \begin{bmatrix} 4 \cdot I_n(x,y) + 1 & 4 \cdot I_n(x,y) + 2 \\ 4 \cdot I_n(x,y) + 3 & 4 \cdot I_n(x,y) \end{bmatrix}, \text{ where } I_2(i,j) = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$$

All we need to do is to slide this matrix over the image. and determine the threshold for each pixel using the formula.

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

Where N is the size of the Bayer index matrix.
Once we get the threshold we calculate the pixel value using the below condition.

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

We repeat the above for different sizes of the Bayer matrix and the outputs are seen accordingly.

2.2.2 Error Diffusion

Error diffusion is one amazing method to get the blue noise in the pixels. The basic idea is to calculate the error between the quantized output and the normalized input image. Once we get the error for each pixel we can dissipate the error using different kind of diffusion matrices to its neighboring pixels. The error diffusion is from left to right. But serpentine scanning does it from left to right and then from right to left and so on like a serpent. By adding this diffused error to each pixel, we can compare this with the threshold and do a binary quantization accordingly. There are three diffusion matrices which are used in this problem. They are the Floyd-Steinberg's error diffusion, Jarvis, Judice and Ninke(JJN) and the Stucki method whose matrices are as below in that order.

$$\frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix} \quad \frac{1}{48}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix} \quad \frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

More details will be discussed in discussion below.

2.2.3 Color Halftoning With Error Diffusion.

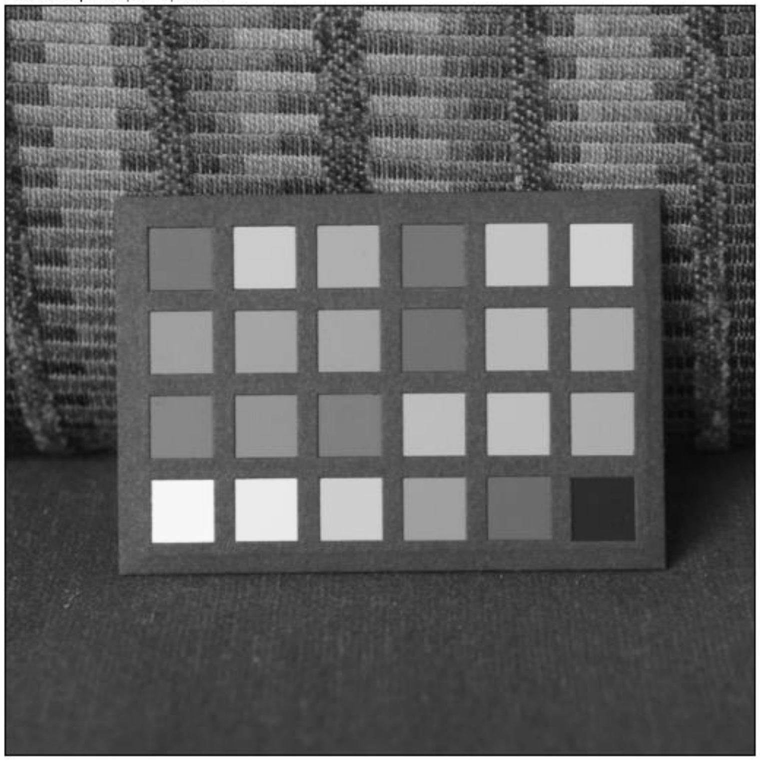
In this problem, first I converted RGB image to CMY space. There are 8 colors in CMY space namely W(0,0,0), K(1,1,1), Y(0,0,1), B(1,1,0), R(1,0,1), C(0,1,0), G(0,1,1) and M(1,0,0). The (W, K), (Y, B), (R, C), and (G, M) are the complementary colors.

Separable error diffusion is similar to FS method above. I just converted the RGB to CMY and applied FS matrix for all the channels separately. I obtained the output halftoned color image.

In MBVQ based error diffusion, CMY space can be divided into 6 Minimum Brightness Variation Quadruples(MBVQ). We then set the error for each pixel as 0. Based on the CMY value, we can place each pixel in one of the six quadruples of the MBVQ. We then find the vertex of this MBVQ closest to CMY value of the pixel plus the error. Once this vertex is found , the error is the difference of the CMY value of the pixel plus the error and the vertex value. We then use error diffusion technique to diffuse the error to neighboring pixels. More is discussed in discussion below.

2.3 Results

512x512 pixels; 8-bit; 256K



Original Color checker image

512x512 pixels; 8-bit; 256K

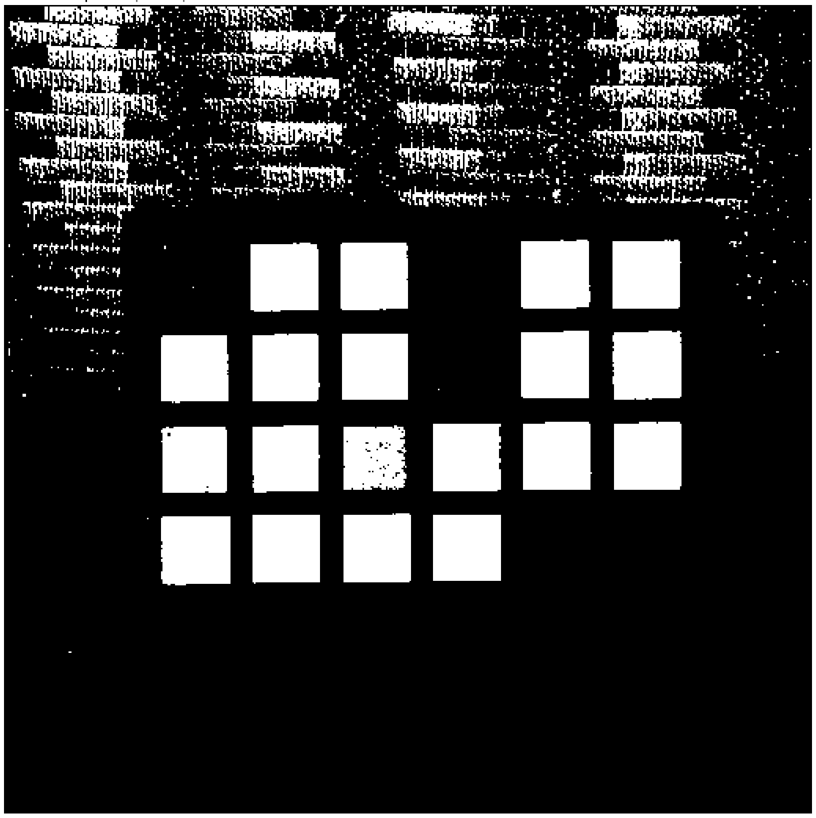


Figure 2a_1

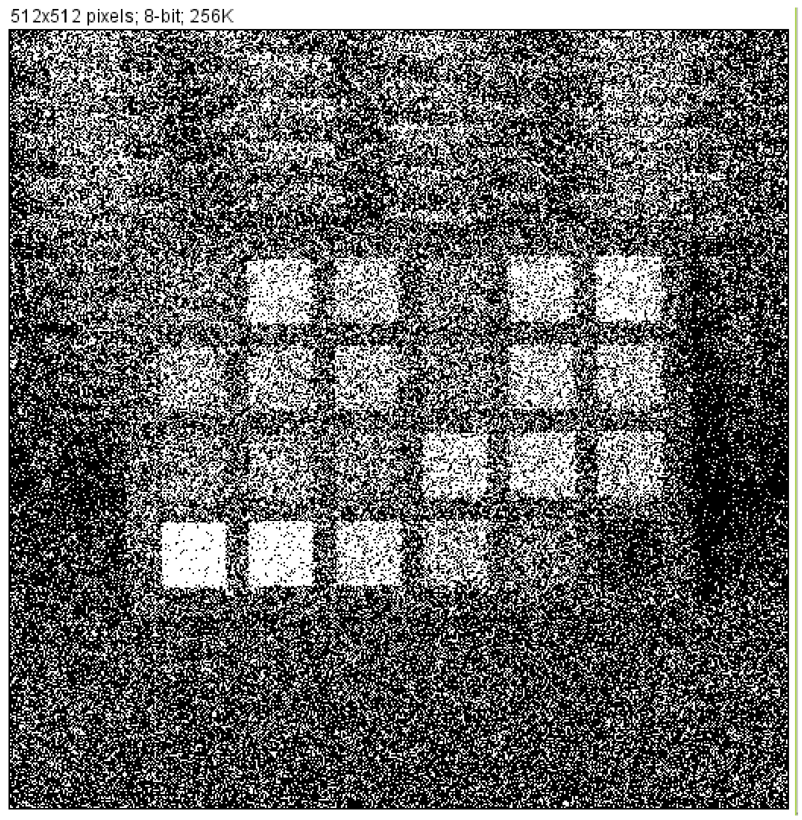


Figure 2a_2

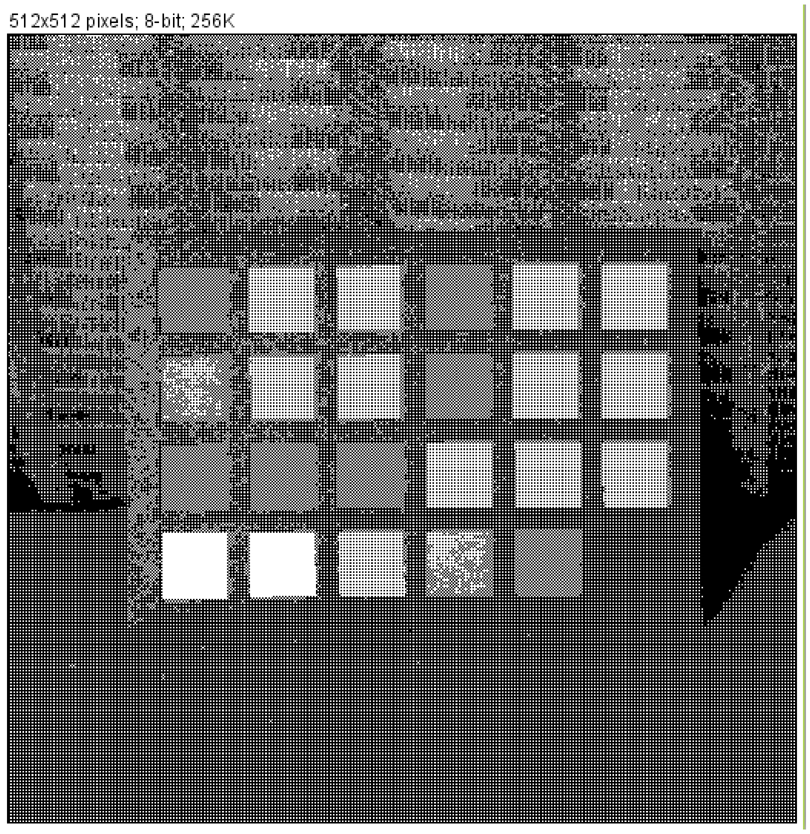


Figure 2a_3_2

512x512 pixels; 8-bit; 256K

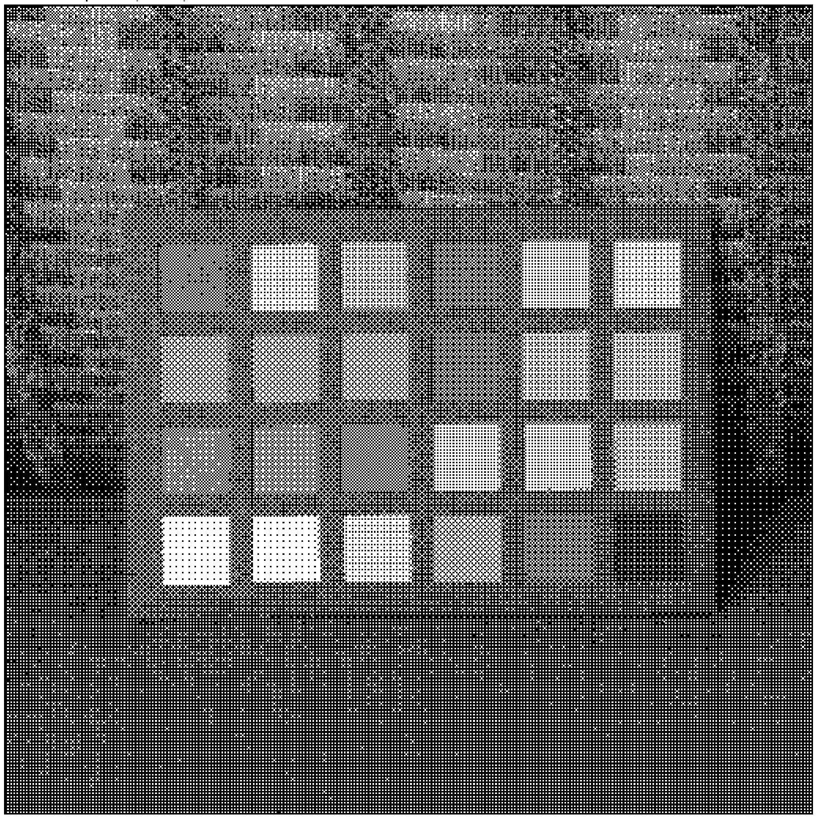


Figure 2a_3_4

512x512 pixels; 8-bit; 256K

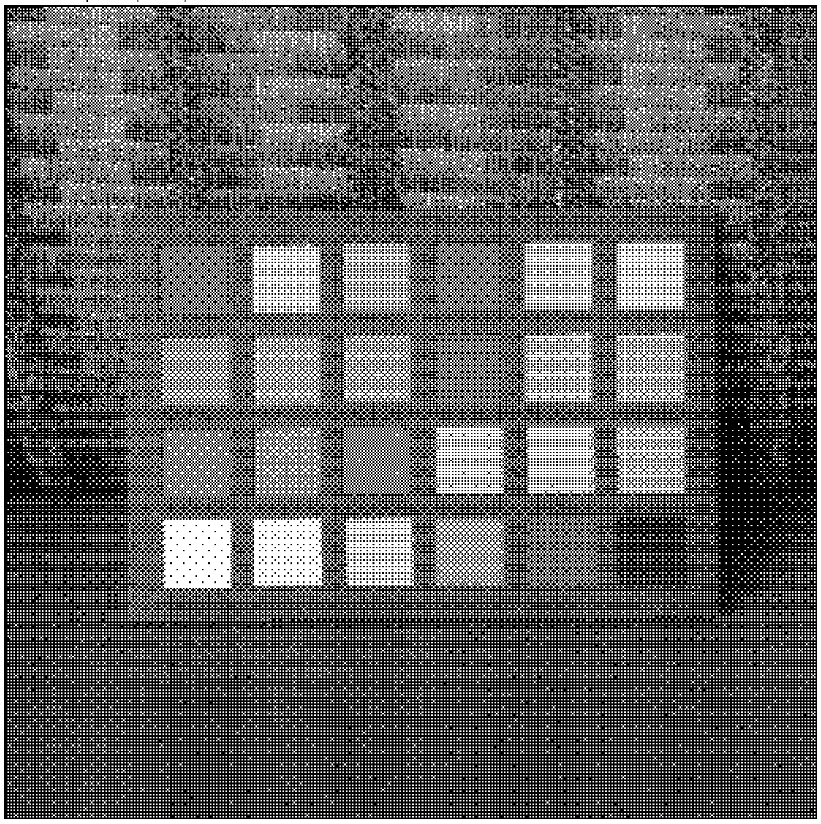


Figure 2a_3_8

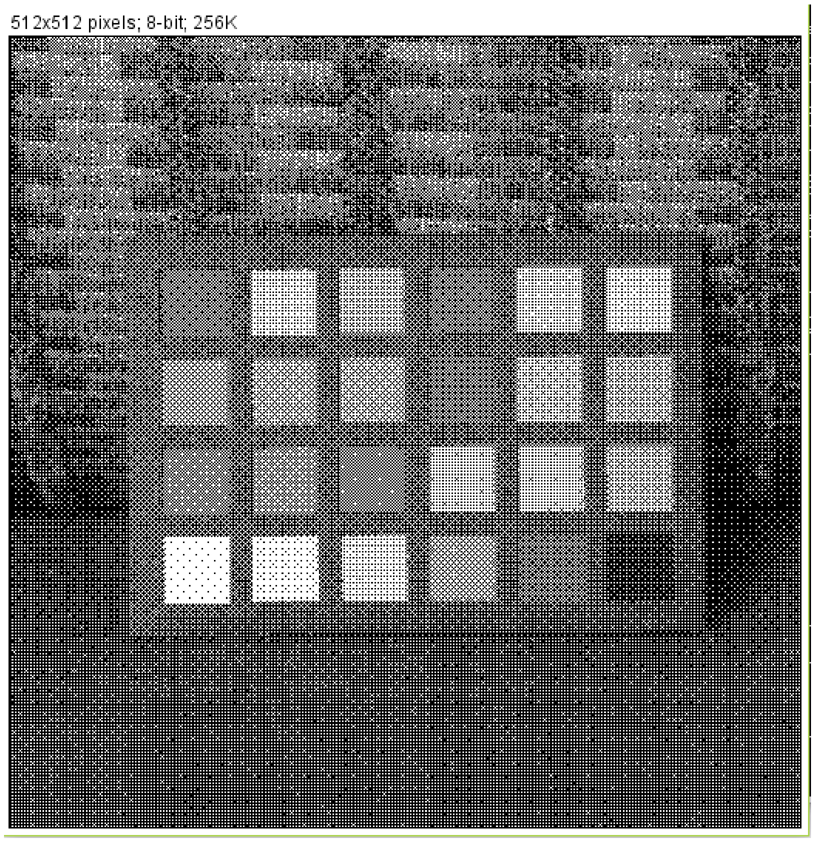


Figure 2a_3_16

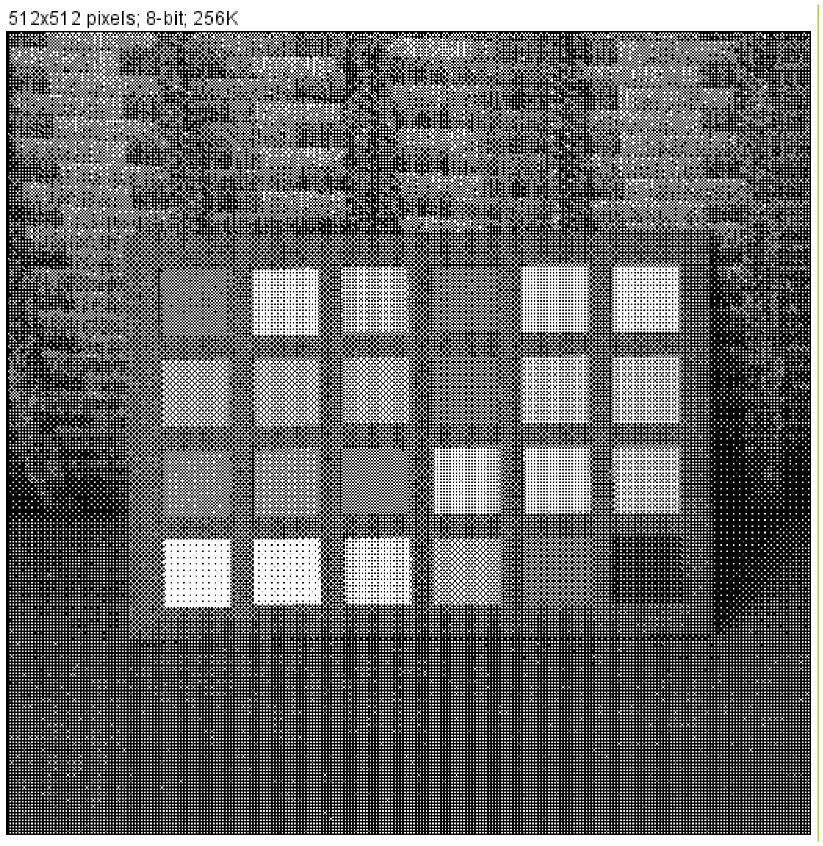
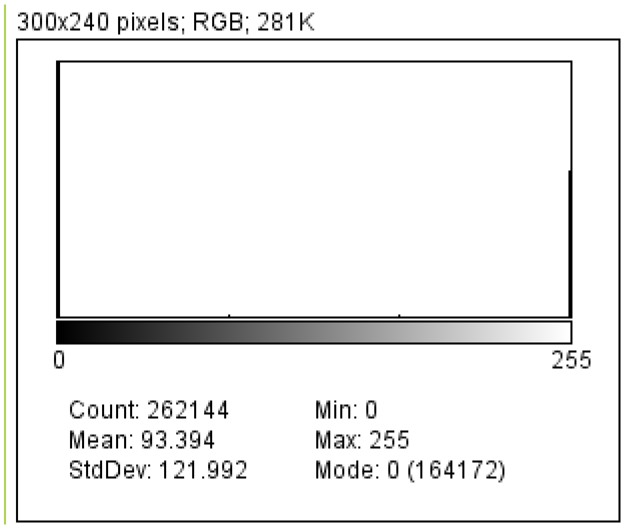


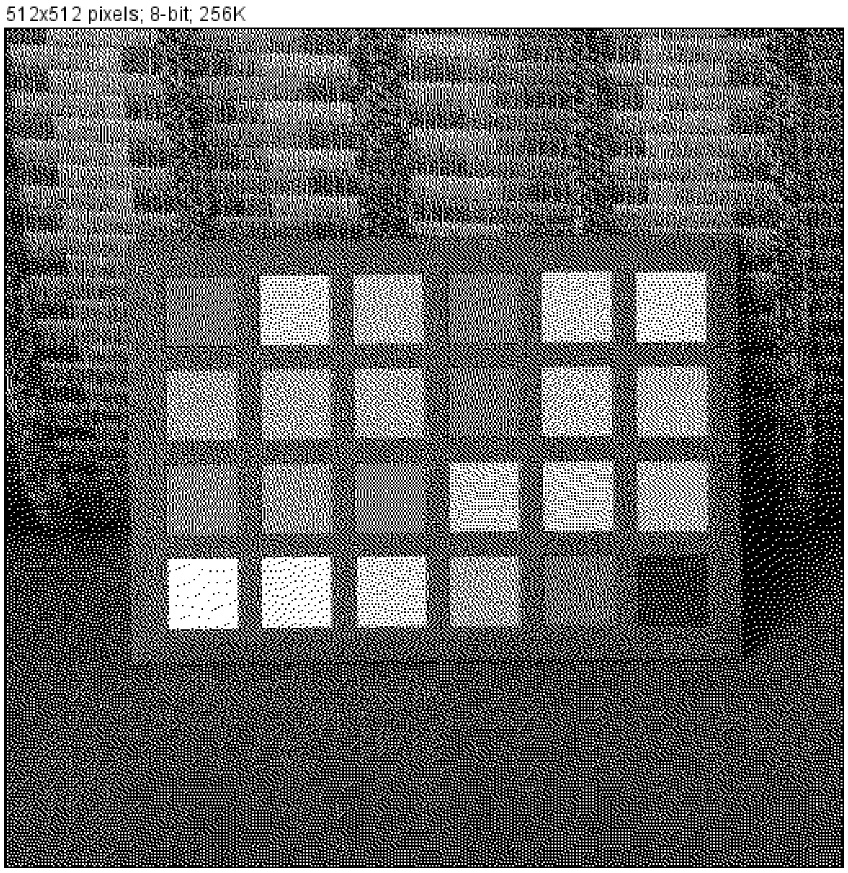
Figure 2a_3_gray



300x240 pixels; RGB; 281K

Count: 262144
Mean: 93.394
StdDev: 121.992
Min: 0
Max: 255
Mode: 0 (164172)

Histogram for gray levels



512x512 pixels; 8-bit; 256K

Figure 2b_FS

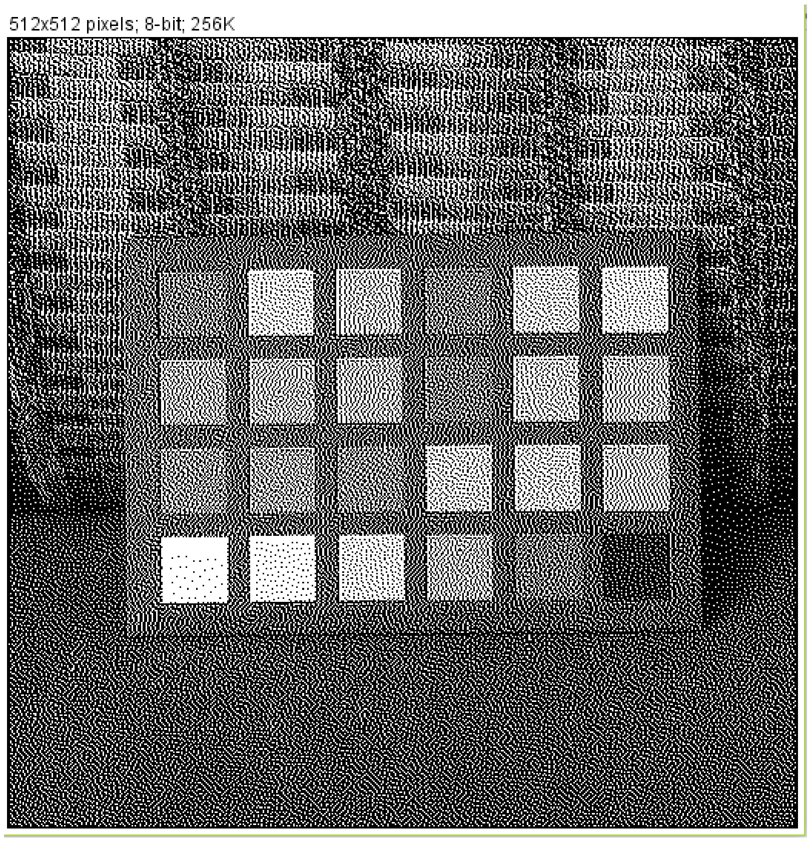


Figure 2b_JJN

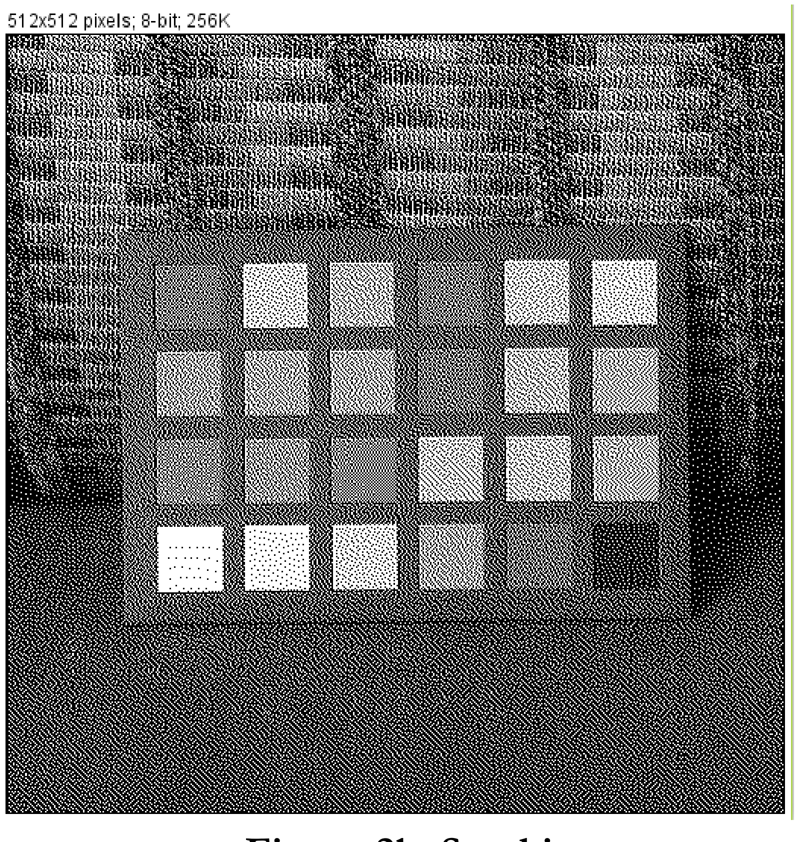
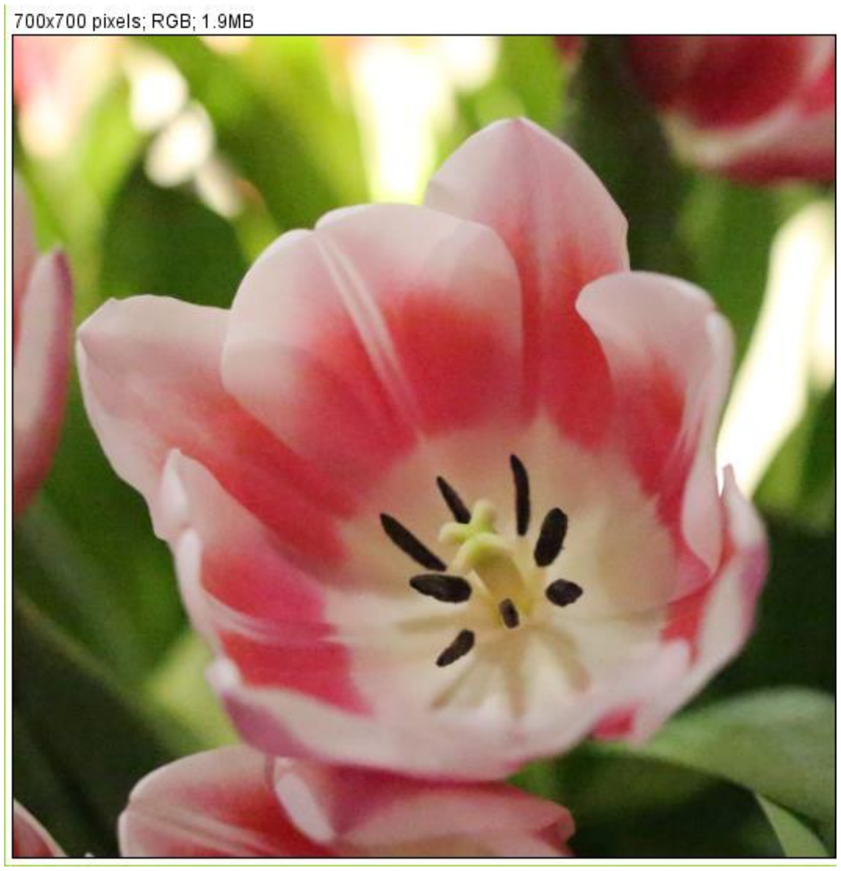


Figure 2b_Stucki



Original Flower image

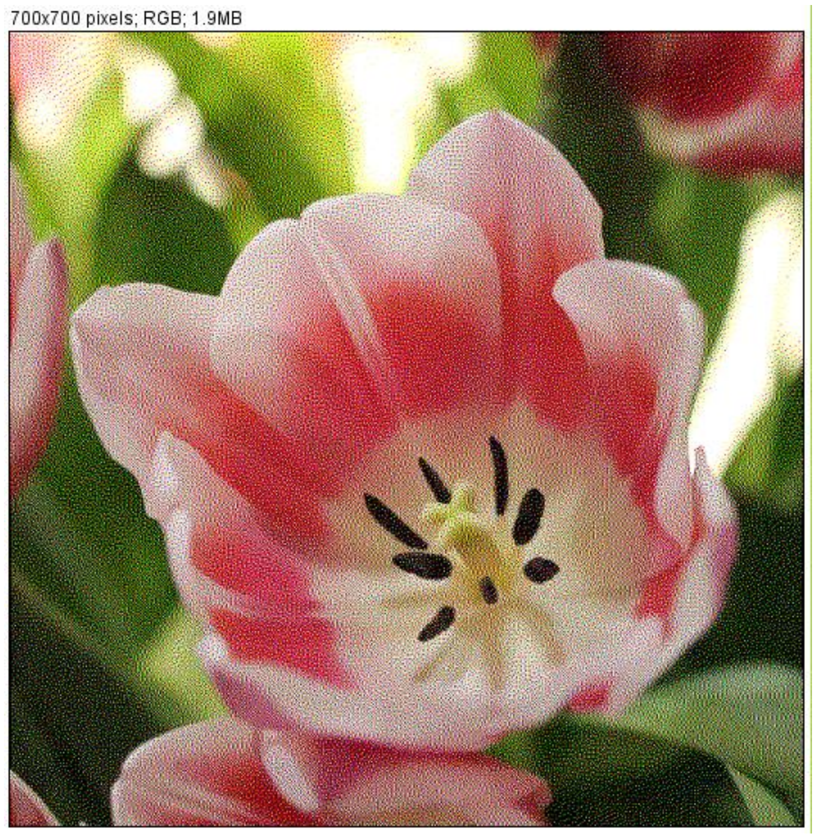


Figure 2c_1

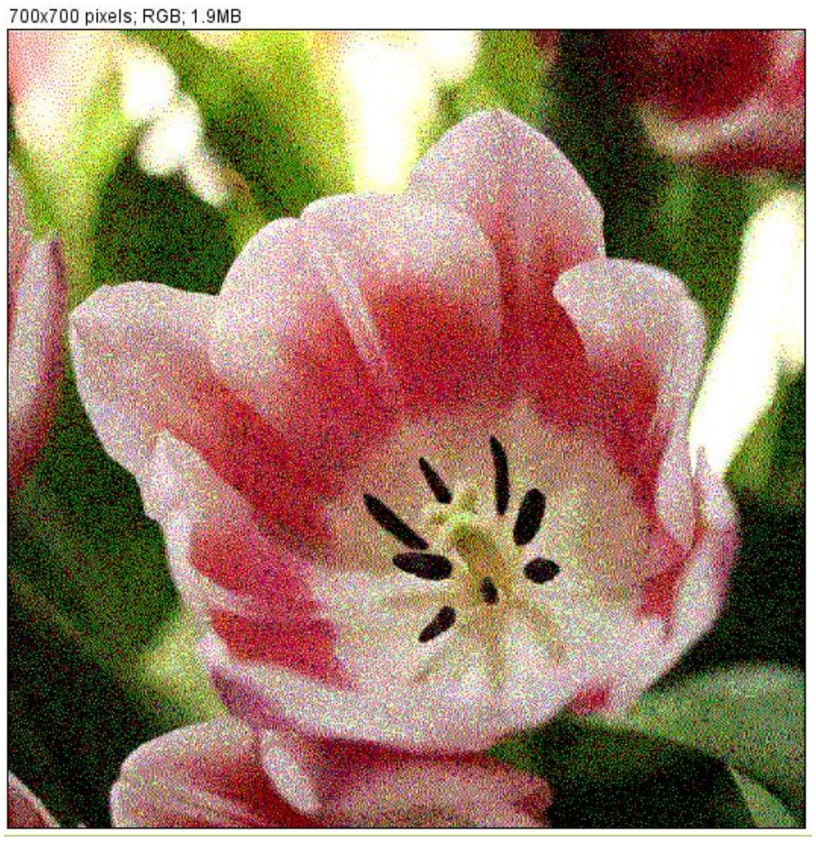


Figure 2c_2

2.4 Discussion

- **Dithering**

The Bayer Index matrices for n=4, and n=8 are as follows.

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

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

We have to slide the Bayer matrix over the image without overlapping and get the T for each pixel value. We then do quantization based on this threshold. If the value of the pixel is greater than T then set the pixel value to 0 else 255. The resulting images are shown in **Figure 2a_3_2, Figure**

2a_3_4 and Figure 2a_3_8. For different value of the Bayer matrices. We can see some black and white line like distortions. This is because Bayer index matrix has small values in each row and column, So the threshold and is set to 0. Hence we get black distortions and similarly for white.

Gray Level Dithering

The key idea is to threshold each pixel twice. I used Bayer indices 4 and 8 in this problem to do thresholding twice. For each pixel value, I use I4 to get threshold to choose where it has to lie in the (0,85) or (170,255). Then I implement I8 to get a threshold to decide which value this pixel should pick. I4 gives halftone sets and I8 gives precise halftone.

First I implement I4, then compare it with the threshold T. If it is larger than T, I assign an indicator_1=1; else 0. I again I implement I8, then compare with T and if it is larger than T, I assign indicator_2=1; else 0. Based on these two indicators I can find the value of the pixel to be allocated.

$$indicator\_1 \begin{cases} = 1 \begin{cases} indicator\_2 = 1 \rightarrow pixel\ value = 0 \\ indicator\_2 = 0 \rightarrow pixel\ value = 85 \end{cases} \\ = 0 \begin{cases} indicator\_2 = 1 \rightarrow pixel\ value = 170 \\ indicator\_2 = 0 \rightarrow pixel\ value = 255 \end{cases} \end{cases}$$

The output is a little better as some pixels are set to 85 and 170 instead of only 0 and 255. Therefore line like distortions are reduced making it better. The output is **Figure_2a_3_gray.**

- **Error Diffusion:**

The output for FS, JJN and Stucki methods are as shown in **Figure 2b_FS, Figure 2b_JJN, and Figure_2b_Stucki.** In the above figures there are slight worm like distortions in FS. But this is better in JJN and Stucki as the matrix size is smaller in FS when compared to JJN and Stucki. So as a result, the error is not greatly diffused as compared to JJN and Stucki. This is because JJN and Stucki have a 5x5 matrix. According to me JJN gives a better definition than Stucki. So I prefer JJN over all other methods as the edges are well defined and distortions are minimal.

My idea to implement a better result is taking FS method and do the serpentine scanning and instead of the standard values in the matrix such as 7,5,3,1, I want it more dynamic based on the pixel value. This would result in a bigger difference between current pixel and neighboring pixel, the more error is distributed to future pixels.

By this method we can greatly compensate the loss in image quality by averaging this based on each pixel value. Hence, I think it will lead to better results.

- **Color Halftoning with error diffusion:**

The output for this part of the problem is as shown in **Figure 2c_1** and **Figure 2c_2.**

As we can see from 2c_1, the image has a lot of artifacts and visible noise when compared to the original. The halftoned image is sharper and a bit lighter than the original. There is a short coming in this approach. The error diffusion is done separately for all the channels and this improves the quality, but when we integrate the channels, the error value for each pixel will also increase greatly. Hence there will be more artifacts when compared to the original image.

The output of MBVQ based error diffusion is as shown in 2c_2. We need to convert the RGB into CMY and then classify the pixel into one of the six quadruples based on the following inference described in the paper provided.

$$R+G\begin{cases} >255\begin{cases} G+B>255\begin{cases} R+B+G>510\rightarrow CMYW \\ R+B+G<=510\rightarrow MYGC \end{cases} \\ G+B<=255\rightarrow RGMY \end{cases} \\ <=255\begin{cases} G+B<=255\begin{cases} R+B+G<=255\rightarrow KRGB \\ R+B+G>255\rightarrow RGBM \end{cases} \\ G+B>255\rightarrow CMGB \end{cases} \end{cases}$$

Once MBVQ is found, we can find the vertex closest to the pixel value plus its error. We then subtract this pixel value from its error and distribute the result in the neighboring pixels by FS matrix. We do this over the entire image. When we compare the 2c_2 and 2c_1, we can see that there are less artifacts in 2c_2 and it has more detail with a good brightness level when compared to 2c_1.
The short coming of separable error diffusion is that it does error diffusion to each channel separately and integrates it, whereas here it is all done at the same time. These are some methods to do color half toning with good enough results.

3. Problem 3

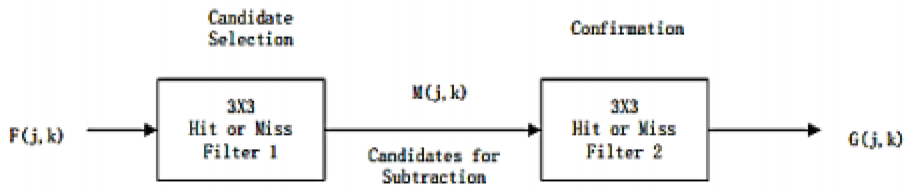
3.1 Motivation

Morphology is a kind of image processing technique that process images based on shapes. In operations that need us to identify a shape or classify a group of similar looking images, morphological image processing is widely used. The value of each pixel in the output image is based on a comparison of the pixel in the input image with its neighbors. The most primitive morphological operations are Shrinking, Thinning and Skeletonizing.
Shrinking refers to repetitive erasing of pixels in a manner that the object reduces to a single pixel. An image which has no holes will shrink to a single dot and the one which has a hole shrinks to an outer ring of pixels. We apply a 3x3 mask twice to get the shrinking effect. There are a set of predefined masks called the conditional and unconditional masks. Thinning and Skeletonizing follow suit and all of these are applied to a binary image. This is really helpful in a lot of image processing applications.

3.2 Approach

3.2.1 Shrinking

In this method, I first convert the grayscale image to a binary image by having a threshold of 127. After this I create two masks called the Conditional mask and the Unconditional masks, they are predefined and created from the given pattern tables. I used a 3x3 window to traverse the image and compare the pixel values with the different masks in two stages as described in the follows.



In the first stage, if there is a match with the pattern from the conditional masks, then the centre pixel for that window is set to 255, else 0. This is done for the entire image and the output is stored in a matrix.

The output of the first stage is passed to the input of the second stage, everything else remains same except for the masks. It is now the unconditional masks. This is where the main magic happens and shrinking takes place. If there is a 'hit' then the center pixel value of that window is set to 0 else 255. This has to be done for around 15 iterations and the image is shrunk to single pixel.

3.2.2 Thinning

Similar procedure as above was applied for the thinning process. But we also had to compliment the binary image to apply the thinning masks. We just had to replace the conditional and unconditional masks with the ones designed for Thinning. The same hit or miss concept was used again. The outputs are as shown in **Figure_3b.**

3.2.3 Skeletonizing

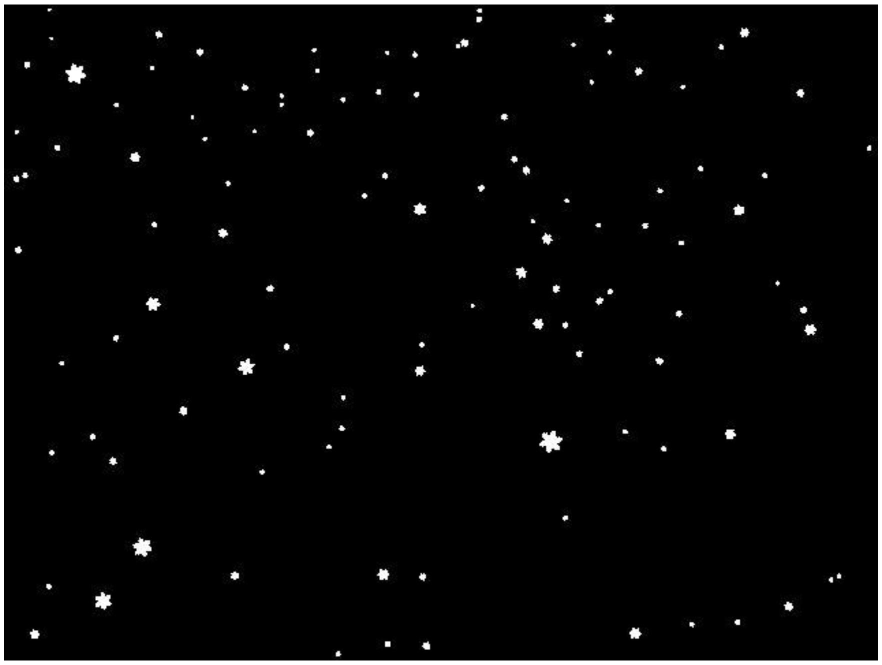
Similar procedure as above was applied for the skeletonizing process. But we also had to compliment the binary image to apply the skeletonizing masks. We just had to replace the conditional and unconditional masks with the ones designed for Skeletonizing. The same hit or miss concept was used again. The outputs are as shown in **Figure_3c**

3.2.4 Counting Game

Similar procedure as above was applied for the Counting Game as well. First converting the board image to binary and then complementing it to apply Shrinking masks. The same hit or miss concept was used again. When the Shrinking reduces the input image to just a dot, we can count the number of dots in the image and that will be the number of objects in the input image.The outputs are as shown in **Figure_3c**

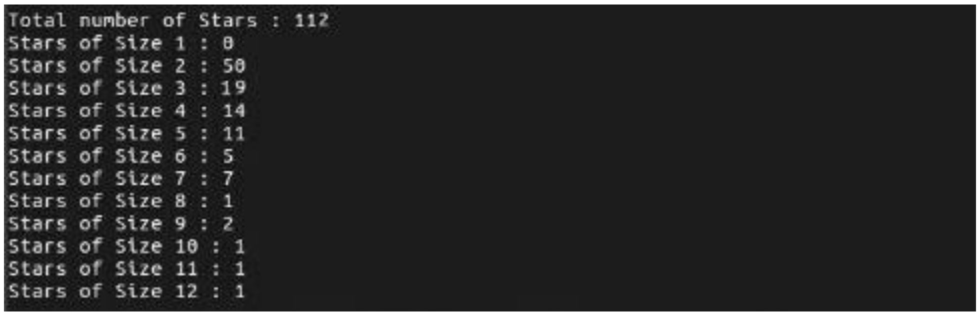
3.3 Results

- **Shrinking**



Original Stars Image

Figure_3a



```
Total number of Stars : 112
Stars of Size 1 : 0
Stars of Size 2 : 50
Stars of Size 3 : 19
Stars of Size 4 : 14
Stars of Size 5 : 11
Stars of Size 6 : 5
Stars of Size 7 : 7
Stars of Size 8 : 1
Stars of Size 9 : 2
Stars of Size 10 : 1
Stars of Size 11 : 1
Stars of Size 12 : 1
```

Figure_3a_output

- **Thinning**

Original Jigsaw 1 image

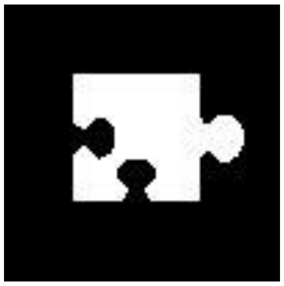


Figure 3b_complementary

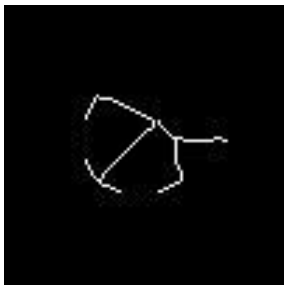


Figure_3b_thinning_comp

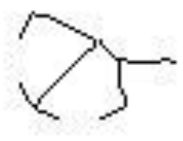
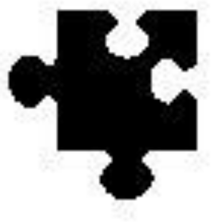


Figure_3b_output

- Skeletonizing



Original jigsaw 2 image

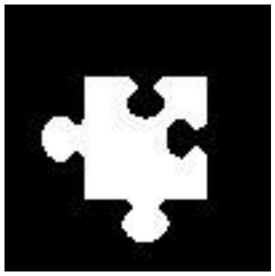


Figure_3c_complementary

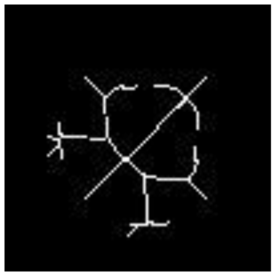


Figure_3c_ comp

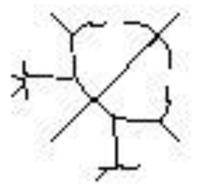
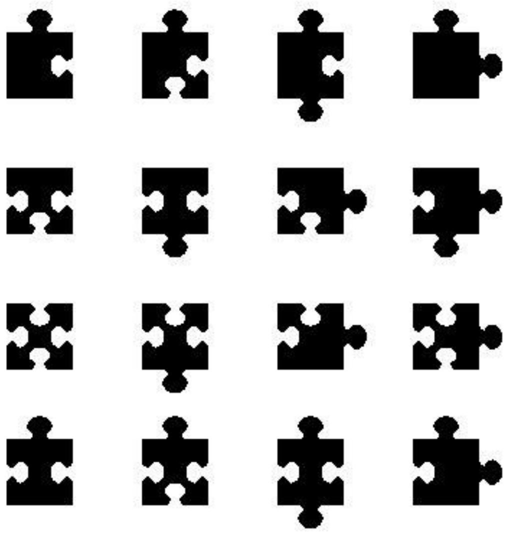


Figure_3c_output

- **Counting game**



Original Board image

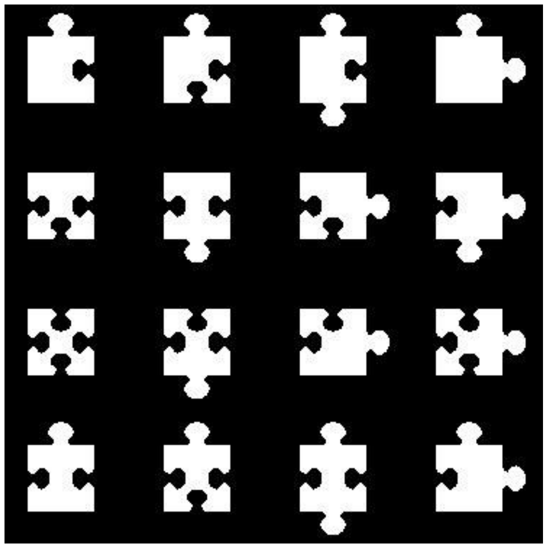


Figure 3d_Complementary

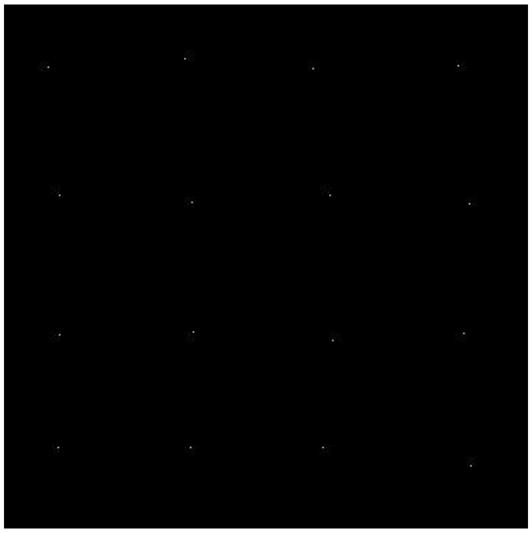


Figure 3d_output

3.4 Discussion

This homework gave good examples to make us understand how useful morphological operations are. This problem segment asked us to implement the Shrinking, Thinning and Skeletonizing operations. The two different masks which we used preserve certain characteristics in the image.

3.4.1 Shrinking

The outputs are as shown above under the results of Shrinking. **Figure_3a** gives the shrunk stars into dots image.

1) The total number of stars=112
2) There are a total of 11 Star shapes and their frequencies are as shown in **Figure_3a_output.**

3.4.2 Thinning

The thinning filter is applied to jigsaw1.raw and the output is as shown under Thinning, in **Figure_3b_output.** The various intermediate outputs for the same are also under Thinning in results tab. It is because of the nature of the masks, we see the that thinning still preserves the basic shape of the image.

3.4.3 Skeletonizing

The Skeletonizing filter is applied to jigsaw2.raw and the output is as shown under Skeletonizing, in **Figure_3c_output.** The various intermediate outputs for the same are also under Skeletonizing in results tab. Skeletonizing is done to extract a region-based shape feature representing the general form of the given object. The pixels obtained after skeletonizing, touch the boundary of the shape at at least two points. It is because of this reason, we can clearly see the connectivity of the pixels in the output image. Therefore, skeletonizing is nothing but thinning, it not only preserves shape, but also preserves the connectivity which can be clearly seen in the output image.

3.4.4 Counting Game

The output for this is as shown in **Figure 3d_output.** The first part of the problem is quite simple. I just had to run the Shrinking code with the board.raw as the input and the resulting image is the one with only dots. I just counted the number of dots obtained and that gives the number of objects in the input image.

The number of pieces in the board image=16;

I was unable to implement the second part of the problem which asks us to find the number of unique pieces in the image due to time constraint.

References:

https://arxiv.org/abs/1509.06344 **(1a)**
http://www.corrmap.com/features/homography_transformation.php **(1b)**
https://pdfs.semanticscholar.org/c67f/37a2ab36bab46bb632b65dc8dc3866f7c80e.pdf **(2c)**
https://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm **(3)**
https://www.wikipedia.org/
https://www.google.com/