

EE 569

PROJECT #2

BY

PRANAV GUNDEWAR

USC ID: 4463612994

EMAIL: gundewar@usc.edu

Table of Contents

Problem 1: GEOMETRICAL IMAGE MODIFICATIONS

- a) Geometrical Warping
- b) Homographic Transformation and Image Stitching

Problem 2: DIGITAL HALF-TONING

- a) Dithering
- b) Error Diffusion
- c) Color Half-toning with error diffusion

Problem 3: MORPHOLOGICAL PROCESING

- a) Shrinking
- b) Thinning
- c) Skeletonizing
- d) Counting Game

PROBLEM 1 – GEOMETRICAL IMAGE MODIFICATIONS

GEOMETRICAL WARPING

Task: Convert square input image into an output image of disk-shaped image.

ABSTRACT AND MOTIVATION

Geometrical image manipulation and warping involves set of three most common image processing operations such as rotation, translation and scaling of the image in spatial domain to create sort of visual effect. This is referred as image warping and involves lots of manipulation such that image projection easily matches surface projection or corresponding shape. They are extensively used in computer graphics, image morphing and panorama stitching. Geometrical warping provides transformation in spatial domain of the image.

APPROACH AND PROCEDURE

Geometrical warping depends on perspective of the image. To convert square image into a disk image and vice-versa, we need to obtain the cartesian co-ordinates of the image and convert them into polar co-ordinates to map from square to circle.

We prefer working in cartesian co-ordinates because we can easily apply filters and effects for translation, rotation and scaling. Hence first step to proceed would be getting cartesian co-ordinates from image co-ordinates. Since we work in augmented domain, we add one (append 1) in the end. The image to cartesian co-ordinates is given by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0.5 \\ -1 & 0 & h - 0.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r \\ c \\ 1 \end{bmatrix}$$

Where,

r & c – image row, column co-ordinates

h – image height

By taking inverse of given matrix, we obtain image co-ordinates by inverse mapping.
To convert disk image into square and vice-versa, we use following transformation:

Elliptical Grid Mapping

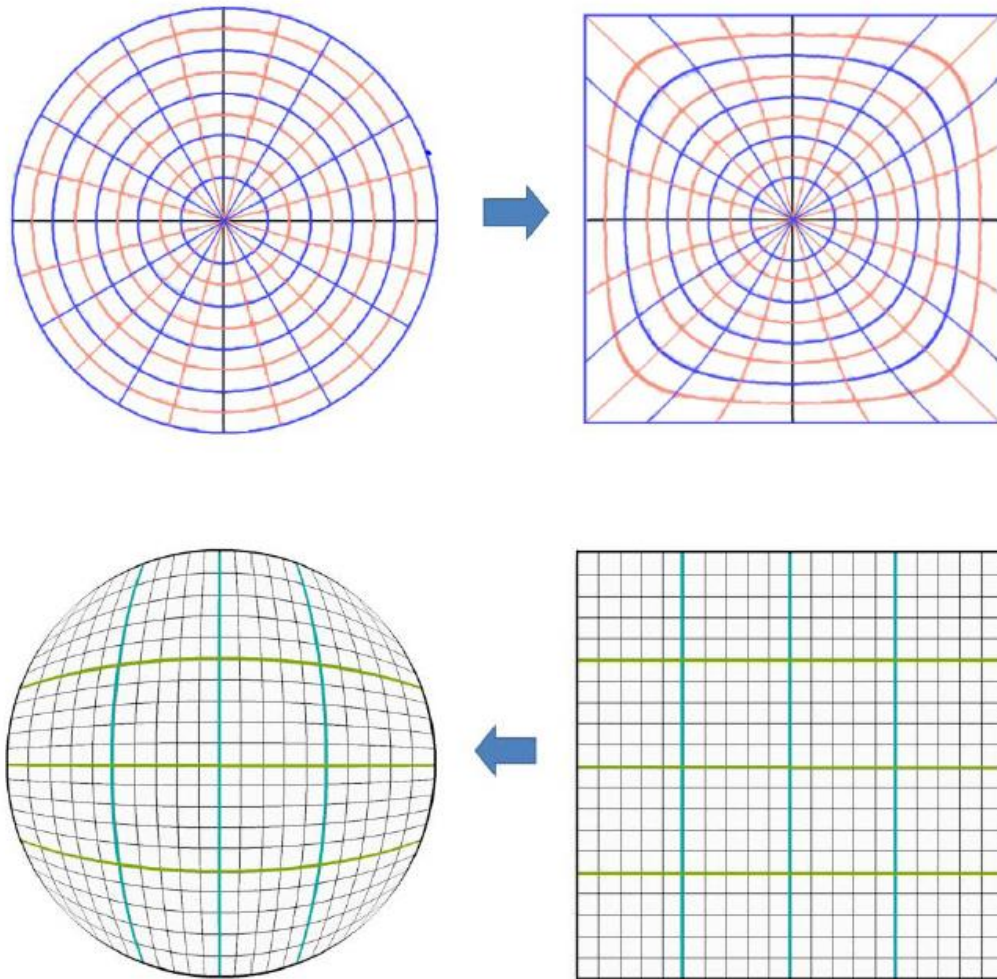


Figure 1 – Elliptical grid mapping

(Source - Analytical Methods for Squaring the Disc)

In mapping, we want map every point to the output image with minimal distortion as much as possible. We will

Reverse Mapping -

In reverse mapping, for every co-ordinate on circle, we find its equivalent co-ordinate from square image using following formula:

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

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

Figure 2 – Disc to Square mapping

(Source - Analytical Methods for Squaring the Disc)

Where u,v are the co-ordinates in disc which are being mapped from x,y co-ordinates of square. The idea was to find circle co-ordinates within a given radius that are being mapped by an equation from square image.

Reverse mapping results into less distortion as it is 1:1 mapping since it keeps relationship between pixels horizontally next to each other. But there is one drawback- it creates some distortion around top and bottom boundary when used in forward mapping. After the points are mapped, we use bilinear interpolation to minimize pixel error by:

Let **I** be an $R \times C$ image.

We want to resize **I** to $R' \times C'$.

Call the new image **J**.

Let $s_R = R / R'$ and $s_C = C / C'$.

Let $r_f = r' \cdot s_R$ for $r' = 1, \dots, R'$

and $c_f = c' \cdot s_C$ for $c' = 1, \dots, C'$.

Let $r = \lfloor r_f \rfloor$ and $c = \lfloor c_f \rfloor$.

Let $\Delta r = r_f - r$ and $\Delta c = c_f - c$.

Then $J(r', c') = I(r, c) \cdot (1 - \Delta r) \cdot (1 - \Delta c)$
 $+ I(r+1, c) \cdot \Delta r \cdot (1 - \Delta c)$
 $+ I(r, c+1) \cdot (1 - \Delta r) \cdot \Delta c$
 $+ I(r+1, c+1) \cdot \Delta r \cdot \Delta c.$

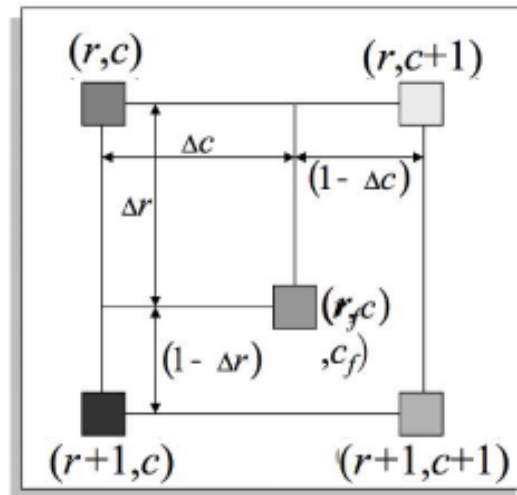


Figure 3 – Bilinear Interpolation Algorithm

(Source - <https://www.codementor.io/tips/7814203938/resize-an-image-with-bilinear-interpolation-without-imresize>)

Forward Mapping –

In forward mapping, for every co-ordinate on square, we find its equivalent co-ordinate from circle image using following formula:

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

Figure 4 – Square to Disc mapping

(Source - Analytical Methods for Squaring the Disc)

Where u,v are the co-ordinates in square which are being mapped from x,y co-ordinates of disc. The idea was to find square co-ordinates that are being mapped by an equation from disc image but within radius.

Forward mapping is not 1:1 mapping hence it will introduce some distortion around the boundary which will be discussed below.

DEVELOPED ALGORITHM-

STEP 1 – Read the input image and store it in 1D array. Get height, width and bytes per pixel.

STEP 2 – Convert 1D array into 3D for easier pixel manipulation.

STEP 3 – Convert the image co-ordinates from 0 to height and width to -1:1 to use the formula. For every i, j, k pixel location, use the reverse mapping formula to find corresponding pixel location from square image.

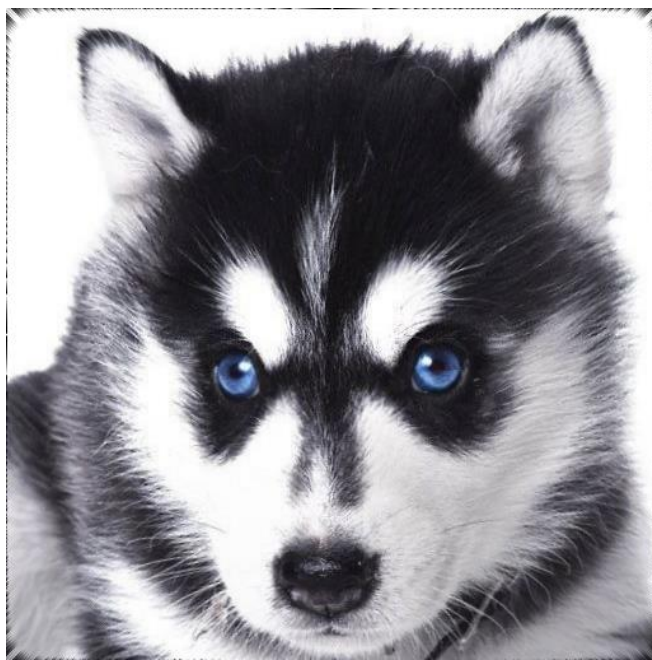
STEP 4 – Using bilinear interpolation, find the pixel intensity and copy the value in output image. At the output, you will get disc image mapped by square image.

STEP 5 - Convert the image co-ordinates from 0 to height and width to -1:1 to use the formula. For every i, j, k pixel location, use the forward mapping formula to find corresponding pixel location from disc image.

STEP 6 - Using bilinear interpolation, find the pixel intensity and copy the value in output image. At the output, you will get square image mapped by disc image

STEP 7 – Output 3D arrays was then converted to 1D for writing to the file which is then retained to the file using write file function.

EXPERIMENTAL RESULTS





Reverse Mapping output



Forward Mapping output

Figure 5 – Geometrical Warping

Figure shows output images using forward and reverse mapping. Forward mapping introduces distortion.





Original Image



Output using forward mapping

Figure 5 – Image Distortion

DISCUSSION

The described formula works for radius of circle 1 i.e. we have to convert co-ordinates from 0 to 512 to -1:1. Note that this is important to make sure center of circle is mapped to the center of mapped image.

Shape of output image looks little distorted and we can observe distortion at the corners because forward mapping is not one to one mapping. Main reason being we have discrete image. For continuous images, transformation transforms every pixel into new position i.e. we can find original pixel for every pixel in transformed image. Thus, we can recover all pixels while doing reverse pixels. Because we are dealing with discrete images, we might not be able to recover all the pixels which introduces distortion. We lose information when we transform line into shorter line.

HOMOGRAPHIC TRANSFORMATION & IMAGE STITCHING

Task: Using homographic transformation and image stitching, create panorama effect consisting of multiple images.

ABSTRACT AND MOTIVATION

The homographic transformation and image overlay techniques can be used to synthesize multiple images giving new perspective. In homographic transformation, we will synthesis and stitch 3 images that are taken from different camera viewpoints. The logic behind it is while converting the world co-ordinates to pixel co-ordinates. For any real-world object, its position is affected by focal length, distance and orientation of the camera. The projective plane is superset of real plane where all 2D points can be projected into other space. The homography between 2 planes is given by for ant given 4 points in plane, there always exists transform that maps them into corresponding 4 points in another plane or space.

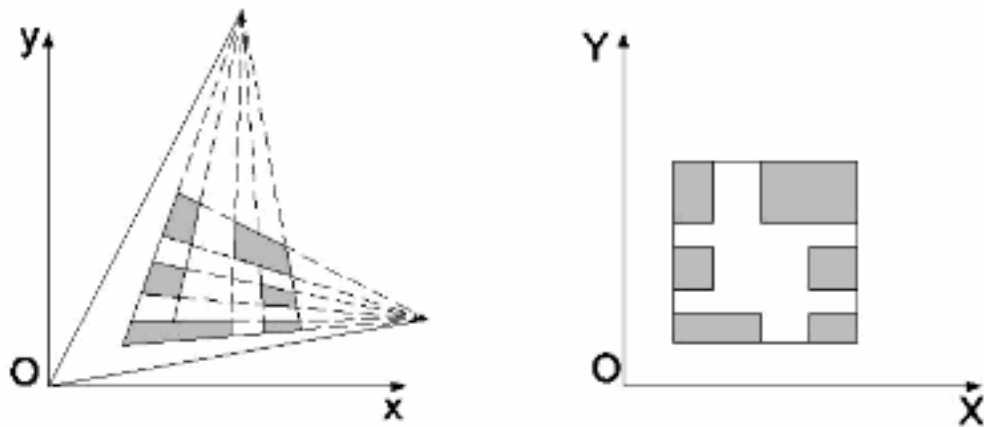


Figure 6 – The perspective-to-plane transformation using homography

(Source - http://www.corrmap.com/features/homography_transformation.php)

The homography transformation method has been extended to maps when aerial photographic techniques have been used. in this type of survey, the homography is used by assuming the map as a perspective view of the ground [1]. This can be observed from figure 7.

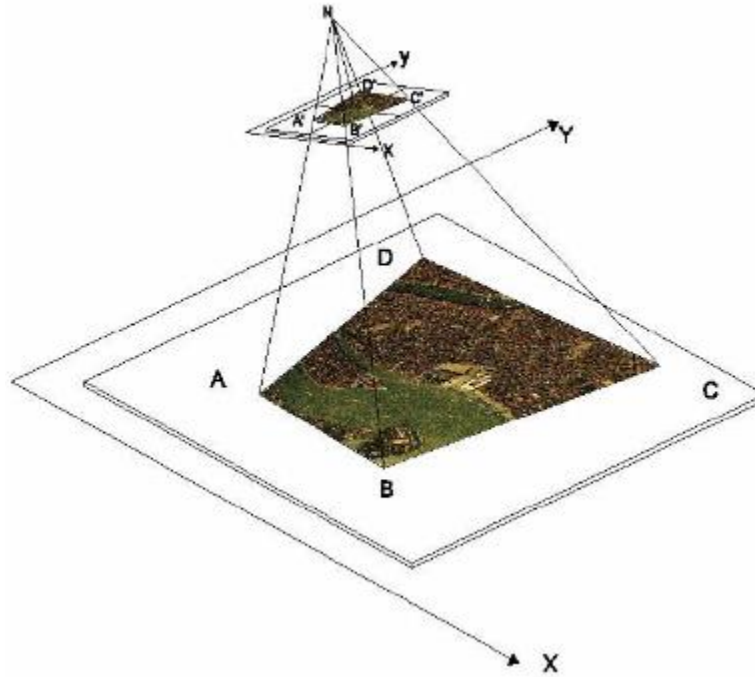


Figure 7 – For aerial photographic surveys the homography technique assumes the map as a perspective view of the ground

(Source - http://www.corrmap.com/features/homography_transformation.php)

APPROACH AND PROCEDURES

The basic approach to get image plane coordinates from world coordinates is through following equation:

$$w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

After we get the image plane coordinates, again we have to convert it to image index based on pixel density. All three operations can be combined into one h matrix. Images of points in a plane, from two different camera viewpoints, under perspective projection (pin hole camera models) are related by a homography:

$$P2 = HP1$$

where H is a 3x3 homographic transformation matrix, $P1$ and $P2$ denote the corresponding image points in homogeneous coordinates before and after the transform, respectively [2].

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

The formula can be further expanded

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -y_2X_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3X_3 & -y_3X_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4X_4 & -y_4X_4 \\ 0 & 0 & 0 & x_1 & y_1 & 0 & -x_1Y_1 & -y_1Y_1 \\ 0 & 0 & 0 & x_2 & y_2 & 0 & -x_2Y_2 & -y_2Y_2 \\ 0 & 0 & 0 & x_3 & y_3 & 0 & -x_3Y_3 & -y_3Y_3 \\ 0 & 0 & 0 & x_4 & y_4 & 0 & -x_4Y_4 & -y_4Y_4 \end{bmatrix} \times \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix}$$

Figure 7 – H matrix

(Source - http://www.corrmmap.com/features/homography_transformation.php)

where,

H_{11} =fixed scale factor in X direction with scale Y unchanged

H_{12} =scale factor in X direction proportional to Y distance from origin

H_{13} =origin translation in X direction.

H_{21} =scale factor in Y direction proportional to X distance from origin

H_{22} =fixed scale factor in Y direction with scale X unchanged

H_{23} =origin translation in Y direction

H_{31} =proportional scale factors X and Y in function of X

H_{32} =proportional scale factors X and Y in function of Y

$H_{33}=1$

To calculate the 3x3 homographic transformation matrix, we have implemented the least square's method by keeping $h_{33} = 1$. This constraint is imposed as we have only eight equations to solve for 9 variables. X- Y are the co-ordinate to be calculated from second system in function of 8 parameters given by 8 unknowns. To calculate 8 unknowns, we need at least 4 control points. After calculating the matrix, using reverse projection, we can project the moving points onto fixed points.

DEVELOPED ALGORITHM-

STEP 1 – Read the input image and store it in 1D array. Get height, width and bytes per pixel.

STEP 2 – Convert 1D array into 3D for easier pixel manipulation.

STEP 3 – Find 4 control points which will cover very good area from left image and right image having bigger dimensions.

STEP 4 – Find projection matrix using these 4 control points and using reverse mapping, project the left image onto middle image of bigger dimensions.

STEP 5 -Perform similar process for middle and right image. Using new projection matrix, project right image into the stitched left and middle image.

STEP 6 – Output 3D arrays was then converted to 1D for writing to the file which is then retained to the file using write file function.

EXPERIMENTAL RESULTS

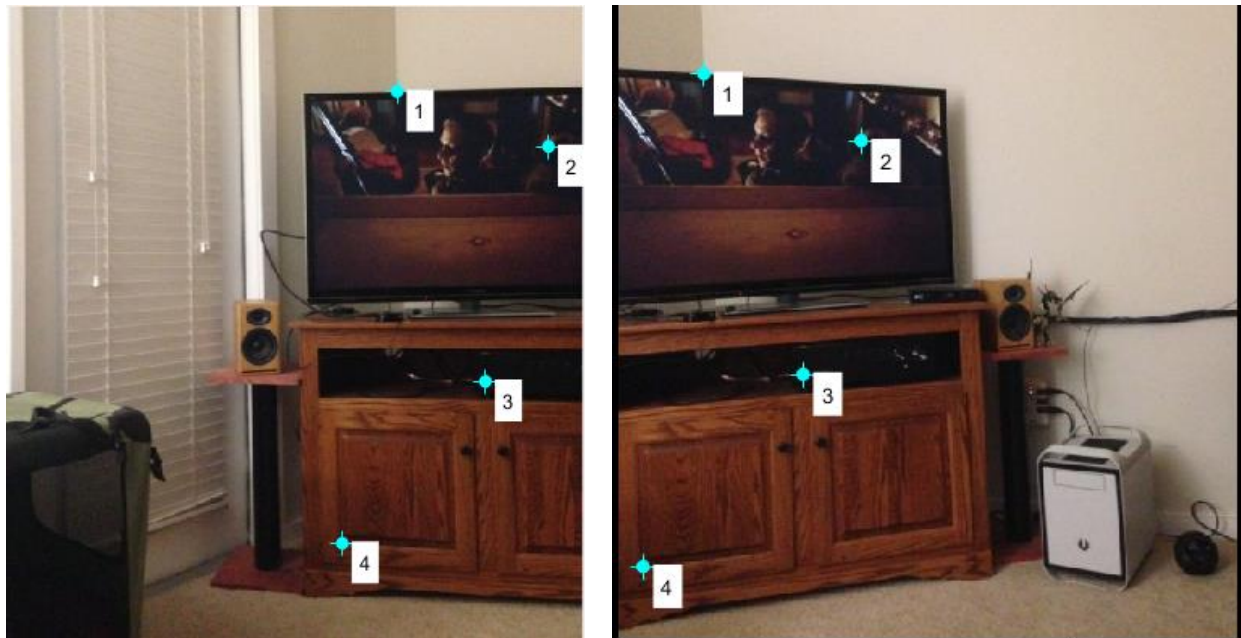


Figure 8 – Control points for left and middle image

We will discuss how to choose control points, why it is necessary to choose 4 diverse control points. For left and middle image,

Control points for middle: (867 390) (989 442)

(944 623) (820 772)

Control points for left: (326 196) (452 240)

(399 429) (280 559)

For right and middle image,

Control points for middle: (1052 403) (1032 564)

(1188 779) (1119 638)

Control points for right: (53 197) (29 366)

(192 569) (126 435)

The sequence of image stitching is not important. We can perform the operations on either left or right image first.

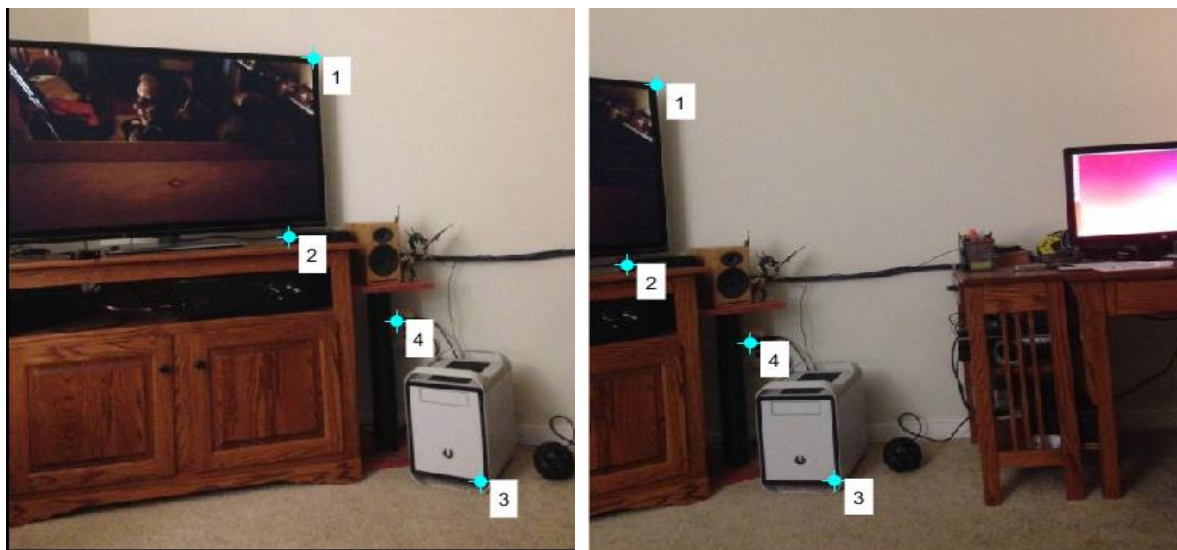


Figure 9 – Control points for middle and right image

After we carefully choose 4 control points, their respective matrices, we project left and right image onto higher dimension middle image and we will get final stitched output.

The right image has slightly different light illumination hence we can see some difference in light intensity as we stitch the image.



Figure 10 – Final stitched output

DISCUSSION

The final stitched output is shown in figure 10. It is almost perfect stitching of given three images. Due to change in light intensity, we can see little dark illumination. Main thing that we have to consider that middle image's dimension must be at least three times the original dimensions. After finding out projection matrix, using bilinear interpolation we project left image pixels into middle to minimize the distortion. As we are stretching the image, bilinear interpolation is important as it reduces distortion near edges.

Selection of control points is very important. While selecting control points, there are 2 things to be considered- its distance from rest of the control points and how accurately you find the location of pixel. Maximum errors will occur in these two things only. For calculation of H matrix, we have 8 equations to solve for nine variables and that is the reason we assume H_{33} as one since it is free variable. Also, for 8 equations, we need at least 4 control points to solve the equation. Hence control points less than 4 would not be the best estimate for projection matrix. As number of control points increases, we get more accurate matrix and stitching will be very close to perfect. The performance wise, this technique produces accurate output as the image gets overlaid perfectly without any deviations as observed in output.

PROBLEM 2 – DIGITAL HALF-TONING

DITHERING

Task: Implement the following four methods to convert *colorchecker* to half-toned images.

ABSTRACT AND MOTIVATION

Dithering is type of thresholding where grey or color image gets converted to cluster of binary pixels. It helps us reduce the set of colors by approximating unavailable colors with available colors. In general, it means applying some noise to signal to randomize the quantization error. It is a way of doing half-toning which represents different shades of grey color by placing black marks on white paper as most of the times background is assumed to be white. It is difficult for the human vision to distinguish two dots very close to each other.

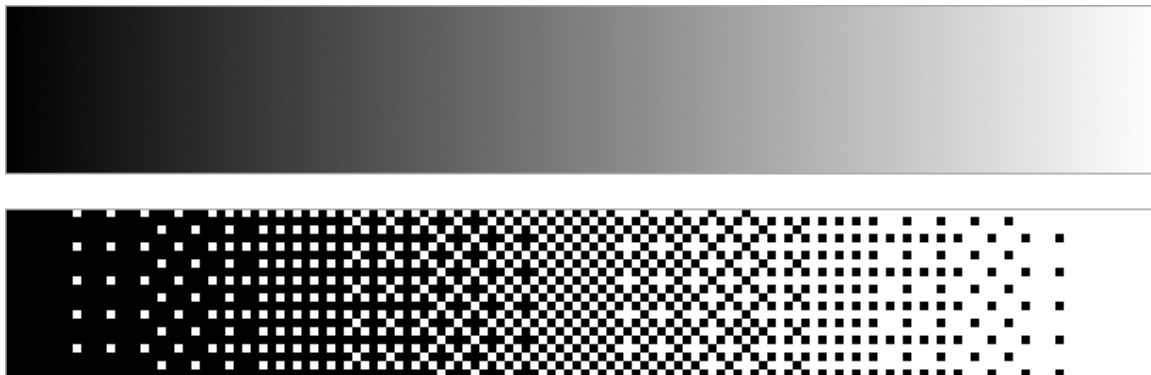


Figure 11 – Black to white gradient with dithered version

(Source - http://alex-charlton.com/posts/Dithering_on_the_GPU)

For digital dithering, we traverse various dithering matrices through the image. When pixel value in image is greater than matrix value, output image corresponds to black dot for that location. Ordered dithering is used to introduce blue noise in the image and approximates the unavailable colors which will result into lesser color. Printing high color image using limited color sets available will result in changing some colors into more subtle color is dithering is not performed.

APPROACH AND MOTIVATION

Fixed thresholding:

We choose a fixed value as threshold and set all pixels which below this threshold to be black and above it to be white.

Transfer function is as follows:

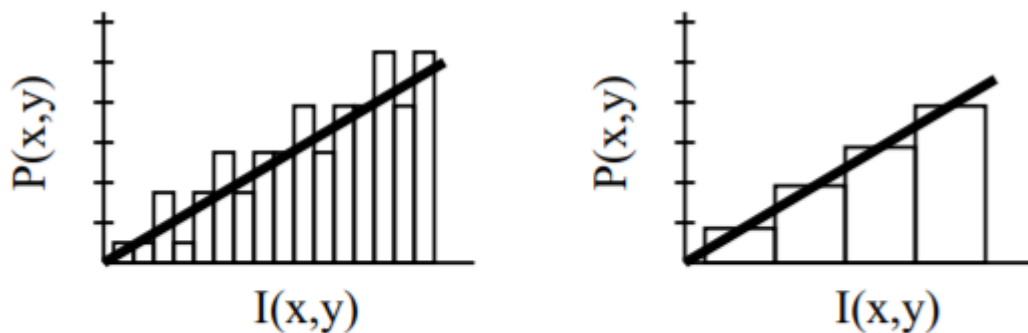
$$G(i, j) = \begin{cases} 0, & \text{if } 0 \leq F(i, j) < T \\ 255, & \text{if } T \leq F(i, j) < 256 \end{cases}$$

where T is the fixed threshold. Clearly, it can be expected that we will lose many details hence is never used in actual practice.

Random thresholding:

We use a random threshold for each pixel in this method. Transfer function is:

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



$$P(x, y) = \text{trunc}(I(x, y) + \text{noise}(x,y) + 0.5)$$

Figure 12 –Random Dithering

(Source <https://www.cs.princeton.edu/courses/cs426/lectures/dither/dither.pdf>)

Random function is random number generator for a particular distribution generally uniform distribution. The error appears as noise. Since the threshold is random, output will not be the same every time.

Ordered Dithering:

This algorithm achieves dithering by using pre-calculated threshold from different ordered matrices. Matrices stores pattern of thresholds.

Dithering parameters are specified by an index matrix. The values in an index matrix indicate how likely a dot will be turned on. Bayer index matrices that are square matrices with length of sides as power of 2. For example, an index matrix is given by

Bayer matrices are calculated using recursion and we will apply I2, I4 and I8 matrices for dithering.

$$I_{2n}(i,j) = \begin{bmatrix} 4 * I_n(x,y) & 4 * I_n(x,y) + 2 \\ 4 * I_n(x,y) + 3 & 4 * I_n(x,y) + 1 \end{bmatrix}$$
$$I_2 = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix} \quad I_4 = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix} \quad I_8 = \begin{bmatrix} 0 & 32 & 8 & 40 & 2 & 34 & 10 & 42 \\ 48 & 16 & 56 & 24 & 50 & 18 & 58 & 26 \\ 12 & 44 & 4 & 36 & 14 & 46 & 6 & 38 \\ 60 & 28 & 52 & 20 & 62 & 30 & 54 & 22 \\ 3 & 35 & 11 & 43 & 1 & 33 & 9 & 41 \\ 51 & 19 & 59 & 27 & 49 & 17 & 57 & 25 \\ 15 & 47 & 7 & 39 & 13 & 45 & 5 & 37 \\ 63 & 31 & 55 & 23 & 61 & 29 & 53 & 21 \end{bmatrix}$$

Figure 13 –Ordered Dithering

(Source https://en.wikipedia.org/wiki/Ordered_dithering)

The given Bayer matrix is then assigned a pixel intensity between 0 to 255 based on likelihood of pixel activation function. For dithering, any N*N window is traversed throughout the image and if input pixel value is greater than the threshold set in the map, we set it to 1 else 0.

$$T(x,y) = \frac{I(x,y) + 0.5}{N^2}$$
$$G(i,j) = \begin{cases} 1 & \text{if } F(i,j) > T(i \bmod N, j \bmod N) \\ 0 & \text{otherwise} \end{cases}$$

Where G(i,j) is normalized output. Binary image which will either contain 0 or 1.

For 4 grey intensity levels, I have designed a novel algorithm based on 2 dithering matrices to get best output. I have chosen I4 & I8 matrices as larger window size give better result. Depending upon the threshold for I8 matrix, we will decide whether pixel has intensity in the upper range or lower range. After we find which range pixel value belongs to, we use I4 dithering matrix thresholding to decide if pixels belong to higher value in given range or not. This algorithm can be explained as follows:

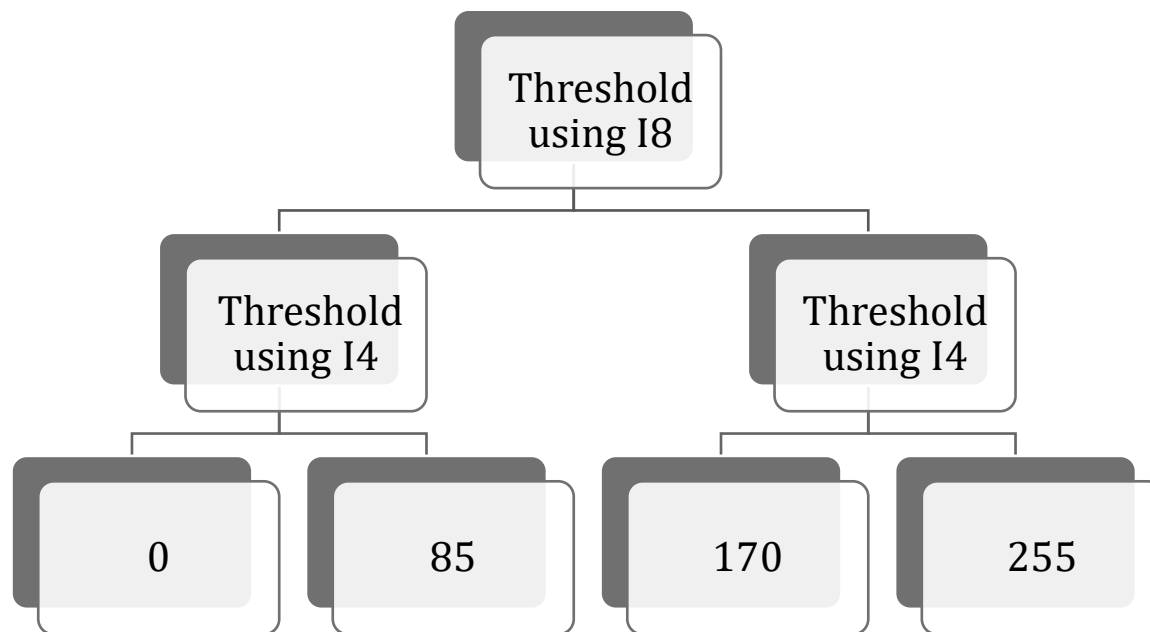


Figure 14 – Algorithm for 4 intensity levels

DEVELOPED ALGORITHM-

STEP 1 – Read the input image and store it in 1D array. Get height, width and bytes per pixel.

STEP 2 – Convert 1D array into 2D for easier pixel manipulation.

STEP 3 – For every pixel, threshold the image using random and fixed thresholding. Output will be binary image.

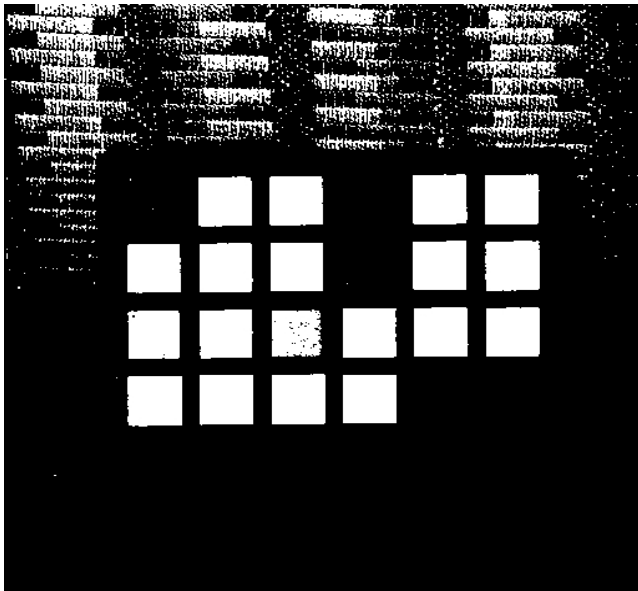
STEP 4 – Find ordered dithering, using recursive functions, find dithering matrices for window 2,4 & 8.

STEP 5 – Using obtained dithered matrices, for every pixel in the image, compare with the threshold obtained by matrices and set accordingly zero or one based on value.

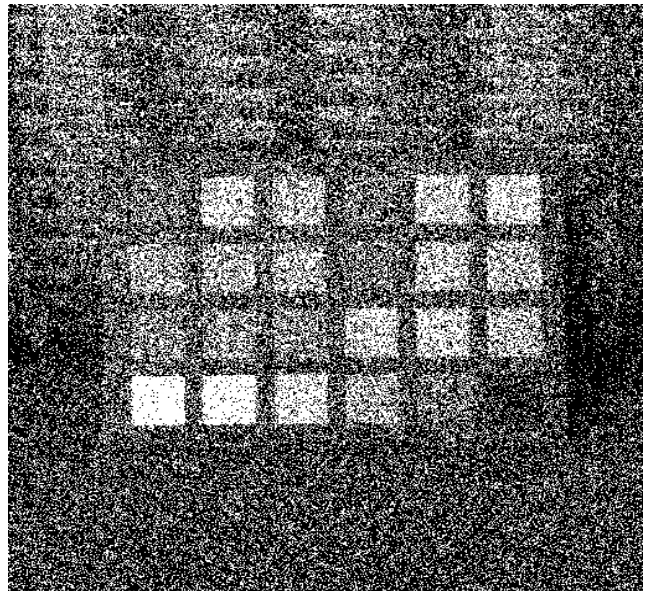
STEP 6 – For four intensity grey level, I have used I4 & I8 dithering matrices. While thresholding, I8 will decide whether pixel belongs to upper class (170,255) or lower class (0,85). I4 will decide which grey value it belongs to depending upon threshold.

STEP 6 – Output 2D arrays was then converted to 1D for writing to the file which is then retained to the file using write file function.

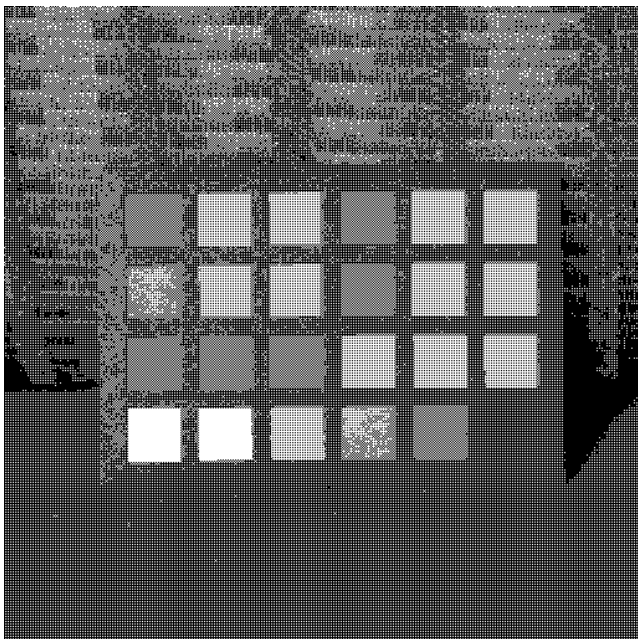
EXPERIMENTAL RESULTS



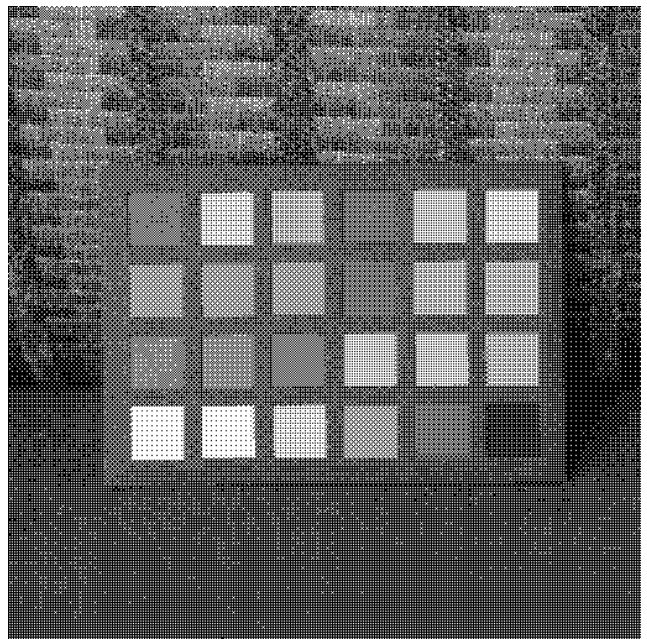
Binary threshold



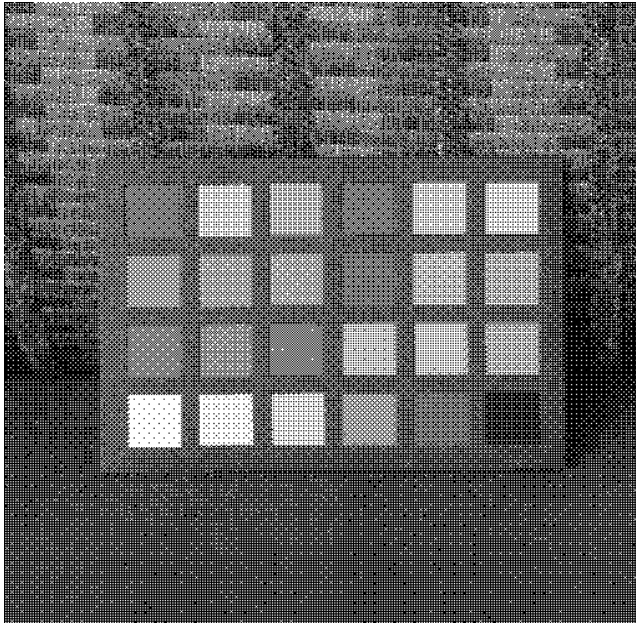
Random threshold



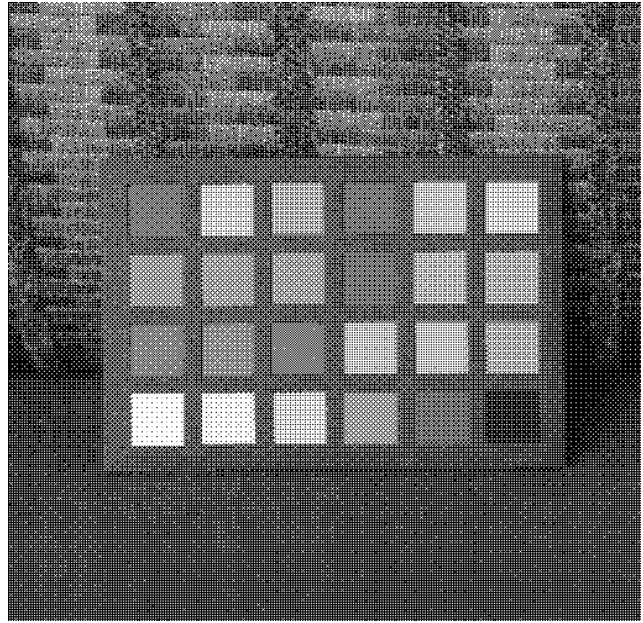
Using I2 dithering matrix



Using I4 dithering matrix



Using I8 dithering matrix



For four grey intensity levels

The table shows half toning output of color checker image using different thresholding method. Binary output loses most of its information. Random number generator depends upon underlined distribution and its effects will be discussed in next section.

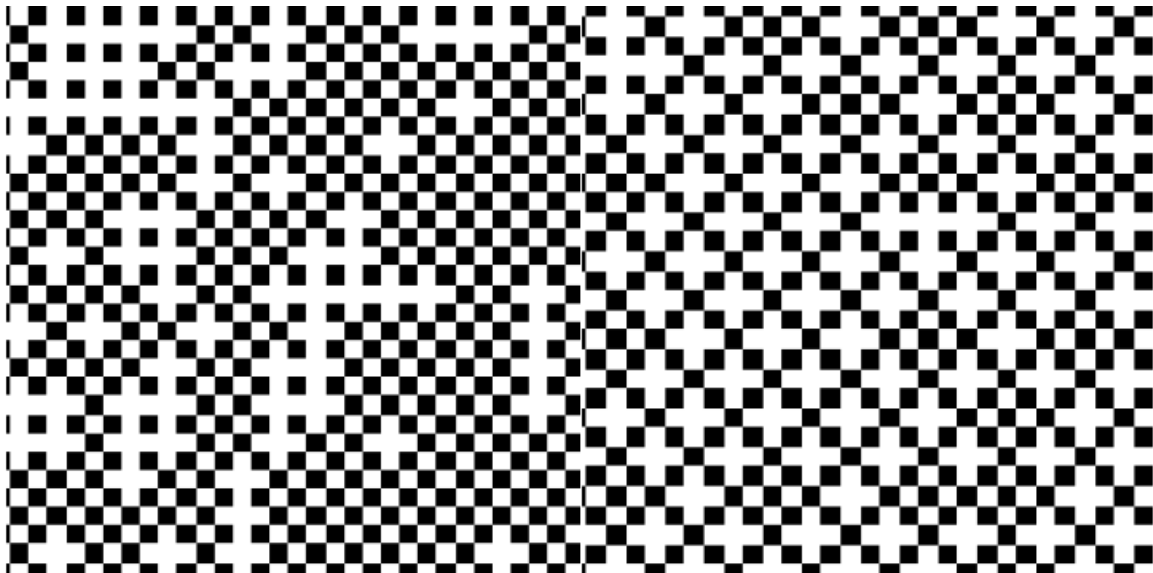


Figure 15 – Enlarged view of I2 and I8 pattern

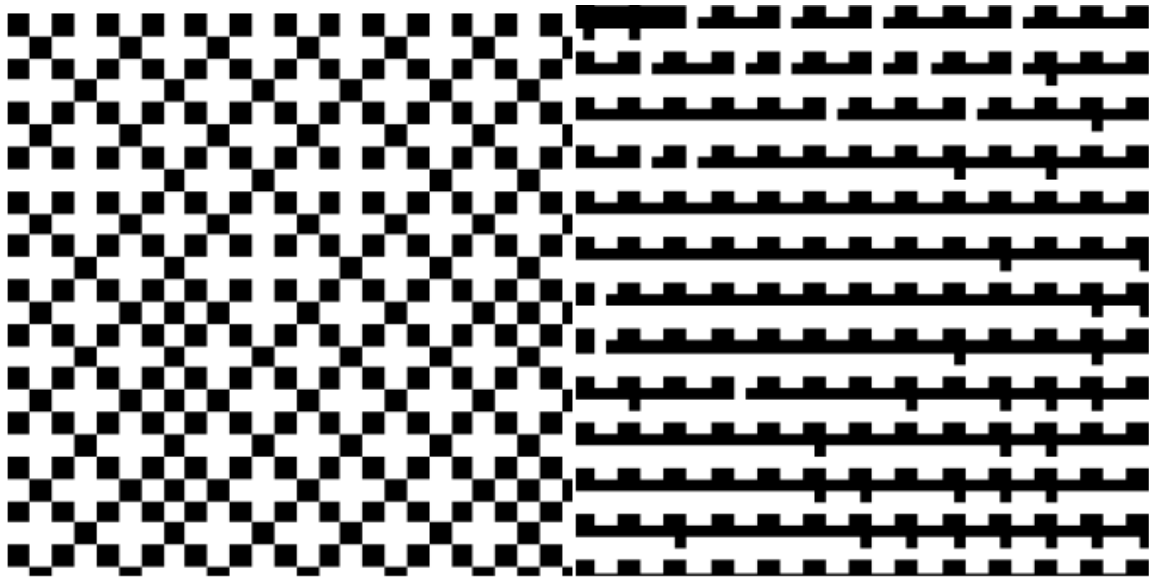


Figure 16 – Enlarged view of I2 and I4 pattern

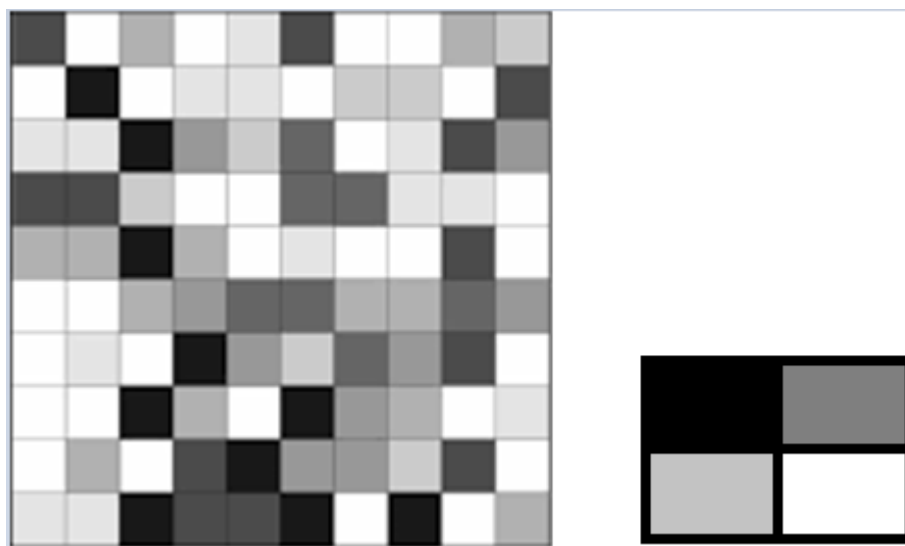


Figure 17 – Enlarged view of 4 grey level intensity

DISCUSSION

Half-toning of color checker image is performed using different methods and looking at the table, we can easily point out advantages and disadvantages of all methods.

Fixed thresholding results into massive loss of information. Random thresholding using Gaussian or Uniform distribution results into distorted output. Gaussian PDF generated much sharper image than uniform PDF.

Bayer matrix of varied sizes distribute the quantization error in different manner for each pixel value. From figure 15 and 16, we can observe periodic pattern introduced by

each dithering matrix. The pattern that I4 exhibits is undesired in half-toning. It has circular pattern which starts from inside and follows out of the matrix. This is one of the main reason why we observe black outlines in the output image. Dithering matrices I2 and I8 exhibits plus sign pattern which smoothens the output image. Hence, in practice I8 matrix is used widely because of its unique pattern. Half – toning with dithering matrix is used popularly but it invites undesired patterns which when looked closely portrays distortion.

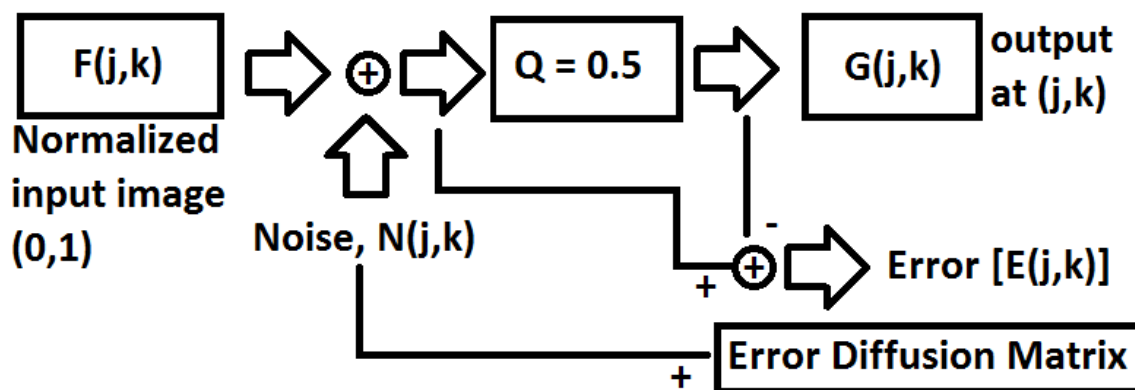
Half-toning using 4 grey levels looks more dithered and produces good mixture of all the levels as density of dots is less compared to Bayer matrices. But due to this, image looks like little grainy but overall looks good.

ERROR DIFFUSION

Task: Convert *colorchecker* to half-toned image using error diffusion methods

ABSTRACT AND MOTIVATION

Dithering is type of thresholding where grey or color image gets converted to cluster of binary pixels. It helps us reduce the set of required colors. The quantization in dithering results into blobs of colors if not diffused properly. Error diffusion is type of halftoning where quantization error is diffused in forward pixels that are yet to proceed. This is used extensively while converting into binary image. In this way, we produce better local average intensity and enhances edges. The distribution of error is different is different for various algorithms.



Q with threshold = 0.5

if $F(j,k) \geq T$ then $G(j,k)$ is 255 (white)

if $F(j,k) < T$ then $G(j,k)$ is 0 (black)

Figure 18 – Error Diffusion algorithm

APPROACH AND PROCEDURE

DEVELOPED ALGORITHM:

$$b(i, j) = \begin{cases} 255, & \text{if } \tilde{f}(i, j) > T \\ 0, & \text{otherwise} \end{cases}$$

$$\tilde{f}(i, j) = f(i, j) + \sum_{k, l} h(k, l) e(i - k, j - l)$$

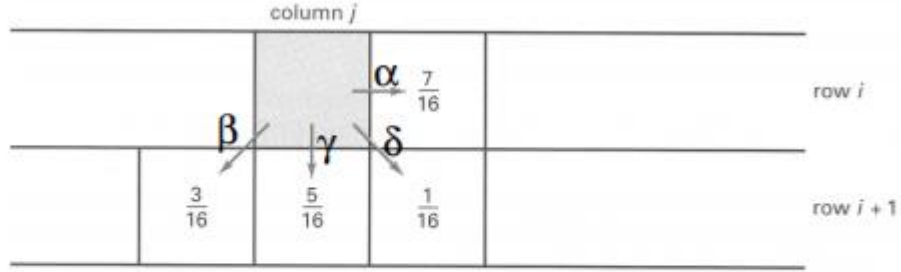
The diagram illustrates the padding process for a 3x3 convolution kernel. The original image is a 3x3 grid of values: 0, 0, 0; 0, 0, 7; 3, 5, 1. This is extended to a 5x5 grid by adding a border of 0s. The extended grid is labeled "boundary extension" and "original image". The diagram also shows "even row" and "odd row" padding for a 5x5 kernel, with green and orange arrows indicating the padding process.

23 | Page

STEP 2 – Initialize the buffer image $\tilde{f}(i,j) \leftarrow f(i,j)$ and computer binary value of pixel

$$b(i,j) = \begin{cases} 255, & \text{if } \tilde{f}(i,j) > T \\ 0, & \text{otherwise} \end{cases}$$

STEP 3 – Diffuse the error ahead using given algorithm-



$$\alpha + \beta + \gamma + \delta = 1.0$$

Figure 19 – Error Diffusion

STEP 4 – Save output binary image in output file.

Various diffusion matrices while scanning from left to right (odd row):

A. Floyd Steinberg's error diffusion matrix,

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

B. Jarvis, Judice and Ninke (JJN) error diffusion matrix,

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

C. Stucki's error diffusion matrix,

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

Various diffusion matrices while scanning from right to left (even row) for serpentine method-

A. Floyd Steinberg's error diffusion matrix,

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

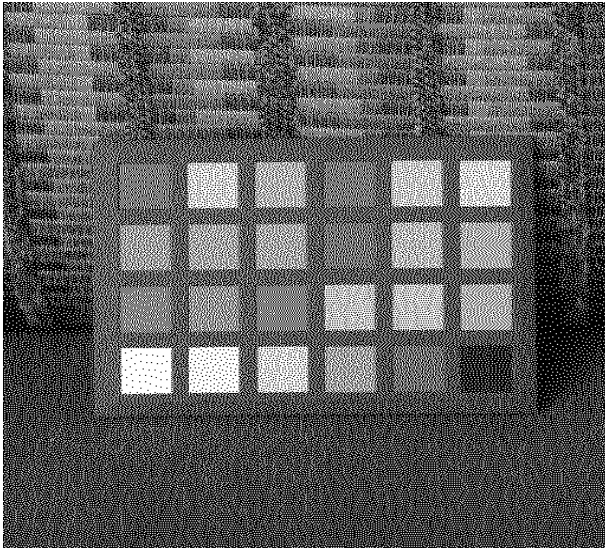
B. Jarvis, Judice and Ninke (JJN) error diffusion matrix,

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

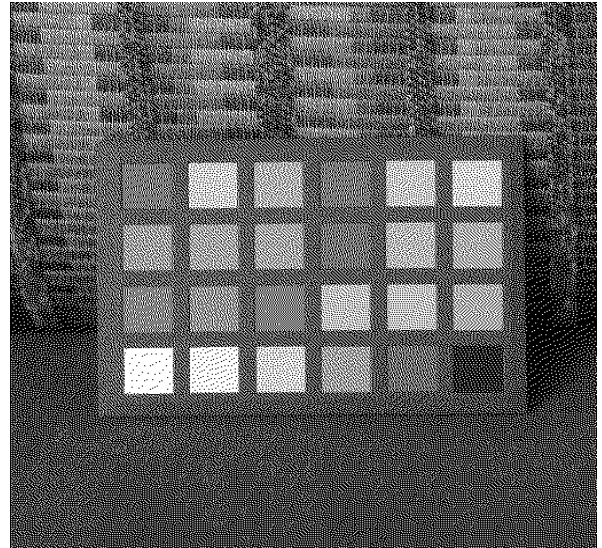
C. Stucki's error diffusion matrix,

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

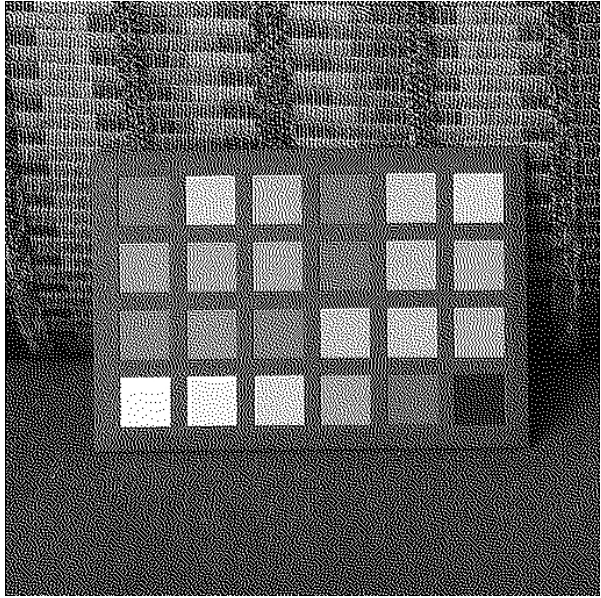
EXPERIMENTAL RESULTS:



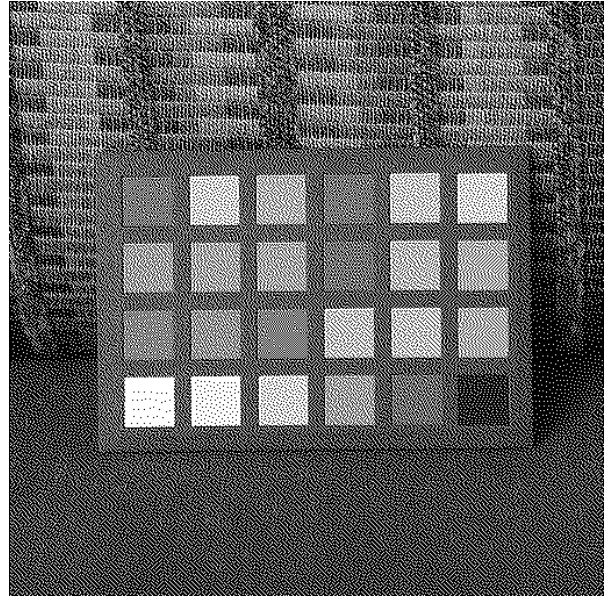
Using Floyd- Steinberg method(Serpentine)



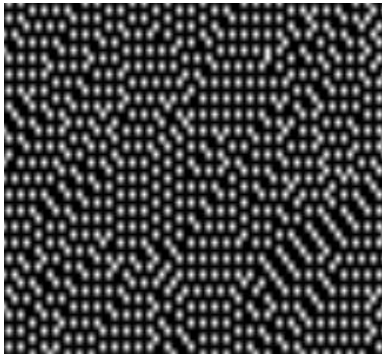
Using Floyd- Steinberg method(Normal)



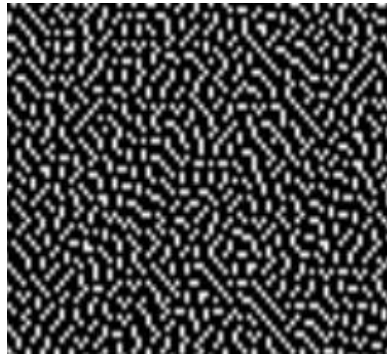
Using JNN method



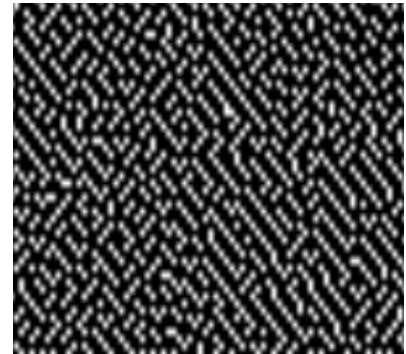
Using Stucki method



FS Method



JNN Method



Stucki Method

Figure 19 – Enlarged view of output

EXPERIMENTAL RESULTS:

Out of all half-toning methods, error diffusion is the best method as we can observe in output as well as it is smoother and does not induce any kind of pattern like dithering matrices.

Error Diffusion Method	PSNR
Floyd Steinberg Method	6.98799
JNN Method	7.1616
Stucki Method	7.12188

Table compares output of all methods. Floyd Steinberg provides low contrast than other 2 methods. JNN and Stucki's output shows much more detailing from the background. Error diffusion in FS method designed so that it creates checker pattern in output as observed from figure 19. Because of this, output does not look sharp and tends to provide lower contrast.

Performance of methods can be found out using visual inspection or using mathematical module like PSNR, MSE etc. From table, JNN and Stucki method better gives better output which can be inferred respective PSNR values. On visual inspection, JNN output looks sharper, describes the edges, sharpness without loss of information.

The burkes error diffusion matrix much better output by removing all lower frequency components and preserving blue noise by defined matrix is a novel way. The image displays more details than JNN method. The cascade inverse halftoning algorithm works especially way as it improves performance by cascading 2 stages, as image has patterns that is rich of edges and the cascade algorithms is designed to reserve edges.

Hence error diffusion is better digital halftoning practice as it preserves the details, and it does fixed patterns in the output image.

COLOR HALFTONING WITH ERROR DIFFUSION

Task: Convert *flower image* to half-toned image using separable error diffusion and MBVQ methods

ABSTRACT AND MOTIVATION

Error Diffusion is a high-performance halftoning method in which quantization errors are diffused to "future" pixels. Originally intended for grayscale images, it is traditionally extended to color images by Error-Diffusing each of the three-color planes independently (separable Error Diffusion) [3].

Color halftoning is process of separating image into different channel and using FS method to diffuse the error in the forward direction for each channel. Hence, we will have 8 colors as following

$$W = (0,0,0), Y = (0,0,1), C = (0,1,0), M = (1,0,0), \\ G = (0,1,1), R = (1,0,1), B = (1,1,0), K = (1,1,1)$$

In this approach, we will encounter 4 complementary pairs hence we need to have 8 colors available while halftoning any colored image.

The minimal brightness variation criterion states that RGB cube can be rendered using 8 basic colors. But in actual practice, we observe that most of the time, we can render any color with no more than 4 colors. Hence, we can approximate any color with 4 colors instead of 8. This will save up resources with different colors requiring different quadrants.

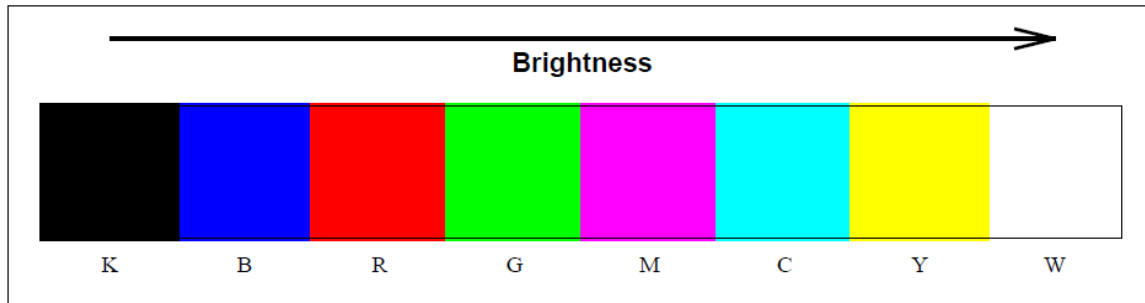


Figure 20 – Brightness scale of 8 basic colors

We can render every input color using one of six halftone quadruples: RGBK, WCMY, MYGC, RGMY, RGBM, or CMGB, each with obviously minimal brightness variation [3].

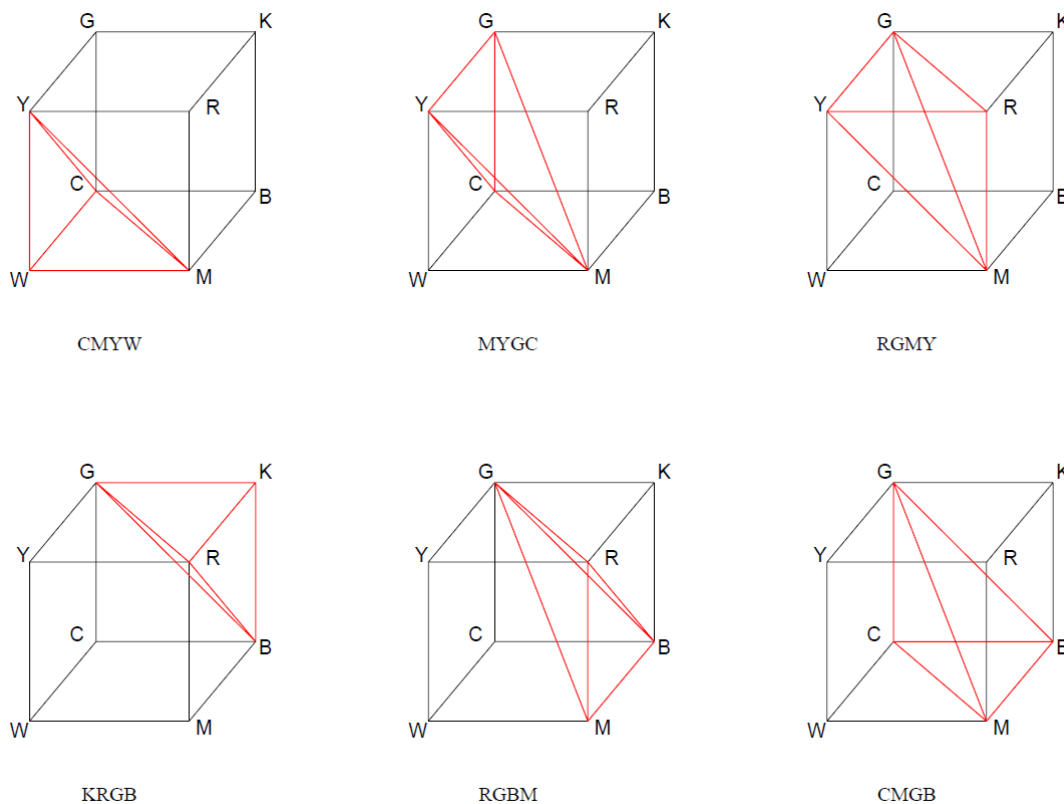


Figure 21 – The partition of RGB to six tetrahedral volumes

APPROACH AND PROCEDURES

The separable error diffusion algorithm works in following manner.

DEVELOPED ALGORITHM:

STEP 1 – Figure 18 can be elaborated as

$$b(i, j) = \begin{cases} 255, & \text{if } \tilde{f}(i, j) > T \\ 0, & \text{otherwise} \end{cases}$$

$$e(i, j) = \tilde{f}(i, j) - b(i, j)$$

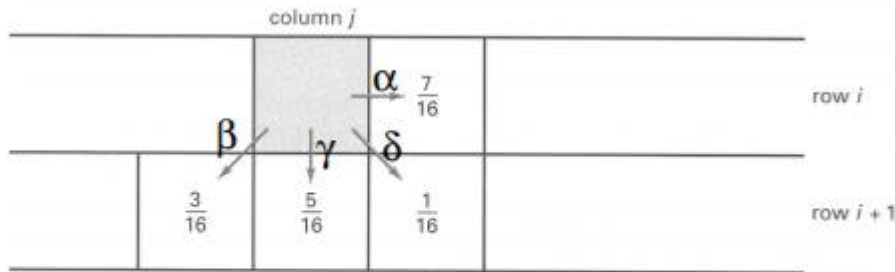
$$\tilde{f}(i, j) = f(i, j) + \sum_{k,l} h(k, l) e(i - k, j - l)$$

Where, $b(i, j)$ is binary output and T is threshold generally used as 127, $H(k, l)$ is error diffusion matrix whose sum of elements adds up to 1, $\tilde{f}(i, j)$ is buffer image, $f(i, j)$ is input image and $e(i, j)$ is error propagated forward. Error diffusion is generally performed on the pixels which are not yet processed.

STEP 2 - Initialize the buffer image $\tilde{f}(i, j) \leftarrow f(i, j)$ and computer binary value of pixel

$$b(i, j) = \begin{cases} 255, & \text{if } \tilde{f}(i, j) > T \\ 0, & \text{otherwise} \end{cases}$$

STEP 3 - Diffuse the error ahead using given algorithm-



$$\alpha + \beta + \gamma + \delta = 1.0$$

Figure 19 – Error Diffusion

STEP 4 – Save output binary image in output file.

STEP5 - Repeat the same procedure for each channel.

The MBVQ works in following manner.

DEVELOPED ALGORITHM:

STEP 1 - Find quadrant of input pixel using following function:

pyramid MBVQ(BYTE R, BYTE G, BYTE B)

{

if((R+G) > 255)

```

    if((G+B) > 255)
        if((R+G+B) > 510) return CMYW;
        else return MYGC;
    else return RGMY;
else
    if(!((G+B) > 255))
        if(!((R+G+B) > 255)) return KRGB;
        else return RGBM;
    else return CMGB;
}

```

STEP 2 - Find the vertex 'v' in MBVQ which is closest to $RGB(i; j) + e(i; j)$.

Where e is diffused error in forward manner.

STEP 3 - Compute the quantization error $RGB(i; j) + e(i; j) - v$.

Now diffuse the error in forward pixels. Diffusion is done using Floyd Steinberg algorithm.

EXPERIMENTAL RESULTS

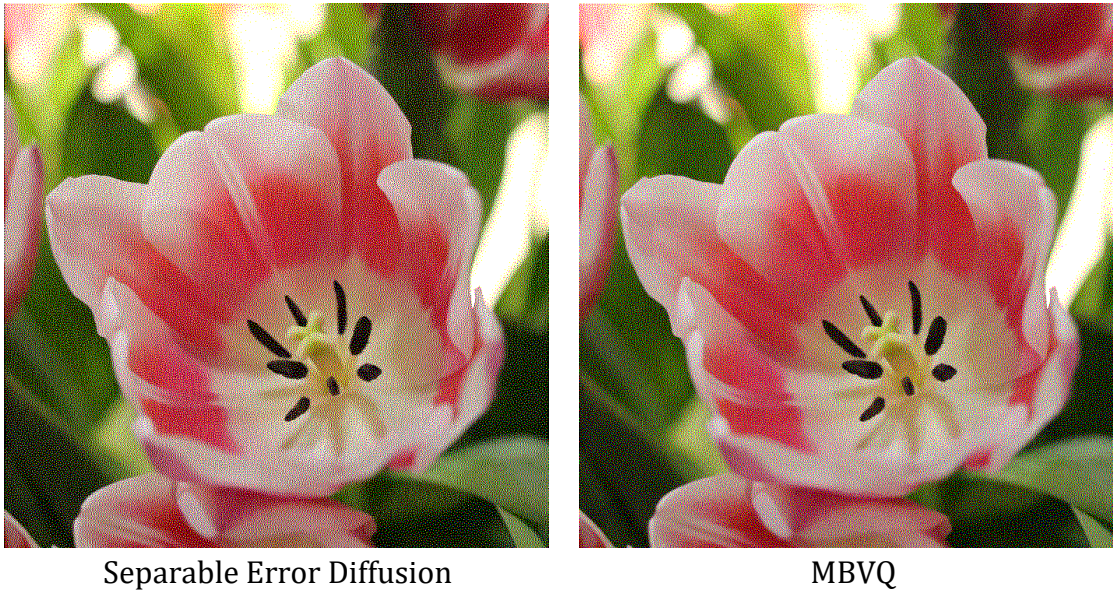
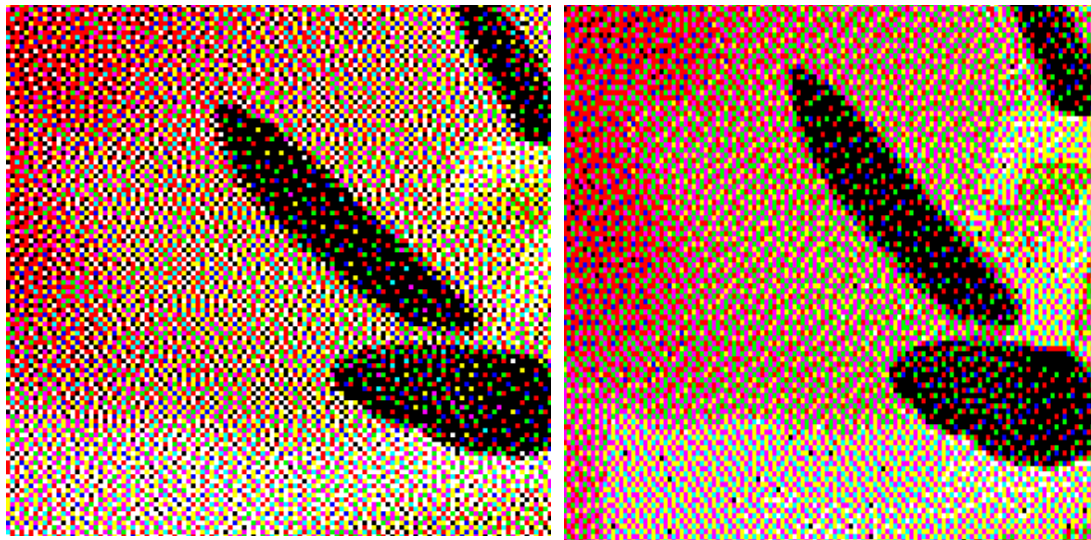


Figure 22 – The error diffusion output

Method	PSNR
Separable Error Diffusion	2.7514
MBVQ	2.7616



Separable Error Diffusion

MBVQ

Figure 23 -- The error diffusion enlarged output

DISCUSSION

As observed in figure 22 and 23, halftoning of output using Separable Error Diffusion and MBVQ has been carried out. Table indicated performance of algorithm using PSNR. Separable error diffusion does not exploit the inter color correlation, hence it leads to color artifacts and poor color rendition. For high sharpness and low artifacts, separable error diffusion does not converge and shows patchy or blurry output. To overcome its disadvantage, MBVQ has been introduced.

The main difference between both the methods is MBVQ looks for closest vertex in quadrant as oppose to closest of 8 vertices. While calculating closest vertex, norm is not specified hence might create an ambiguity there in MBVQ based on which norm we prefer while calculating closest vertex. Easiest to compute is the L2 tessellation, in which the decision planes inside the cube are valid for all R3 [4]. MBVQ is preferred over separable error diffusion as it reduces halftone noise as observed in figure 23. Regarding run time, MBVQ takes little more time than separable error diffusion as MBVQ has high pixel density, retains local averaging and reduces noise which results in more time.

PROBLEM 3 – MORPHOLOGICAL PROCESSING

SHRINKING

Task: Apply shrinking filters to stars image. Count the total number of stars and plot the histogram of star sizes with respect to frequency.

ABSTRACT AND MOTIVATION

Using morphological processing, we can process and extract information in image in spatial domain without taking color planes into consideration. Generally binary images are preferred but often contains imperfection and loss of information. Various processes tackle these losses and works in spatial domain and thus preserves form and structure of the image. They can be extended to grey images as a collection of non-linear operations related to the shape or morphology of features in an image. Morphological operations rely only on the relative ordering of pixel values, not on their numerical values. Here, structuring elements is traversed all over the image and compared with neighbouring pixels to find pattern.

Shrinking is process of reducing white pixels from the image and replacing those pixels with black pixels and thereby reducing white pixels over a region in such extent that white pixels in a localized area gets reduced to a single dot. Hence, we have to erode every black pixel such that any object without holed will erode to single pixel around its centre of mass.

APPROACH AND PROCEDURE

Shrinking of binary image with holes and without holes is given below. The basic logic is to apply unconditional and conditional masks in two stages.

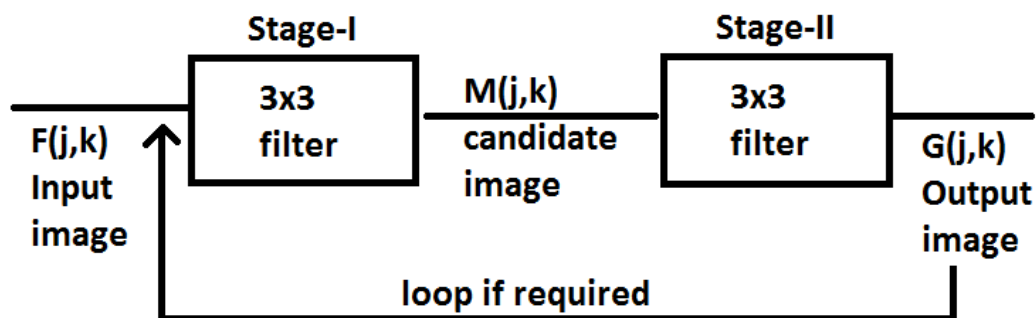


Figure 24 – The shrinking of binary image

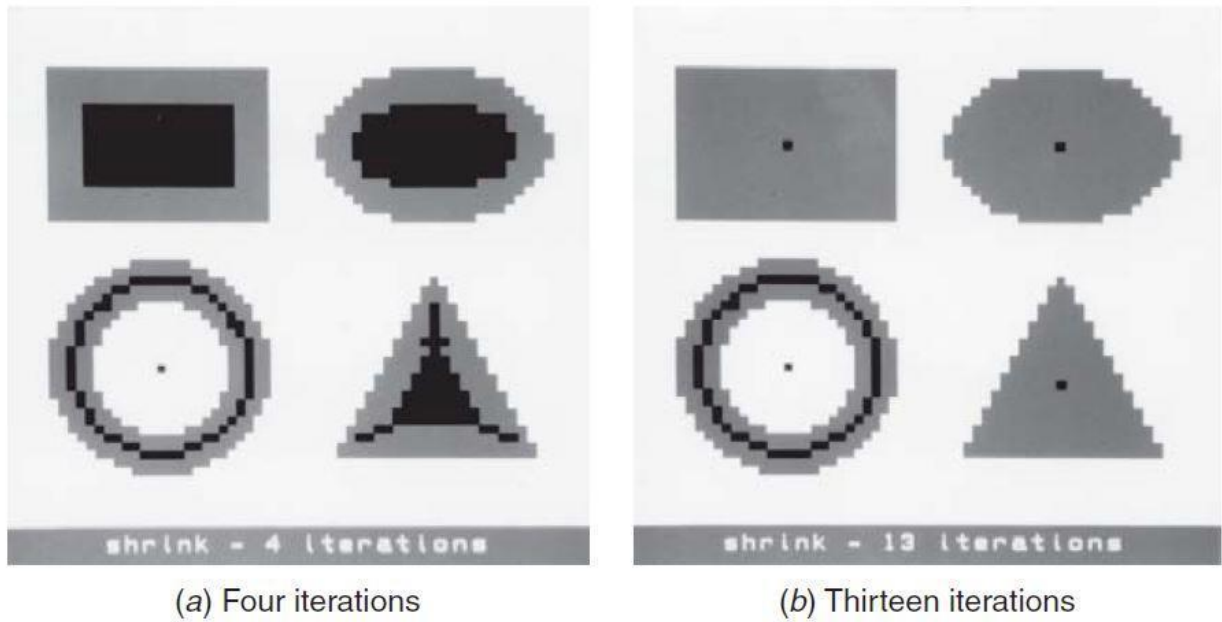


Figure 25 – Shrinking of binary image

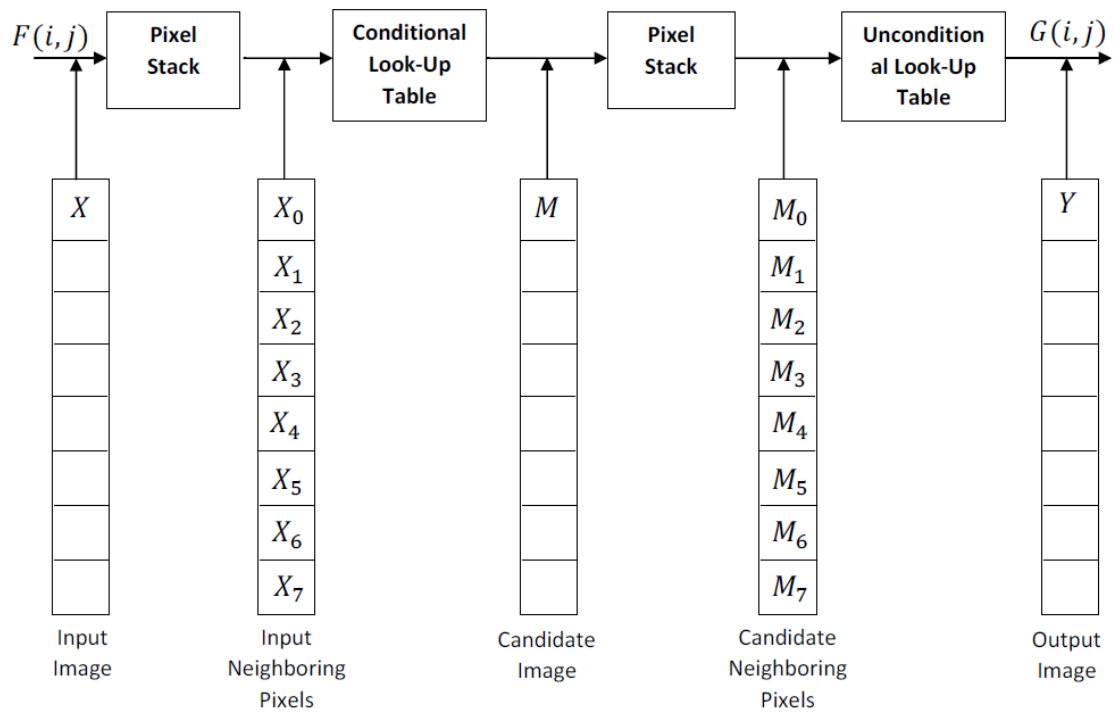


Figure 26 – Look-up table flowchart for binary conditional mark operations

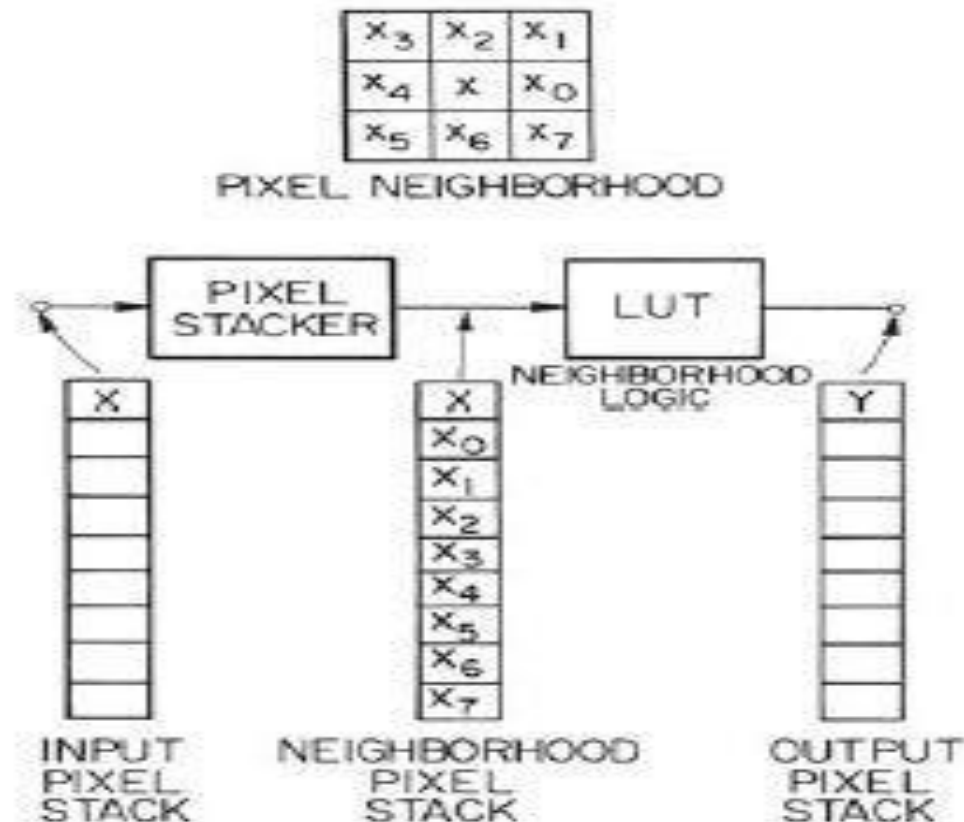


Figure 27 – Look-up table flowchart for binary unconditional mark operations

The shrinking operation can be expressed logically below.

$$G(j,k) = X \cap [M \cup P(M, M_0, \dots, M_7)]$$

Where, $P(M, M_0, \dots, M_7)$ = An erasure inhibiting logical variable

It is not possible to implement shrinking in a single-stage 3x3 pixel hit-or miss transformation. To get better output, we use 5x5 mask. But for that, we need 2^{25} filters. Hence, we will be using 2 3x3 filter hit-or-miss transformations. The convolution of this concatenation we result in a $2^6 - 1 \times 2^6 - 1$ mask, where $k = 3$, therefore the equivalent is a 5x5 filter.

DEVELOPED ALGORITHM:

STEP 1 – Make a table for conditional and unconditional masks for shrinking.

STEP 2 – If there is a hit as per conditional (LUT), reduce one-layer pixel of the white part to black else let be as it is and store

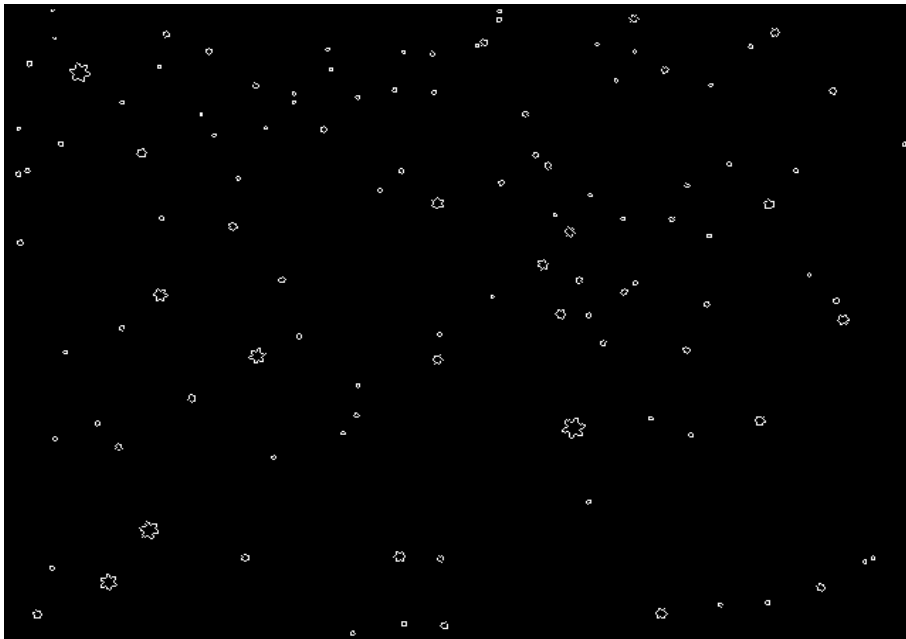
STEP 3 – Run nested loop across whole image. If there is a hit as per unconditional (LUT) copy the pixels else set it to black and array will indicate the 1st level shrunk image.

STEP 4 – Run this for 13 iterations until all your object shrinks to single pixel. Count number of white pixels surrounded by black neighbours will be number of stars.

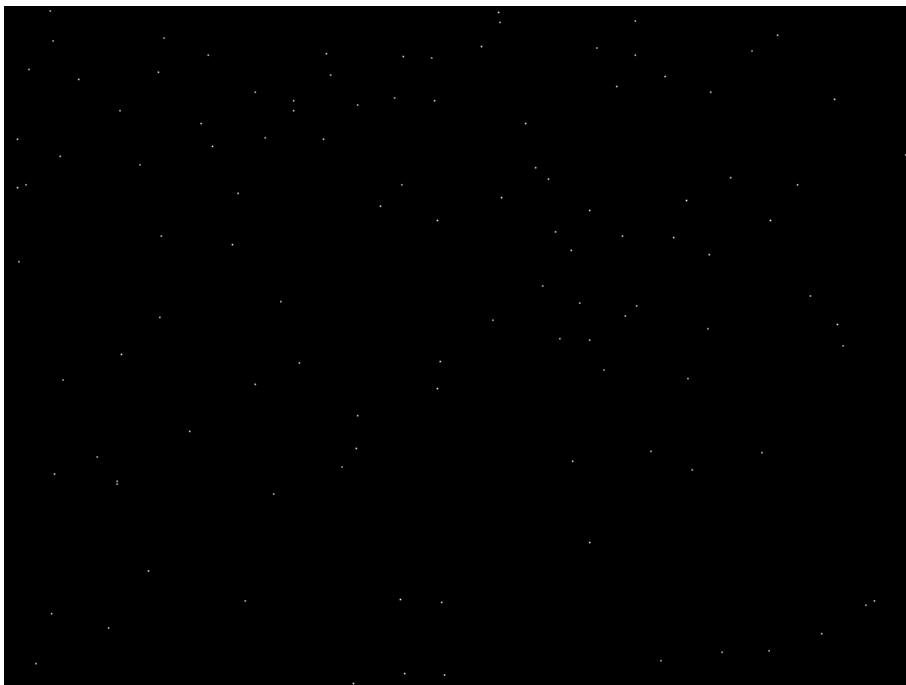
STEP 5 – To calculate number of stars of each size, check at every iteration, how many stars have been reduced to a single dot. On every iteration keep checking the number of

white dots and see how many iterations it took to reach there. The difference in number of white dots gives the frequency and the number of iterations it took to reach there defines the size.

EXPERIMENTAL RESULTS



Stage 1 output



Stage 2

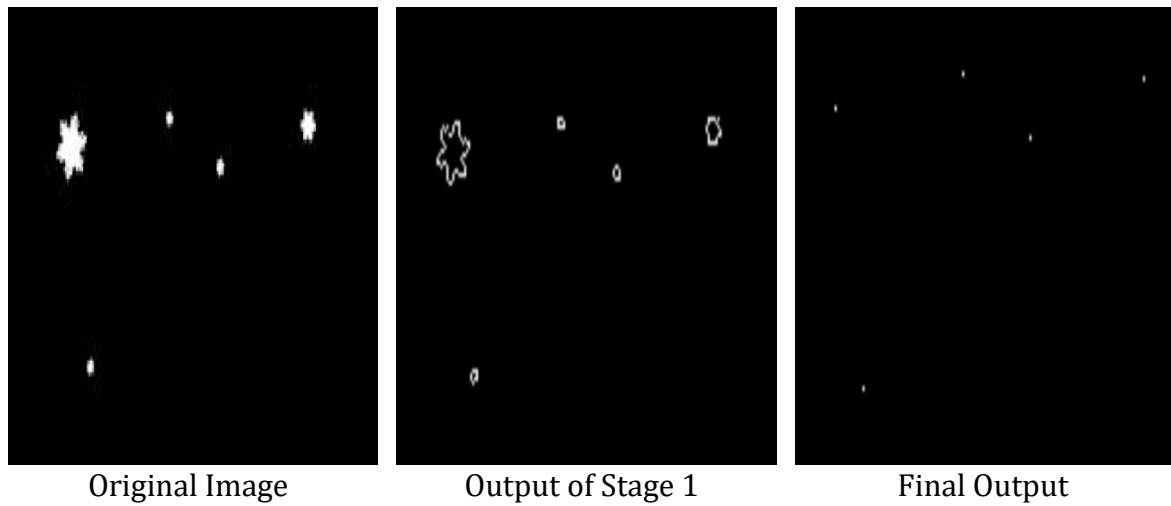


Figure 28 – Shrinking step by step Output

To find total number of stars and calculate stars of each size, I wrote another function to count dots at every iteration and find difference to star sizes.

```

"F:\Books\EE 569\Assignment2\q3shrinking\bin\Debug\q3shrinking.exe" stars.raw
File stars.raw has been read successfully!
File Star_grey.raw has been written successfully!
1289  1289  684  441  311  229  175  149  132  123  118  113  112
File stars_out.raw has been written successfully!

Number of Stars: 112
Star of size 1: 0
Star of size 2: 7
Star of size 3: 51
Star of size 4: 19
Star of size 5: 11
Star of size 6: 7
Star of size 7: 8
Star of size 8: 2
Star of size 9: 3
Star of size 10: 1
Star of size 11: 0
Star of size 12: 2
Star of size 13: 1
Star of size 14: 0

Process returned 0 (0x0)   execution time : 0.365 s
Press any key to continue.

```

Figure 29 – Total number of stars and star sizes

DISCUSSION

For shrinking, number of iterations is very important. Shrinking was done is no more than 13 iterations only as two rows and columns of white pixels were being erased at that time. To get desired output, we need to choose number of iterations. Here, I kept a counter for leftover pixels. For each iteration, I keep checking difference between counters and if that difference becomes zeros, then I stop at that iteration.

Total number of stars were found out to be 112. Its occurrence and different sizes are mentioned in the figure 27. However, while converting from grey to binary image using thresholding, two stars that are very close to each other will be connected in the binary image and then be shrunked to one pixel. Hence threshold value plays important role as we get different stars for different threshold value. To overcome this, we can erode the binary image first so that we will not lose small stars.

THINNING

Task: Apply thinning filters to Jigsaw image

ABSTRACT AND MOTIVATION

Thinning is process of reducing white pixels from the image and replacing those pixels with black pixels and thereby reducing white pixels over a region in such extent that white pixels in a localized area gets reduced to a single line. Hence, we have to erode every black pixel such that any object without holed will erode to single stroke around its centre of mass and object with holes will erode to minimally connected ring midway between hole and outer boundary.

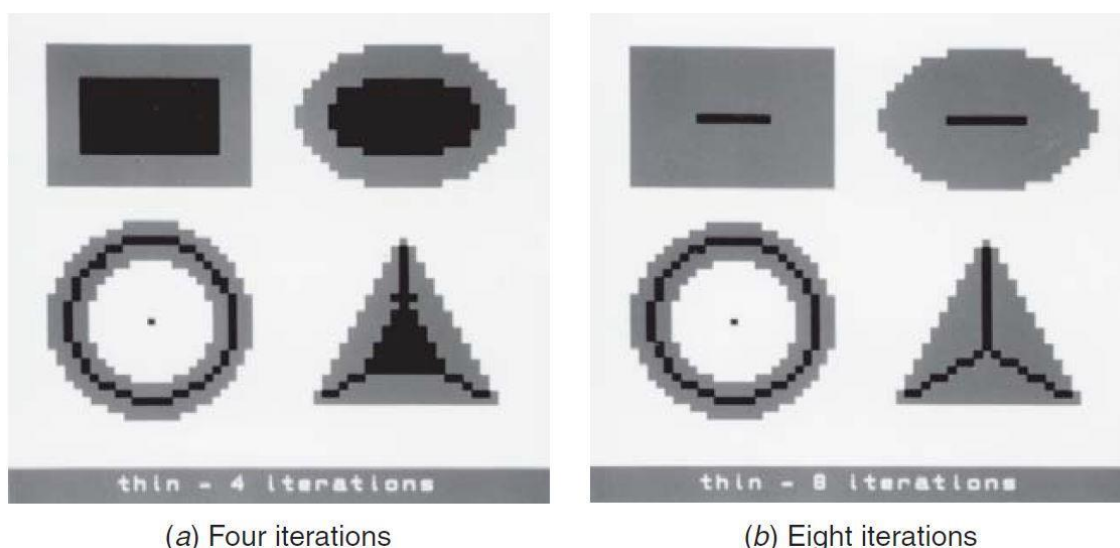


Figure 30 – Thinning of binary image

ABSTRACT AND MOTIVATION

For thinning of binary image, we follow the same procedure like shrinking only change would be different

DEVELOPED ALGORITHM:

STEP 1 – Make a table for conditional and unconditional masks for thinning.

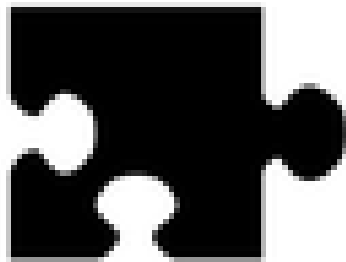
STEP 2 – If there is a hit as per conditional (LUT), reduce one-layer pixel of the white part to black else let be as it is and store

STEP 3 – Run nested loop across whole image. If there is a hit as per unconditional (LUT) copy the pixels else set it to black and array will indicate the 1st level shrunk image.

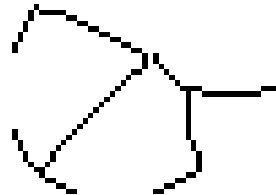
STEP 4 – Run this for many iterations until all your object shrinks to single stroke.

STEP 5 – Stop the iterations when your line will not get thinner or it breaks.

EXPERIMENTAL RESULTS

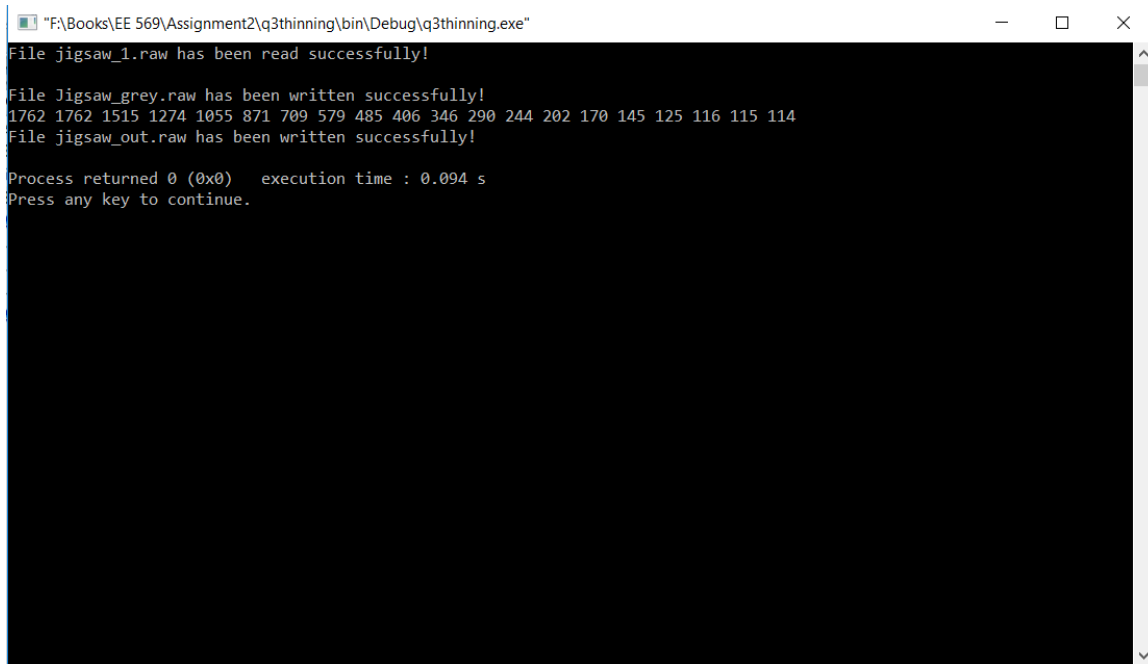


Input Image



Thinned Output Image

The output is as expected and it is a stroke line like structure of the given input image.



```
"F:\Books\EE 569\Assignment2\q3thinning\bin\Debug\q3thinning.exe"
File jigsaw_1.raw has been read successfully!
File jigsaw_grey.raw has been written successfully!
1762 1762 1515 1274 1055 871 709 579 485 406 346 290 244 202 170 145 125 116 115 114
File jigsaw_out.raw has been written successfully!
Process returned 0 (0x0) execution time : 0.094 s
Press any key to continue.
```

Figure 31 – Thinning remaining pixels

DISCUSSION

For thinning, I thresholded the image using threshold of 127. I used this binary image for thinning and depending upon number of remaining pixels, I started thinning the image. From figure 31, we can observe that it took me 19 iterations for thinning because after that, stroke line does not change hence number of remaining pixels will remain the same. We have to make sure that line does not break hence value of threshold can be changed if line is breaking. The advantage of this method is that when just one stroke of white pixels left and then even on doing any further iterations of thinning, it does not affect the image at all.

SKELETONIZATION

Task: Apply skeletonizing filters to Jigsaw image

ABSTRACT AND MOTIVATION

Skeletonizing is process of reducing white pixels from the image and replacing those pixels with black pixels and thereby reducing white pixels over a region in such extent that white pixels in a localized area gets reduced to a single rooted line. It is type of

thinning where we erase black pixels such that object without holes erodes to minimally connected rooted stroke located equidistant from its nearest outer boundaries.

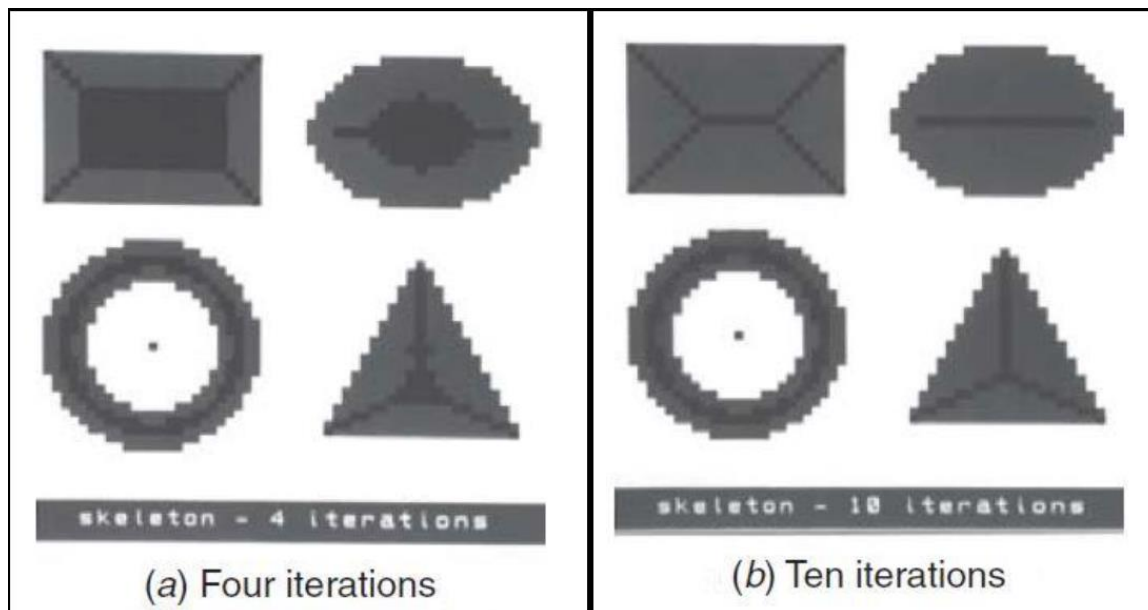


Figure 32 – Skeletonization of binary image

ABSTRACT AND MOTIVATION

For skeletonizing of binary image, we follow the same procedure as shrinking only conditional and unconditional patterns would be different

DEVELOPED ALGORITHM:

STEP 1 – Make a table for conditional and unconditional masks for skeletonization.

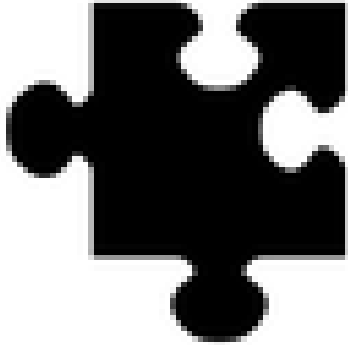
STEP 2 – If there is a hit as per conditional (LUT), reduce one-layer pixel of the white part to black else let be as it is and store

STEP 3 – Run nested loop across whole image. If there is a hit as per unconditional (LUT) copy the pixels else set it to black and array will indicate the 1st level skeletonized image.

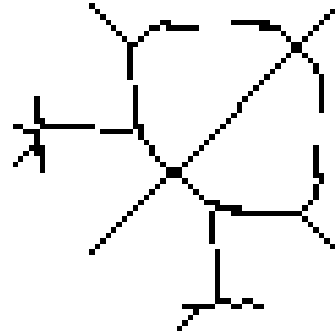
STEP 4 – Run this for many iterations until all your object shrinks to single root line.

STEP 5 – Stop the iterations when your line will not get thinner or it breaks

EXPERIMENTAL RESULTS



Input Image



Skeletonized Output Image

```
"F:\Books\EE 569\Assignment2\q3skeletonizing\bin\Debug\q3skeletonizing.exe"
File jigsaw_2.raw has been read successfully!
File jigsaw_grey.raw has been written successfully!
File jigsaw_binary_out.raw has been written successfully!
1882 1882 1560 1277 1025 840 684 551 477 415 359 318 286 260 246 238
File jigsaw_out.raw has been written successfully!
Process returned 0 (0x0)   execution time : 0.152 s
Press any key to continue.
```

Figure 33 – Skeletonization Output

DISCUSSION

For skeletonizing, output is similar to thinning but has more edges at sharp corners. Output image shows stick like structure. It took me 15 iterations to get output where stroke line does not break. Skeletonization is the "thinning" of a binary image or silhouette to a one-pixel width spine. The advantage of this method is that when just one

stroke of white pixels left and then even on doing any further iterations of skeletonizing, it does not affect the image at all.

COUNTING GAME

Task: Count the number of pieces in the board and count the number or unique pieces in the board.

ABSTRACT AND MOTIVATION

In this experiment, we are supposed to design an algorithm that uses morphological and logical operations to count number of pieces in the image. In next part, we have to design an algorithm to find unique pieces out of all available pieces. Multiple approaches can be considered for counting number of jigsaws and finding similarities between pieces depending upon geometrical modifications.

APPROACH AND PROCEDURE

DEVELOPED ALGORITHM:

STEP 1 – Binarize the image using thresholding. I have kept high value of 220 to ensure pixels do not break.

STEP 2 – Apply shrinking filter to board image. It will reduce to single dots. Count number of dots. The count will be number of jigsaws in image.

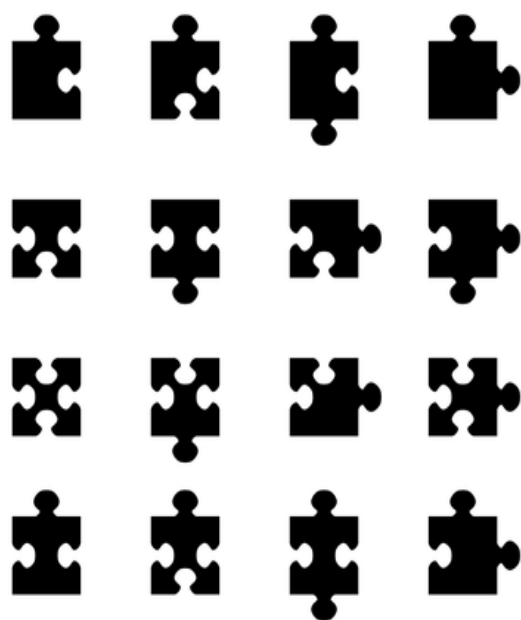
STEP 3 – Find the starting locations of all 16 jigsaws using patterns and save it in an array.

STEP 4 – Find pattern for every piece. I have assigned 0 for hole, 1 for flat line and 2 for head. Save this pattern in an array.

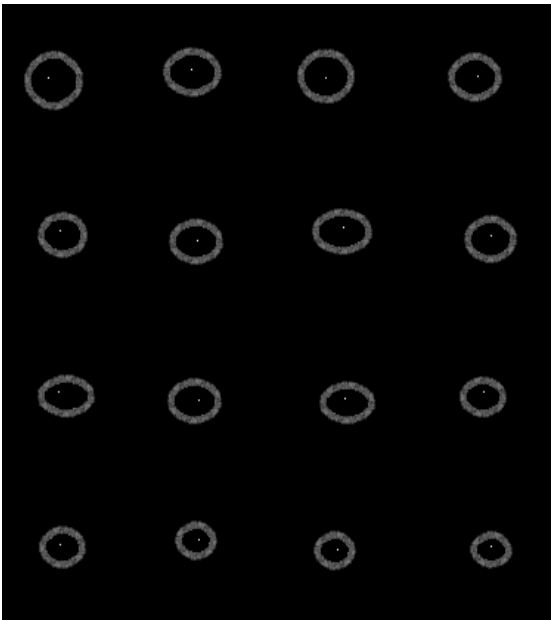
STEP 5 – Traverse through all patterns using rotation and flipping and compare the patterns with each other and decrement counter from 16 for every match.

STEP 6 – Final counter value will be unique jigsaw pieces.

EXPERIMENTAL RESULTS



Input Image



Number of jigsaws in image

```
"F:\Books\EE 569\Assignment2\countinggame\bin\Debug\countinggame.exe"
File board.raw has been read successfully!
File Star_grey.raw has been written successfully!
File Star_grey.raw has been written successfully!
30889 30889 26739 22760 19042 15649 12753 10243 8312 6994 5874 4918 4091 3370 2766
2244 1827 1595 1412 1261 1132 1022 919 819 730 647 576 517 463 417
367 320 277 239 206 174 142 113 87 66 49 35 23 18 16
File board_out.raw has been written successfully!
Number of Pieces: 16
22 16 22 61
22 110 22 155
22 204 22 249
22 298 22 343
116 16 116 61
116 110 116 155
116 204 116 249
116 298 116 343
210 16 210 61
210 110 210 155
210 204 210 249
210 298 210 343
304 16 304 61
304 110 304 155
304 204 304 249
304 298 304 343
Process returned 0 (0x0) execution time : 0.493 s
Press any key to continue.
```

Figure 33 – Starting locations and ending location of jigsaws

```
"F:\Books\EE 569\Assignment2\countinggame\bin\Debug\countinggame.exe"
File board.raw has been read successfully!
Number of Pieces: 16
Pattern for all pieces:
2 0 1 1
2 0 0 1
2 0 2 1
2 2 1 1
1 0 0 0
1 0 2 0
1 2 0 0
1 2 2 0
0 0 0 0
0 0 2 0
0 2 1 0
0 2 0 0
2 0 1 0
2 0 0 0
2 0 2 0
2 2 1 0
Match: 2 7
Match: 6 13
Match: 7 11
Match: 8 16
Match: 10 12
Match: 10 14
Unique pieces: 10

Process returned 0 (0x0)   execution time : 0.605 s
Press any key to continue.
```

Figure 34 – Pattern for each piece, matching pieces and unique pieces

DISCUSSION

For counting game, there are multiple approaches to find number of pieces. One being connected component analysis. This method compares its neighbors and assigns label with particular neighboring pixels. But the algorithm was very tedious. I found very better and efficient way to calculate jigsaw. I shrinked the image using different threshold by observing pixel intensities. Hence, I got 16 white pixels left in final iteration. Hence, I can conclude that there were 16 jigsaws.

Now, I assigned 0 for hole, 1 for level and 2 for head. Hence, I computer 16 patterns for each piece. Now, I compared every pattern with rest pattern to find matching patterns. After all iterations, I found out similar patterns hence I found out unique pieces.

REFERENCES

- [1] Analytical Methods for Squaring the Disc
- [2] Color Diffusion: Error-Diffusion for Color Halftones
- [3] Class Notes and assignments
- [4]<http://www.eie.polyu.edu.hk/~enyhchan/JEI%20-%20Blue%20noise%20digital%20color%20halftoning%20with%20multiscale%20error%20diffusion.pdf>