EE-569 DIGITAL IMAGE PROCESSING HOMEWORK #2

Name: Amit Abhimanyu Pandey
Email: amitabhp@usc.edu
Issued Date: 3th February 2017
Due Date: 26th February 2017

Amit Abhimanyu Pandey
amitabhp@usc.edu

Problem 1: Geometric Image Modification (40%)

Problem 1: Geometric Image Modification (40%)

In this problem, you will need to apply geometric modification and spatial warping techniques to do some interesting image processing tricks. Please note that during these operations, you may need to solve some linear equations to get the matrix parameters. **ONLY** for this part, you can use some offline tools, such as Matlab or online equation solver.

(a) Geometrical Warping (Basic: 10%)

Design and implement a spatial warping technique that transforms an input square image into an output image of a diamond shape. An example is given in Figure 1.

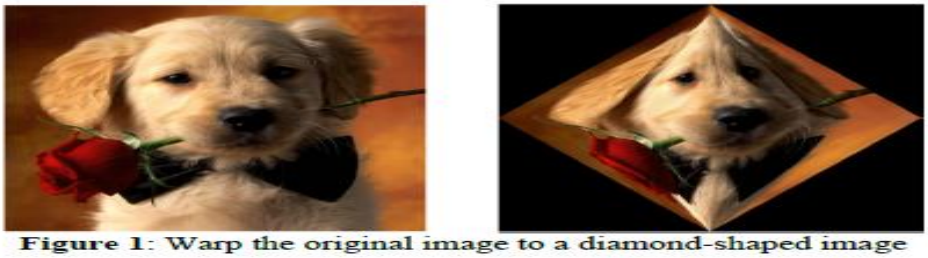


Figure 1: Warp the original image to a diamond-shaped image

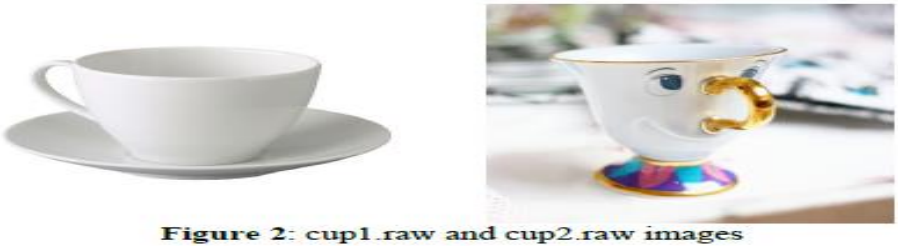


Figure 2: cup1.raw and cup2.raw images

Apply the same developed spatial warping algorithm to both 500x500 color images, *Cup1* and *Cup2* in Figure 2.

(b) Puzzle Matching (Basic: 15%)

You are given two images: *Hillary* and *Trump* with size 512x512, and two puzzle pieces in one image *Pieces*, each of which is shown in Figure 3 below. Apply geometric transformations to each puzzle piece and put them back to the correct positions. The holes in *Hillary* and *Trump* images are all of size 100x100.

Figure 3: Component images: (a) Pieces (b) Hillary (c) Trump

Write a program to implement the hole-filling algorithm. Print out the result for both images after hole-filling.

One possible way of doing this is:

1. Find the coordinates of corners in each puzzle piece image. This must be done by your program. You are NOT allowed to do this manually.
2. Design a generic geometric transform on each puzzle piece. Here you may need to combine multiple operations (e.g. translation, rotation, scaling, etc). After the transform, each of your puzzle pieces should be a square image with its sides aligned to the horizontal and vertical axes. For up-scaling, you may need to implement an interpolation function. Drop redundant pixels when down-scaling.
3. Find the coordinates of holes in *Hillary* and *Trump* images. This must be done by your program.
4. Put the transformed puzzle piece back to the original image. You should use your program to achieve this task.

Note: Bilinear interpolation could be needed to generate pixel values at fractional positions.

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

One can use the homographic transformation and the image overlap techniques to synthesize interesting images. One example is shown in Figure. 4, where the left two images are seed images and the right one is the desired output. The field image is the host image and the *Tartans* image is the embedded text image. Note that the black region outside the yellow contour of the embedded image is removed.

The homographic transformation procedure is stated below. Images of points in a plane, from two different camera viewpoints, under perspective projection (pin hole camera models) are related by a homography:

$$P_2 = HP_1$$

where H is a 3x3 homographic transformation matrix, P_1 and P_2 denote the corresponding image points in homogeneous coordinates before and after the transform, respectively. Specifically, we have

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

To estimate matrix H, you can proceed with the following steps:

- Fix $H_{33} = 1$ so that there are only 8 parameters to be determined.
- Select four point pairs in two images to build eight linear equations.
- Solve the equations to get the 8 parameters of matrix H.
- After you determine matrix H, you can project all points from one image to another by following the backward mapping procedure and applying the interpolation technique.

Implement above homographic transformation steps to generate the desired result as shown in Figure 4.

Now, let us keep the same host image and the same overlay region, apply the algorithm to the embedded text image *Trojans* in Figure 5 to generate a new image overlay. Show the result and make discussion on the performance.

Figure 4: An example of homographic transformation and image overlay

Figure 5: The Trojans text image.

1.a) Image Warping

1.a.1) Motivation

Image warping is a process of digitally changing or manipulating the given image in to some shape specified. Basically the new image formed after manipulating is created without the loss of the color of the original images. We just need to find which coordinates are getting mapped to which coordinates in the output image, and then we need to project value of that coordinate at the output image which is known as forward mapping, or we can also do the reverse mapping. Warping may be used for creative purposes as well when extended to do morphing. Pure warping means that the points are mapped to the points without changing the colors. Warping effect in the real world can sometimes be seen with the help of the distorting mirrors or carnival mirrors. Pretty much we, can say if we want to achieve this effect, or any other such effects on digital images, it's called Digital Image Warping.

Figure 1: Distorting Mirror reflecting warped image

1.a.2) Approach

So the problem statement was to warp two following given images as diamond shaped. There are few ways to perform the warping. I chose Linear Warping for solving the above problem statement.

Figure 2 Input Images for Warping

Figure 3:Dog Image Warped to Diamond Shape

So our end goal with both the cup images is to achieve the same effect as that shown in the above figure for the Dog's Image. So my algorithm is specific to this particular shape only.

Algorithm for Linear Warping

- Firstly, create a new image array of the same size, this would be our output image, as we are not changing the size of the image, we are just projecting it differently.
- We need identify four control points, control points are the points whose values are not changing. They help us in many ways, along with helping in providing specific shape to the image.
- When we know these control points, we ultimately know that our new image is going to mapped inside the region formed after connecting all four control points. In our case we had four control points (0,249), (249,0), (249,499) and (499,249). Also while doing all this I worked in the Cartesian coordinates back and forth. As for looping we need to work in the image coordinate.
- As the shape is diamond, and we can form a diamond by connecting four control points, with a line. So it is important to find the slope of those four lines.
- After finding the slope of those four lines we now just need to find the intensity of each pixel in the new image.
- So if the pixel lies inside the diamond i.e. within the boundary region or on the boundary will only get the intensity from the input image, otherwise it will be assigned black color.
- Now suppose if the coordinate of the pixel is inside the diamond region then we just need to do find the number of the pixels in that row who are well within the boundary region, to get the resize number which is used to calculate the factor for nearest neighbor interpolation or bilinear interpolation (reverse mapping).

$$factor = \frac{Resize}{original\ size}$$

Where Resize=number of pixels in the current row of the output image within the boundary.
Original Size= number of pixel in the current row of the input image.

- So we need to calculate the factor of each row as number of pixels in the output image that will get pixel intensities are changing with each row.

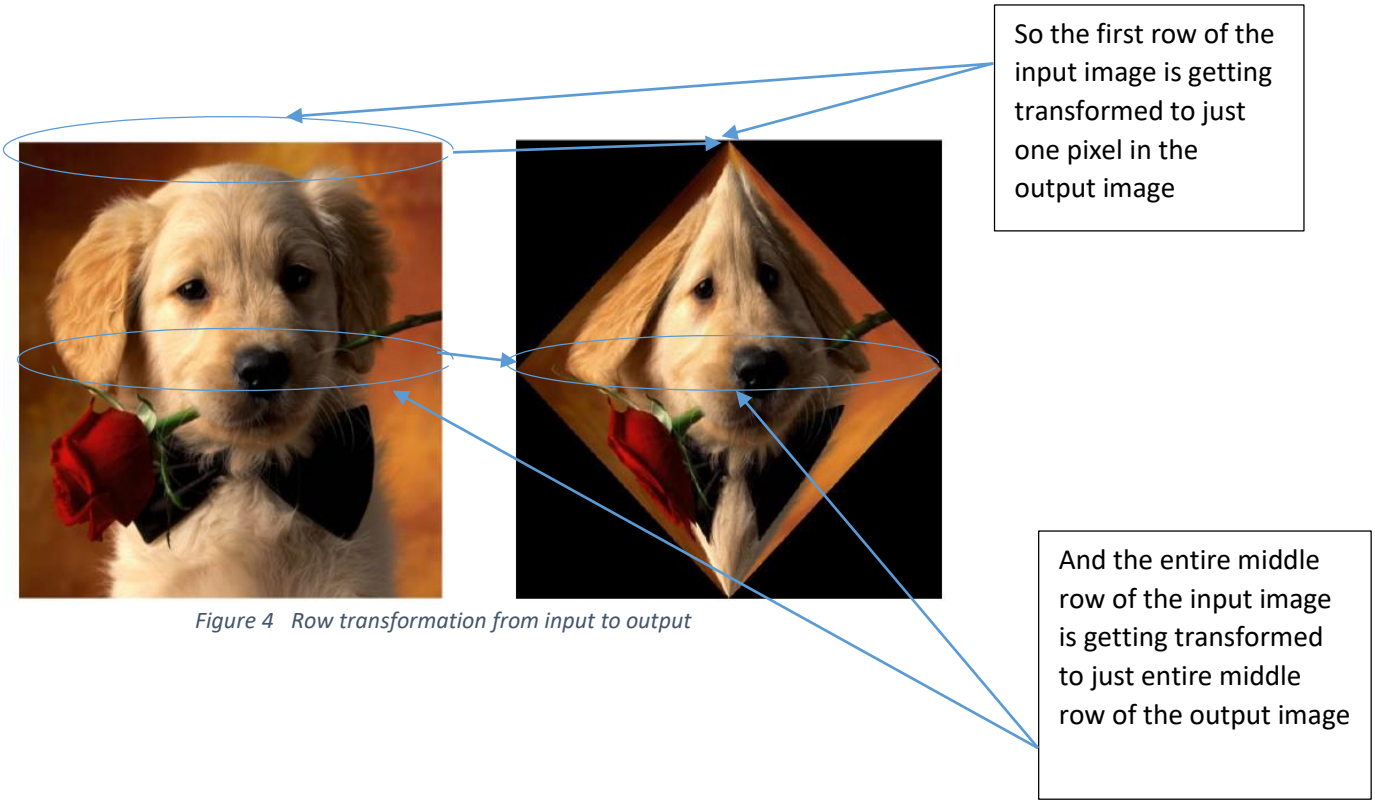


Figure 4 Row transformation from input to output

- So this what was required to be done, so keeping above facts and information in mind, I performed the Linear Warping.

1.3) Results

Figure 5: Before Warping Cup1

Figure 6: After Warping Cup 1

Figure 7: Before Warping Cup 2

Figure 8: After Warping Cup 2

1.a.4) Discussion

The main challenge while implementing warping, was to decide upon the method by which I should perform the warping. In class Professor had described warping with the help of triangle and control points. But implementing this method was very tedious. And even after implementing it I didn't get the desired results. The problem was with my understanding of use of triangles to implement warping. But eventually after quite deliberation and research I found that linear warping would be the best way to achieve the desired result.

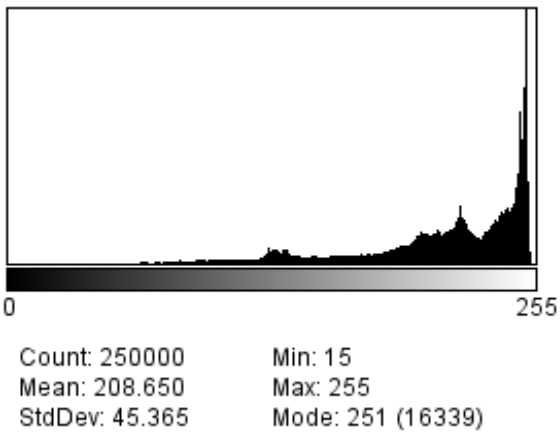


Figure 9: Histogram of Cup 2 before Warping

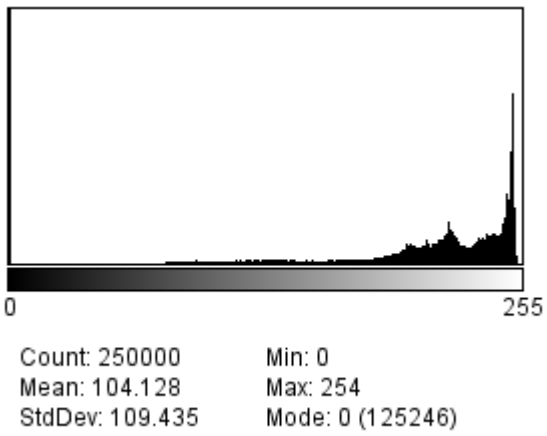


Figure 10: Histogram of Cup 2 After Warping

As it was claimed in the start that linear warping is just mapping of points to another points without changing the color. Which can be commented after viewing the histogram of both images. The structure of the histogram is not changed from input to output majorly. It is just that lot of pixels have been dropped in between, due to which the frequency of the intensity values has decreased, along with increment of frequency of black intensity level.

1.b) Puzzle Matching

1.b.1) Motivation

The whole purpose of the first question is to make us familiar with the geometrical modification involved in image processing. Geometrical modification with the images become intuitively easily to understand when we work in the Cartesian coordinate system. Conversion in the Cartesian coordinate system help when dealing angles, lines and slopes i.e. basically geometrical modification. Because it is very easy to calculate slopes and angles in Cartesian coordinate system, as that what we have been doing since high school. Now in this section of the first question, the problem statement asks to perform puzzle matching on digital images. The main motivation behind giving us this question was to motivate us to use various geometrical image processing tools like translation, rotation, scaling etc.

1.b.2) Approach

So we were given following three images to perform puzzle matching, it involved use of translation, rotation and scaling matrices in order to complete the whole puzzle matching.

Figure 11: Missing Pieces

Figure 12: Hillary missing eyes

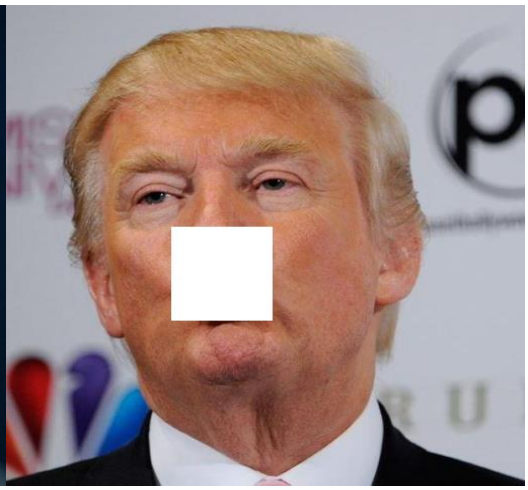


Figure 13: Trump Missing Mouth

So above fig 11 consists the two missing pieces of the Hillary and Trump images. Thus we need to extract the missing pieces and put them at the perfect positions in the Hillary and Trump images.

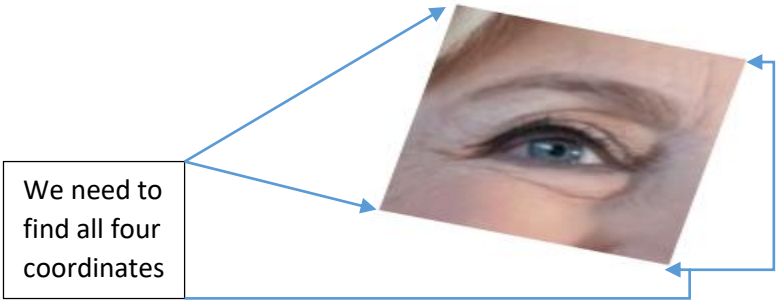
Algorithm for Puzzle Matching:

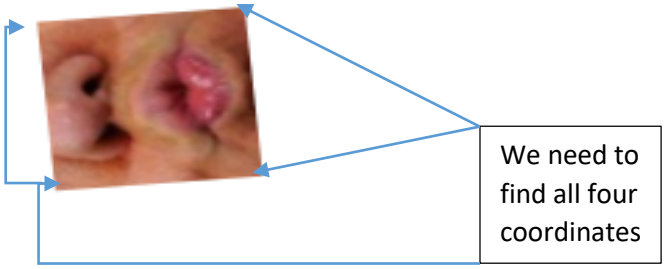
- Firstly, it was imperative to extract the puzzle pieces from the whole piece image, for placing them in the Hillary and trump's images appropriately
- But main problem was that these pieces didn't not have correct orientation i.e. they can't be directly placed in the main images. They need to straightened up, translated and scaled appropriately before we can place them in the main image.
- So what I thought would ease the process for me was to firstly separate two pieces from each other, and then perform manipulations on them accordingly. Separating them out really helped me programming.

Figure 14: Extracted Hillary's Piece

Figure 15: Extracted Trump's Piece

- I figured out that both the pieces are pretty much demarcated from each other and the extraction wouldn't cause me miss any pixel of the either of the pieces from the entire piece image.
- After doing that, next step was to correct the orientation of the extracted pieces. They both can be seen to tilted in some direction. Thus solution to straighten up the pieces was to perform rotation.
- But before performing rotation we had to find the coordinates of the extracted pieces:

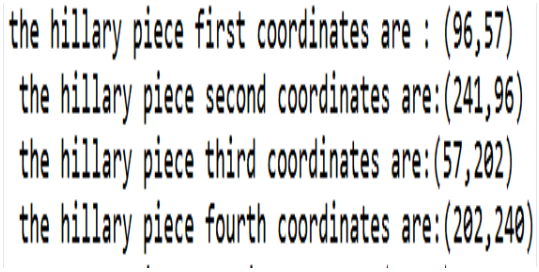




- Before finding the coordinates we need to convert the image coordinates to Cartesian coordinates and the formula for it is as follows:

$$y = P - p - 0.5$$
$$x = q + 0.5$$

$where\ x, y\ are\ cartesian\ coordinates\ of\ q, p\ image\ coordinates$
$and\ P = height\ of\ the\ image$



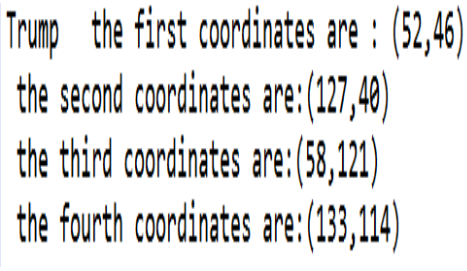


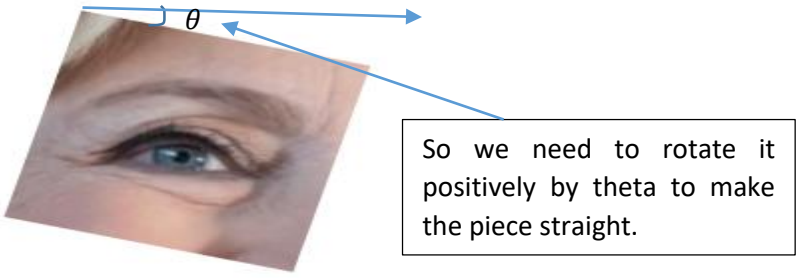
Figure16: Image Coordinates of the Hillary Piece *Figure17: Image Coordinates of the Trumps Piece*

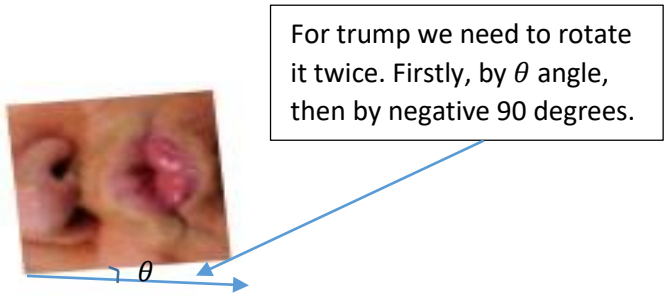
Above figure shows the image coordinates of the Hillary and trump's pieces from the extracted pieces.

- So now we know what are the coordinate locations of the tilted pieces. Through this information we can calculate the slope of the top side of the tilted piece, which we can use to find out θ.

$$\theta = \tan^{-1}(slope)$$

Where, $\theta = angle\ of\ rotation$





- The coordinates of the tilted pieces help us to calculate the θ, which we can substitute in the rotation matrix to help us rotate the image.
- Rotation Algorithm is pretty straight forward, once we find the angle of rotation. I directly used the reverse mapping technique to find the intensity values at each output coordinates via manipulating the rotation matrix formula.

$$\begin{bmatrix} j \\ i \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} * \begin{bmatrix} q \\ p \end{bmatrix}$$

Rotation Matrix

- By directly incorporating the reverse mapping technique, I took the inverse of the rotation matrix, and found which image coordinates of the output image requires intensity values from which coordinates in the input image.

Figure 16: Straight (After rotation Hillary's piece) *Figure 17: Straight (After rotation Trump's piece)*

- So as soon as I completed the task of straightening up the pieces is done, we need to scale to the exact size of the white patch seen in the main image to fit them over there.
- For that we need to find the size of the white patch in the main image. For which we again need to find out their coordinates which would help us to place the scaled up images to exact locations along with providing us the information of the height and the width of the white patch.

```
the first coordinates are : (163,236)
the second coordinates are:(262,236)
the third coordinates are:(163,335)
the fourth coordinates are:(262,335)
```

```
the first coordinates are : (173,135)
the second coordinates are:(272,135)
the third coordinates are:(173,234)
the fourth coordinates are:(272,234)
```

Coordinates of the Hillary and Trump white patch respectively

- Now as we know the height and width of the white patches of the main images, we scale the straight images to their size and with the help of the coordinates found we can replace the white pixels with the pixels of the respective pieces, to complete the puzzle.

1.b.3) Results

Figure 18: Puzzle Matched Hillary (First Try)

Figure 19: Puzzle Matched Trump (First Try)

Figure 20: Puzzle Matched Hillary (Second Try)

Figure 21: Puzzle Matched Trump (Second Try)

1.b.4) Discussion

It can be seen from the above fig 18 and 19, that the pieces were placed at the right position, but there were white borders around the pieces placed in the final image. This was because the piece image that was given to us, was having diluted white edges around boundary as shown below. Thus after all geometrical manipulations, that mixed white boundaries were still present, which I eventually placed in the final image.

Thus to eliminate the above problem as TA, suggest I scaled the piece images 4 lines bigger than that was required and eventually placed only the inside 100*100 portion of the piece image on the final image. Thus eliminated the white boundary.

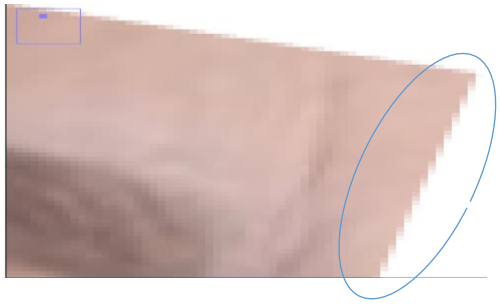


Figure 22: Mixed white Edges at the boundary of the Pieces

Also the trumps piece image had to rotated twice, firstly I had to correct the small tilt theta angle, and after that I had to rotate by 270 (positive) degrees/ 90(negative) degrees, correctly straighten it up.

***I have performed bilinear interpolation at various places instead of nearest neighbor interpolation. Due to which my piece images look a bit different than the original piece images in terms of color intensity.**

1.c) Homographic Transformation and Image Overlay (Advanced: 15%)

1.c.1) Motivation

The main motivation behind doing homographic transformation is to provide a relation between two camera projections. The way one camera sees the world can be related with another camera looking at the same thing from different view point. It is used to project a 2d plane on to different 2d plane with help of Homography matrix (assuming pin hole camera model). Homography matrix is estimated firstly with help of 8 points, and then it is used to project one 2d plane on to another. The homographic transformation procedure is stated below. Images of points in a plane, from two different camera viewpoints, under perspective projection (pin hole camera models) are related by a Homography:

$$P_2 = HP_1$$

where H is a 3x3 homographic transformation matrix, P_2 and P_1 denote the corresponding image points in homogeneous coordinates before and after the transform, respectively.

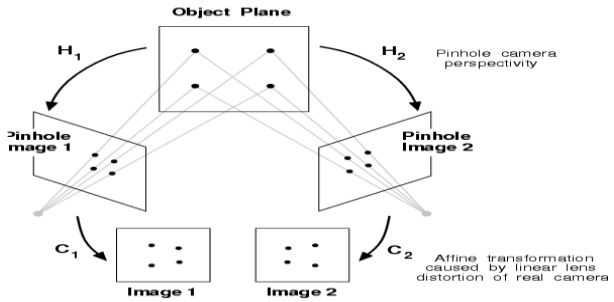


Figure 23: Use of Homography technique

1.c.2) Approach

So the problem statement motivated us to project the following Trojan SC image on two the field image. Example was given of the Tartans image projected on the field:

Figure 24: Example Tartans Logo

Figure 25: Field Image on which the projection is to made

Figure 26: Example Tartans project on the field.

Figure 27: Trojan SC needs to be projected on the field.

Algorithm for performing Homography to overlay/project images:

- The whole relation between the input and the output image (reverse mapping) is given as follows:

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

Where: $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_2'/w_2' \\ y_2'/w_2' \end{bmatrix}$

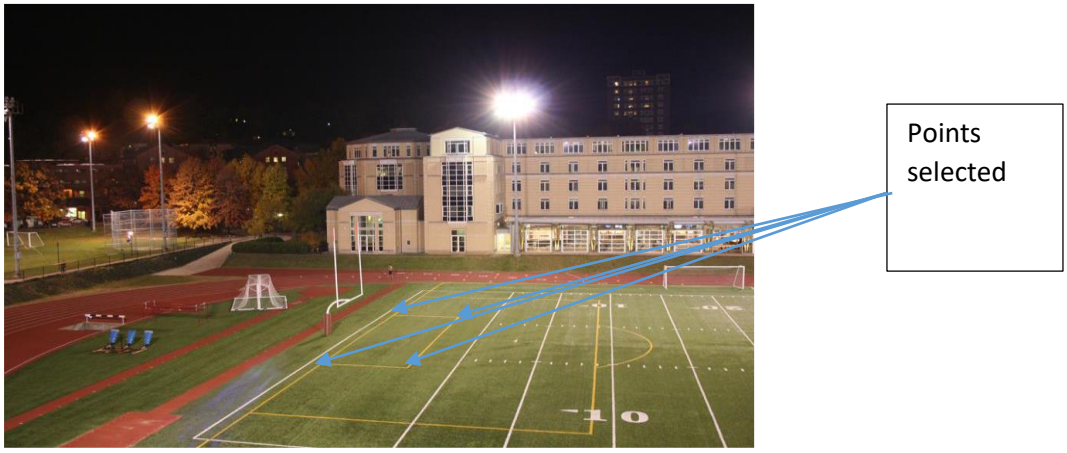
Where $x_2 and\ y_2\ are\ the\ image\ coordinates\ of\ the\ Trojan\ SC\ image$

$x_1 and\ y_1\ are\ the\ image\ coordinates\ of\ the\ Field\ image$

$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix}$: Homography Matrix

- Homogenous coordinate system is followed, for helping solving the matrices above.
- Now as we are familiar with the equations, we now need to calculate the values of the H matrix. For easy of calculation we set $H_{33} = 1$

- Now we for calculating 8 unknown values of the H matrix we will use 4 points from the Trojan SC image and 4 points from the field image.
- But these four points will not be random: The points should be selected keeping in mind that, the points selected from the TrojanSC images, will form a boundary when connected such that only pixels in that boundary or on the boundary will be projected. Similarly, for the field image the 4 points selected will provide 4 coordinates, which will form the boundary of the region in which the TrojanSC image will be projected. I selected following points:
- For Trojan SC I selected all the end points, as I want to project entire image on the field. For field I selected end points of the rectangle in the D area near the goal post:



- After selection of the 8 coordinates we can calculate the Homography matrix as follows:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix}$$

Figure 28: Example Final Matrix to be solved for H matrix values (Collins, Planar Homographies)

- After estimating the Homography matrix, we just need to reverse mapping, and calculate the coordinates of the Trojan SC image that we need to project on the field.

- So the points on the blue lines or within the region formed after connecting the blue lines, will only be given for calculation, no other points or pixel will be manipulated.
- For achieving this the coordinates initially supplied for Homography matrix, will be used to calculate the 4 line equations. And with their help we will check whether the given coordinate lies within the rectangle or not.

1.c.3) Results

Figure 29: Final Field Image with Trojan SC Projected

1.c.4) Discussion

The main crux while implementing the above algorithm was to work in the Cartesian coordinate system. Working in Cartesian coordinate eased out the process of calculation of the line equations. Eventually we had to convert back to the image coordinate before accessing the image array.

Figure 29: Final Field Image with Trojan SC Projected with white Background

The only problem that, I faced was the whole trojanSC image getting projected on the field, which was not desired. So eliminate the whole white Background, I used a range [255, 240], if R, G, B are having intensities in within this range don't project it.

Problem 2: Digital Half toning

There are 256 gray levels for pixels in Figure 6. Please implement the following procedures (dithering matrices and error diffusion) to convert *Man* image to a binary image. In the following discussion, $F(i,j)$ and $G(i,j)$ denote the pixel of the input and the output images at position (i, j) , respectively. **Compare the results obtained by different algorithms in your report.**

Figure 6: Man.raw

(a) Dithering Matrix (Basic: 15%)

Convert the 8-bit *Man* image in Figure 6 to a half-toned image using the dithering method. Dithering parameters are specified by an index matrix. The values in an index matrix indicate how likely a dot will be turned on. For example, an index matrix is given by

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

where 0 indicates the pixel most likely to be turned on, and 3 is the least likely one. This index matrix is a special case of a family of dithering matrices first introduced by Bayer.

The Bayer index matrices are defined recursively using the formula:

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

The index matrix can then be transformed into a threshold matrix T for an input gray-level image with normalized pixel values (*i.e.* with its dynamic range between 0 and 255) by the following formula:

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

where N^2 denotes the number of pixels in the matrix. Since the image is usually much larger than the threshold matrix, the matrix is repeated periodically across the full image. This is done by using the following formula:

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

where $F(I,j)$ and $G(I, j)$ are the normalized input and output images.
There are different ways to construct a dithering matrix. Another example for a 4x4 matrix is:

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

Answer the following questions.

1. Construct $I_2(i,j)$ and $I_8(i,j)$ Bayer index matrices and apply them to the *Man* image.
2. Compare the above $A_4(i,j)$ matrix with $I_4(i,j)$ Bayer Matrix and discuss their differences. Apply both and compare the results.
3. If a screen can only display FOUR intensity levels, design a method to generate a display-ready *House* image. Show your best result in gray-scale with four gray-levels (0, 85, 170, 255) and explain your design idea and detailed algorithm.

(b) Error Diffusion (Basic: 15%)

Convert the 8-bit *Man* image to a half-toned one using the error diffusion method. Show the outputs of the following three variations, and discuss these obtained results. Compare these results with dithering matrix. Which method do you prefer? Why?

1. Floyd-Steinberg's error diffusion with the serpentine scanning, where the error diffusion matrix is:

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

2. Error diffusion proposed by Jarvis, Judice, and Ninke (JJN), where the error diffusion matrix is:

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

3. Error diffusion proposed by Stucki, where the error diffusion matrix is:

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

Describe your own idea to get better results. There is no need to implement it if you do not have time. However, please explain why your proposed method will lead to better results.

2.1) Motivation

Half toning is technique to produce continuous tone imagery through the use of dots. Printing uses this technology, where the dot firing is used which is actually modulated via various techniques. It came in to evolution because of the fact that our eyes average out effect the color or shades spatially. So different shades of gray can be produced with the change in the density of the dots per inch. This can be extended to color printing as well. Where CMYK are used to produce different colors, concept of angles also comes in picture there. Continuous tone imagery contains many tones of gray colors; via half toning we can create an effect of shades of gray with just one color with differing size of dots. This heavily relies on the fact that tiny halftone dots blend in to smooth tones by our eye. But if observed at microscopic level we can observe that the printing is not done using multiple colors, instead it is done with help of just one color in case of black and white printing. This certainly helps in reducing the cost of the ink cartridge. The high dot density stands for darker impression, while the low dot density stands for brighter impression.

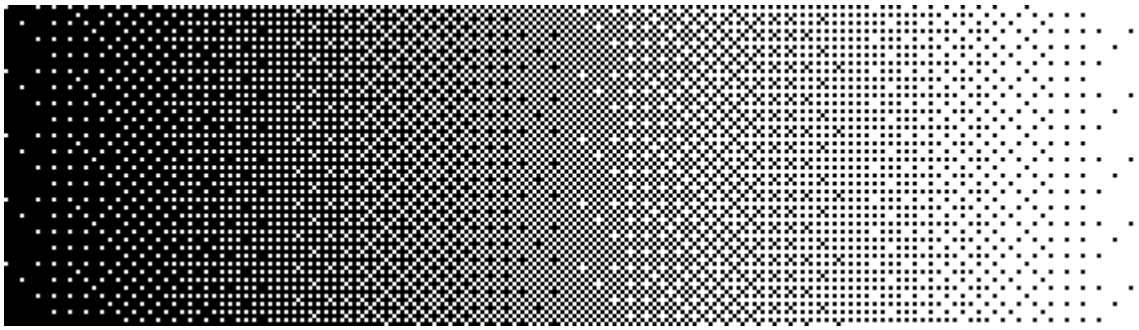


Figure 30: Bayer's Dithering used for Half toning

Figure 31: Input Image

2.2) Approach

2.a) Dithering Matrix (Basic: 15%)

Dithering is process of creating illusions of the colors which are not present, this is possible with the help of random arrangement of pixels. Dithering is generally used to improve the thresholding. We generally do the thresholding using a constant value, which results in losing the fine details of the image. So basically dithering randomizes the thresholding, due to which we get a better contrast image. This is done via preparing new thresholding matrix of the size and values in accordance to the various matrices. Commonly used is Bayer's index matrices.
So following steps needs to followed while implementing dithering method for half toning:

- Firstly, we need to choose an initial Bayer's Index Matrix: I_2, I_4, I_8 etc. These matrices can be computed with help of initial matrix I_2, which is given as below:

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

- Then the I_2 can be substituted in the following formula to recursively compute the rest of the matrices:

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

Eg:

$$I_4 = \begin{bmatrix} 4 * I_2(x,y) + 1 & 4 * I_2(x,y) + 2 \\ 4 * I_2(x,y) + 3 & 4 * I_2(x,y) \end{bmatrix}$$

- Then the index matrix can then be transformed in to threshold matrix by using following formula:

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

where N^2 denotes the number of pixels in the matrix. N=2,4,8.....

- Since the image matrix is much larger than the threshold matrix, the matrix is repeated periodically across the full image matrix. And is this done via using following formula:

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

Where F(i,j) and G(i,j) denote the pixel of the input and output images at position (i,j).

2.b.) Error Diffusion (Basic: 15%)

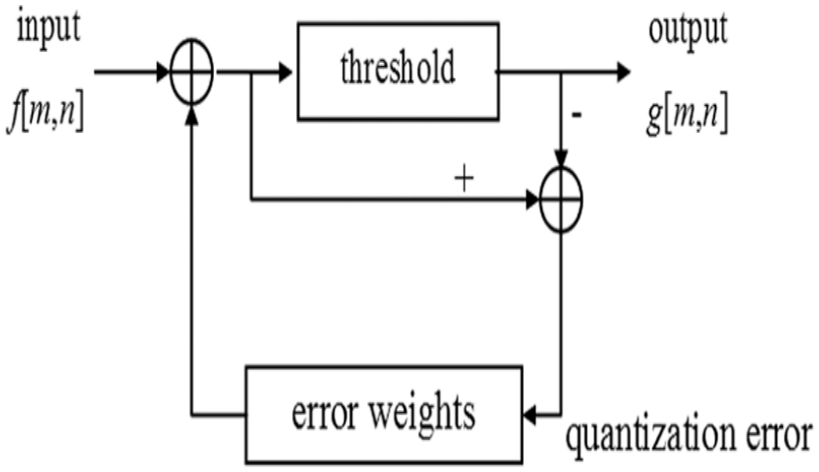


Figure 32: Error Diffusion Block Diagram [3]

Error Diffusion is referred as area operation technique for dithering, as it not only affects the pixel getting processed, but also it affects the neighbors of the pixel being processed. Error diffusion is a type of half toning in which the quantization residual is distributed to neighboring pixels that have not yet been processed[1]. It generally enhances the edges of the images.

- As indicated in the block diagram, that firstly we need to threshold our input image pixel. I used a threshold value of 127. It the image pixel intensity is above the threshold value then new pixel is set to 255 otherwise set to 0.
- After which the quantization error is calculated via subtracting the input image pixel value and new pixel value.
- This error is then multiplied with the various error diffusion matrices and the result is copied back on the input image.
- Interesting thing to not down is the positions where the results after multiplication of the error and diffusion matrices goes. The results are added to the neighboring pixels of the pixel that was processed.
- For example, for Floyd–Steinberg the error will be diffused to the pixels marked by blue ink of the original input image. If we consider x(i,j) to be the pixel being processed of the input image.

$$\begin{bmatrix} x(i-1,j-1) & x(i-1,j) & x(i-1,j+1) \\ x(i,j-1) & x(i,j) & x(i,j+1) \\ x(i+1,j-1) & x(i+1,j) & x(x+1,j+1) \end{bmatrix}$$

- The algorithm scans the input image from left to right and top to bottom, quantizing each pixel one by one. And each time the error is diffused in the neighbors of the pixel that is getting processed i.e. the quantization error is getting transferred in the neighbors, without affecting the pixel that has already been quantized. Thus which provides an effect that if one pixel was rounded downwards, it is highly likely that the next pixel will be rounded upwards such that on an average the quantization error goes to zero.
- Various Error Diffusion Matrices are given below:

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

Floyd-Steinberg's error diffusion matrix *Jarvis, Judice, and Ninke (JJN)error diffusion matrix* *Stucki error diffusion matrix*

2.3) Results

2.a.3) Results

Figure 33: I2 Index Matrix *Figure 34: I4 Index Matrix* *Figure 35: I8 Index Matrix*

2.b.3) Results

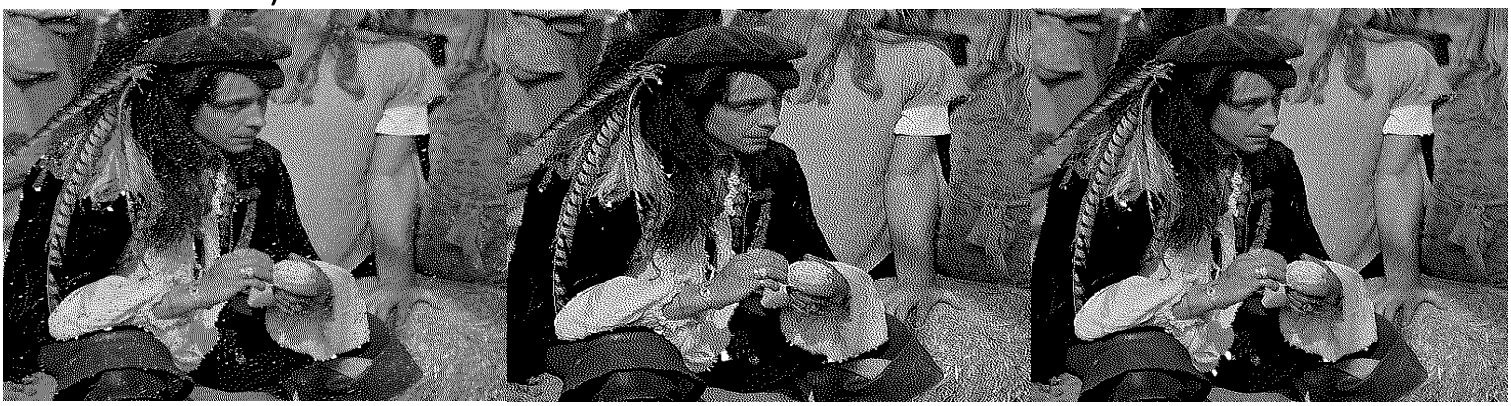


Figure 36 Floyd-Steinberg's error diffusion *Figure 37: Jarvis, Judice, and Ninke (JJN)error diffusion* *Figure 38: Stucki error diffusion*

2.4) Discussion

2.a.4) Discussion

- So we were asked in the problem statement that to use following A4 matrix in similar way as the Bayer's index matrices and then compare the results with I4 matrix:

```
0  8  2  10
12 1  14 3
3  11 1  9
15 4  13 2
```

I4 Matrix

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

Figure 39: I4 Index Matrix

Figure 40: A4 Index Matrix

Comparing both the results we can see that, the I4 matrix produces an image which with black and white dots, whereas the one produced via A4 matrix results in contains grid lines in it. This because the difference in the threshold matrix pattern.

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

The threshold values in the I4 matrix is much more close to neighbors, whereas in A4 the value jumps around according to the above shown pattern. So the pattern basically, makes the values around the center of A4 which is 0 to be small, i.e. to be on and as we go away from the center the values increase thus turning the them off. Thus in the center we get 1 and outside eventually we get zeros, thus producing the grid like effect. Whereas in I4 this is not the case thus we get smoother looking image.

- Comparison of the I2, I4 and I8 results.

It can be said that as the matrix size increases, thus increases the size of the threshold matrix. Which eventually indicates more levels of thresholds. Thus more chances of the dots being on. Due to which better resolution will be there as more shades of gray can be represented now. Thus I8 produces better results than I4 and I2. Similarly, I4 produces better results than I2.

- Comparison of Floyd, JJN and Stucki error diffusion.

The JJN and Stucki gives better results than Floyd Error diffusion based on following reason:

- Firstly, from general view point the man's image generated via Floyd error diffusion has less contrast as compared to that generated by other two.
- In the man's image generated by the Floyd error diffusion has more white spots, at various places which should had be black, like hair for instances.
- When the matrices are compared the Floyd matrix is smaller in size, thus it has less probability of turning the dot ON as compared to the other two.
- Also Floyd matrix puts more weight on the just immediate neighbors for errors, thus this leads to more sharp error diffusion. Whereas in the other two error is diffused to a bit far away neighbors as well. Also the weight is more balanced in the below rows in the other two.

- Algorithm for Four Intensity Level:

So Firstly I decided to find the midpoint of each range [0,85], [85,170], [170,255].

The Midpoints for each range will be the thresholds. Therefore, eventually I will have 4 thresholds.

E.g: The threshold for [0,85] would be 42;

Thus if the pixel intensity is less than this threshold then quantizes it to zero, otherwise if the intensity is above the threshold and below 85 then quantize it to 85.

Similarly, for other ranges as well.

Figure 41: Four Intensity Display for man.raw

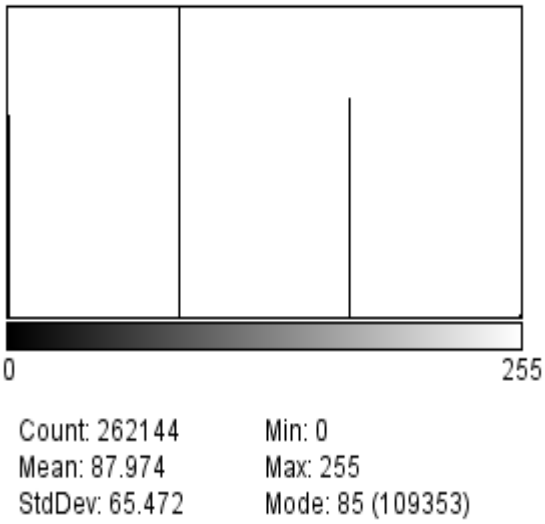


Figure 42: Histogram of Four Intensity Display for man.raw

- **My idea to implement better results in the Error Diffusion:**

I feel the hard thresholding can be replaced with adaptive thresholding, i.e. we can use Bayer's index matrices to do the thresholding. And eventually follow all most the same method of error diffusion.

Problem 3: Morphological Processing (30%)

In this problem, you will implement three morphological processing operations: shrinking, thinning, and skeletonizing. A pattern table (patterntables.pdf) is attached for your reference. Please show outputs for all following parts in your report and discuss them thoroughly. Please state any assumptions you make in your solution.

(a) Shrinking (Basic: 7%)

Please apply the 'shrinking' filter to the squares image (squares.raw). Please implement the filter, and discuss your solution for the following questions:

- Count the total number of squares in the image.
- How many different square sizes are present in the image? What is the frequency of these square sizes? (Hint: Plot the histogram of the square size with respect to frequency.)

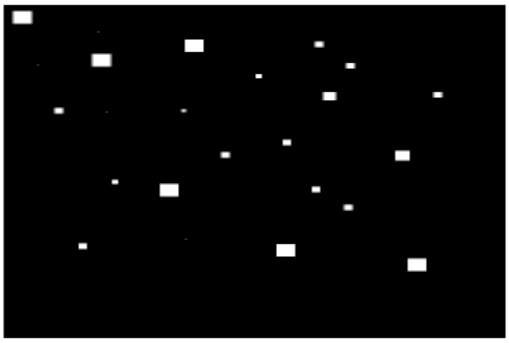


Figure 7: squares.raw

(b) Thinning (Basic: 7%)

Please apply the 'thinning' filter to the letterE image (letterE.raw) and show your result.

(c) Skeletonizing (Basic: 8%)

Please apply the 'skeletonizing' filter to the letterE image (letterE.raw) and show your result.

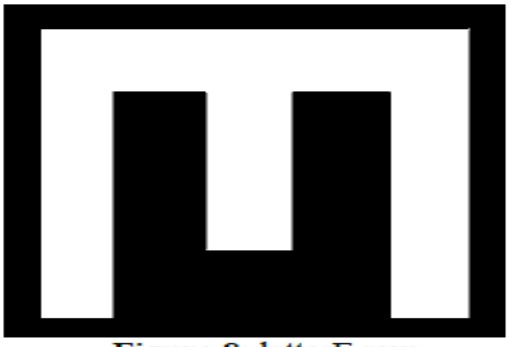


Figure 8: letterE.raw

(d) Counting game (Advanced: 8%)

Figure 3 (see below) is target image with circle and square objects (the background is black and the objects are white). Some of these objects have one or more holes in them. You need to design an algorithm that uses morphological and logical operations to solve and answer the questions below. You may use any of these operators that you learned in class or ones that you invent, but all of them must be detailed in your report with respect to how they operate and how their results help you solve the problem. In your report, discuss your algorithm, results, and analysis in detail. You should also submit your MATLAB or C/C++ programs along with your source code submission. State any assumptions you make for this problem.

Note: Do NOT use any built-in morphological processing functions or logical operators Note: Manual counting of objects is not permitted; your code must automatically do this.

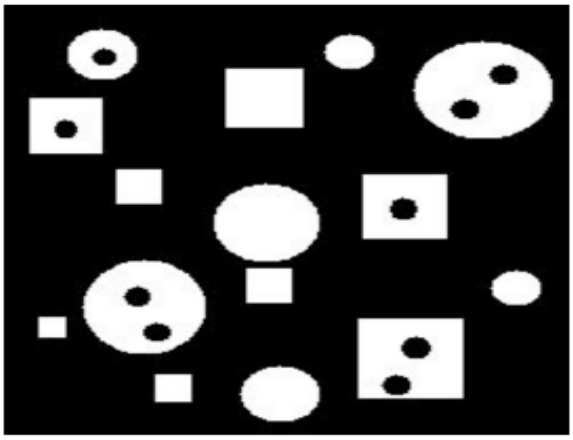


Figure 9: board.raw

- Find the total number of white objects in the image
- Find the total number of holes (black circular holes within white objects) in the image.
- Find the total number of white square objects (with or without holes) in the image.
- Find the total number of white circle objects (with or without holes) in the image.

3.1) Motivation

Morphological processing is imperative for applications which are centered around the shapes and form of the image. It involves various operations such as shrinking, thinning, skeletonizing. Erosion and dilations are the basic of all the morphological operators. These operations help in understanding and analyze the shape of the objects in the image. Various applications such as finger print recognition, character recognition, or image to text conversion uses morphological operators.

3.2) Approach

3.a.2) Shrinking

We were asked to perform shrinking on the given below image and answer various questions:

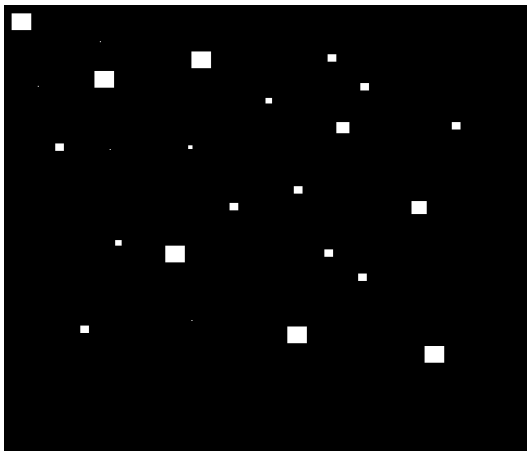


Figure 43: Input Image for Shrinking

Algorithm for Shrinking:

- We need binary image for performing morphological operations, due to which we firstly need to normalize the input image by dividing each pixel by 255, which would result in a 2d array consisting of only 1's and 0's.
- After that we need to construct an M array. The size of the M array will be same as that of the original image.
- Now we need to consider a window of 3*3 and we need to scan the binary image array.

X0	X1	X2
X7	X(i,j)	X3
X6	X5	X4

- If center value X(i,j)==0 then set M(i,j)=0; If it's 1 then go on to calculate the bond.
- Bond=2*(X1+X3+X5+X7) +(X0+X2+X6+X4).
- If the bond is between 1 and 10 then only proceed ahead otherwise set M(i,j)=0.
- If bond is between 1 and 10, then form a string (X0X1X2X7XX3X6X5X4) and compare this string with the string table provided. For restricting the string comparison, I used switch case in c++, so that if the bond is 1 then only the string comparison will be done with four patterns not all, and similarly for other bonds as well.
- If the string matches any of the corresponding bond pattern then set M(i,j)=1, if not then set M(i,j)=0.
- Now as soon as the whole binary image is scanned to form an entire M array move to next step.
- In this step we need to scan the M array. If M(i,j)==0. Then set F(i,j)=X(i,j); where F is the output image matrix and X is the input binary image matrix.
- If M(i,j)==1. Then compare (M0M1M2M7MM3M6M5M4) with the unconditional mask table provide for shrinking. The window size for scanning should again be 3*3.
- If there is a pattern match, then set F(i,j)=X(i,j), if not then set F(i,j)=0.
- Then after the entire M array is completely scanned copy the X(i,j)=F(i,j). Now the output becomes the input.
- This process would repeat until all the objects in the image is shrunk to just one white pixel.

3.b.2) Thinning

We were asked to perform thinning on the given below image and answer various questions:

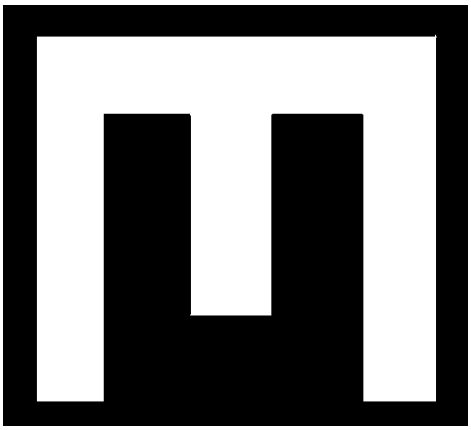


Figure 44: Input Image for Thinning and Skeletonizing

Algorithm for Thinning:

- The algorithm for thinning and shrinking is pretty much the same. The only difference is in the range of the bond values.
- Now If the bond is between 4 and 10 then only proceed ahead otherwise set M(i,j)=0.
- Rest everything is same.
- We need to stop thinning once we have lines which are just one pixel in size. And if we make our output equal to the input, then there is no change in the number of white pixels, then we need to stop.

3.c.2) Skeletonizing

We were to perform skeletonizing on the same letterE.raw image and generate the output.

Algorithm for Skeletonizing:

- The algorithm for Skeletonizing and thinning is pretty much the same. The only difference is in the range of the bond values.
- Now If the bond is between 4 and 10 and not equal to 5 then only proceed ahead otherwise set M(i,j)=0.
- Also the unconditional mask has string table has changed which is provided to us.
- Rest everything is same.
- We need to stop thinning once we have lines which are just one pixel in size. And if we make our output equal to the input, then there is no change in the number of white pixels, then we need to stop.

3.3) Results

3.a.3) Results

Figure 45: Shrinking Output

3.b.3) Results and 3.c.3) Results

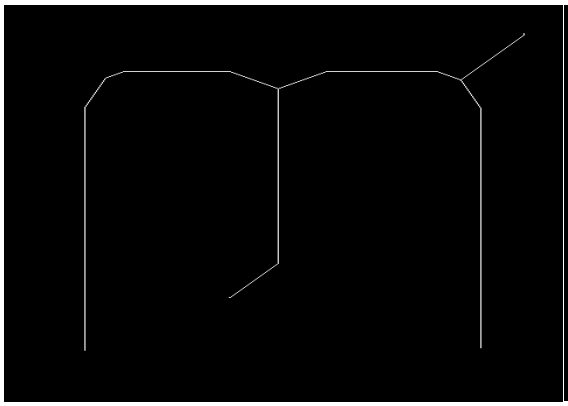


Figure 46: Thinning Output

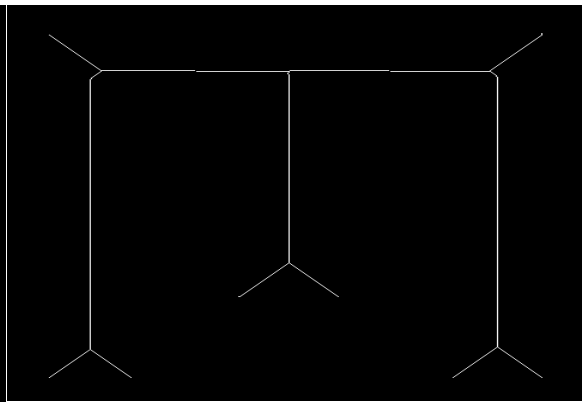


Figure 47: Skeletonizing Output

3.3) Discussion

3.a.3) Discussion

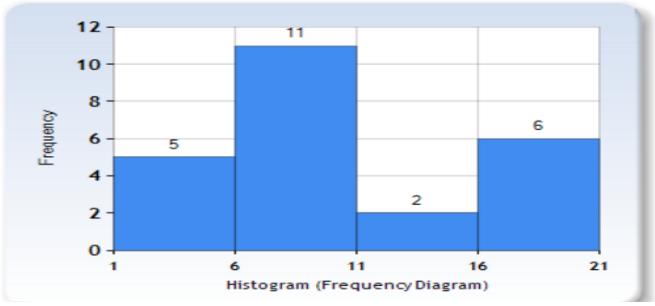
- We were asked to count the number of squares:

So for counting the number of squares it was imperative to shrink the all squares to one pixel in size. This took me 10 iterations to completely shrink all the squares. And eventually I found out the following result:

```
number of Squares:24
number of Iterations:10
```

- We were also asked to find the varying sizes of different squares:

```
Sizes of different Squares
18 1 18 8 18 8 1 6 12 8 8 4 1 8 14 8 6 18 8 8 1 8 18 18
```



Histogram of The Sizes of the square

3.b.3 and 3.c.3) Discussion

number of Iterations:67 number of Iterations:127

Above given are the number of iterations required for thinning and skeletonizing respectively.

3.d) Counting Game

So in this part of the question we were asked to count the number of the objects in the given below image and answer various questions. It actually required us to put in our knowledge of morphological processing:

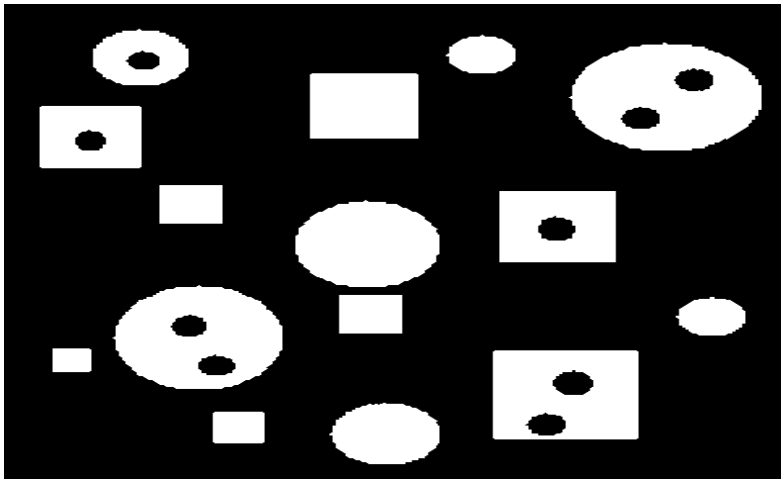


Figure 48: Board Input Image

It can be seen from the above image that it contains variety of circles and squares, some without holes in the center and some with holes in the center.

3.d.2) Approach & 3.d.3) Results & 3.d.4) Discussion

- We were asked following questions:
  1. Find the total number of white objects in the image

So to solve this problem, we had to count number of white object in the image. In total there are total 15 white objects few with holes and few without holes. But how to count them that was the question. I implemented following algorithm to achieve that:

- Firstly, here shrinking was imperative, but we can't directly shrink the image as many white objects have holes in them. Due to which it was important to fill those holes and then shrink them.
- But for filling those holes it was imperative to find the center of those holes after which we can apply flood fill algorithm to fill those holes.
- Thus to find those center, I firstly normalized and complemented my image:

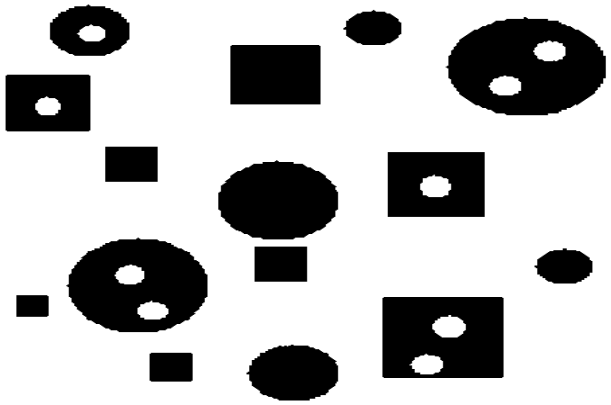


Figure 49: After complementing the image

- After complementing, I applied the shrinking logic, up till 15 iterations to get the following output:

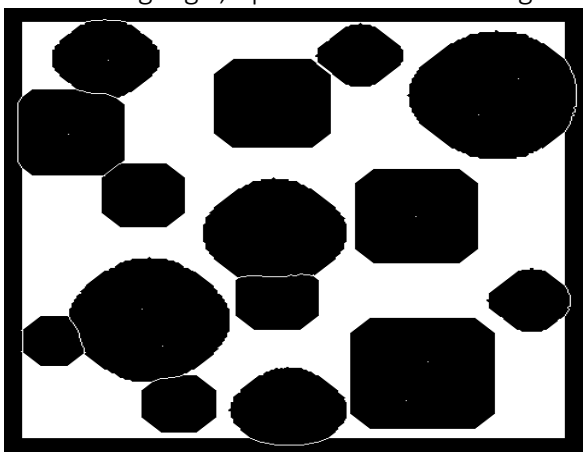


Figure 50: After complementing and Shrinking

- Now the white pixels in the center of the objects represent the holes. And I want to their coordinates, thus I used a mask of 3*3 which will scan the entire above image to check for isolated 1's surrounded by all zeros. And consecutively stored their coordinate locations.
- Now as I have the coordinates or the center locations of each holes I can consider the original Input Image to fill those holes via flood fill algorithm.

```
Flood-fill (node, target-color, replacement-color):
 1. If target-color is equal to replacement-color, return.
 2. If the color of node is not equal to target-color, return.
 3. Set the color of node to replacement-color.
 4. Perform Flood-fill (one step to the south of node, target-color, replacement-color).
    Perform Flood-fill (one step to the north of node, target-color, replacement-color).
    Perform Flood-fill (one step to the west of node, target-color, replacement-color).
    Perform Flood-fill (one step to the east of node, target-color, replacement-color).
 5. Return.
```

Figure 51: Flood Fill Algorithm [1]

- I followed the above given flood fill algorithm where the node is the center points of the holes. Target color is black and replacement color is white.
- After following this algorithm, I was able to fill all the holes to get the following result:

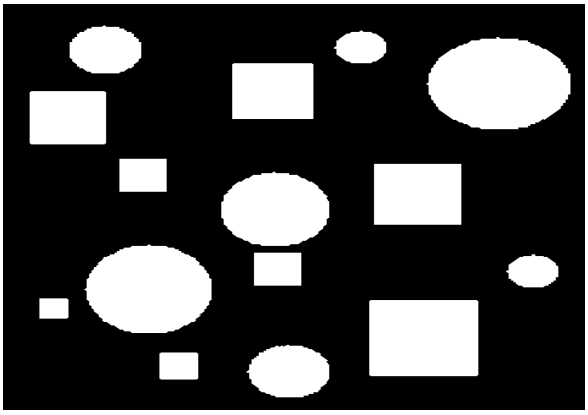


Figure 52: After Flood Filling the holes

- Now as all the holes are filled we can easily apply the normal shrinking logic, to reduce the all objects to just one pixel and then we can count the number of white pixels in order to get the number of white objects.

Figure 53: Final Image After Shrinking

2. Find the total number of holes (black circular holes within white objects) in the image.

- This question was easily answered by just count the white pixels surrounded by all zeros, after complementing and shrinking.

3. Find the total number of white square objects (with or without holes) in the image.

- After the final shrinking is done, coordinate locations of each white pixel denotes the center of each object. Now we have regular square and irregular circles.

- Thus if we calculate the distance to each side from the center and compare it we can differentiate between the squares and the circles. If the distances are equal, then it is a square. Thus increment the counter of the square. If not, then increment the counter of the circle.
- Thus eventually we will get count of squares and circles.

4. Find the total number of white circle objects (with or without holes) in the image.

- The count of circles from above step would give me the number of circles.

Final Result:

```
Number of Holes :9
number of white objects:15
Number of Squares:8
Number of circles:7
```


REFERENCES:

[1]www.wikipedia.com

[2] http://electronicimaging.spiedigitallibrary.org/data/journals/electim/23507/013018_1_1.png

[3] www.youtube.com

[4]www.google.com