HOMEWORK SOLUTION 2

NAME: AKSHATA BHAT
USCID: 9560895350

INDEX

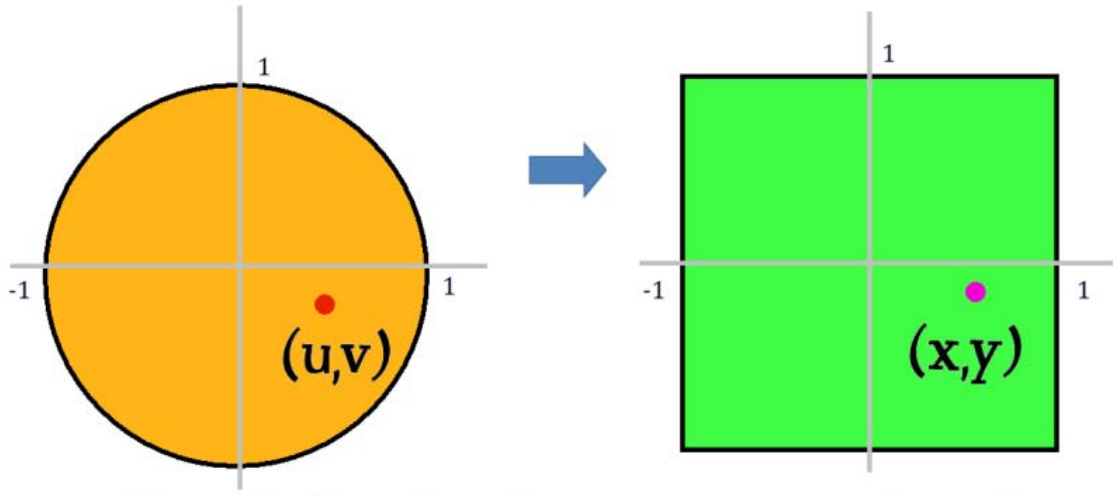
PROBLEM 1 - GEOMETRICAL TRANSFORMATIONS

1 A. GEOMETRICAL WARPING

Abstract and Motivation

Geometrical transformations are carried out in the image domain and they are generally performed my either moving the content from one location to another within the same image or by transformation of the coordinates. A square image can be represented in the form of a circular image or an elliptical one, in that case, a geometrical warping operation is carried out to map the image coordinates in the square image to a circular image.

In the current problem, we have transformed a square image to a disc image by using the following formulae for elliptical Grid mapping after transforming the pixel index values to lie in the range -1 to 1 :



domain: $\{(u,v) | u^2 + v^2 \leq 1\}$ range: [-1, 1] x [-1, 1] **[5]**

Disc to square mapping:

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

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

Square to disc mapping:

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

[5]

APPROACH AND PROCEDURE

1) Read the input color image into a 1D array
2) Separate the color channels into 3 two dimensional arrays of Red, Green and Blue pixel values.
3) Modify the pixel index locations to lie in the range of -1 to 1 in both the horizontal and vertical directions.
4) For example if the width of the image is 512, then the index values of the pixels will be in the range 0 to 511. We subtract each pixel index by 255 and divide them by 255 to get index values in the range -1 to 1
5) The index values of the square image are then transformed to get the mapping in the circular/disc image by using the formulae mentioned above.
6) This is repeated in both the horizontal and vertical directions to get the u,v values which map to the pixel locations in the disc image.
7) After obtaining the u,v values, pixels in those locations in the original square image are copied to the corresponding locations in the disc image and finally get the geometrically wrapped disc image.
8) Once the square to disc conversion has been completed, we try to reverse transform the disc to get the square.
9) Using the formula mentioned above to perform the disc to square transformation, we can obtain the square image.
10) In the final result, final result, the reverse mapping is completed by mapping the pixel coordinates from disc image to a square.

EXPERIMENTAL RESULTS:

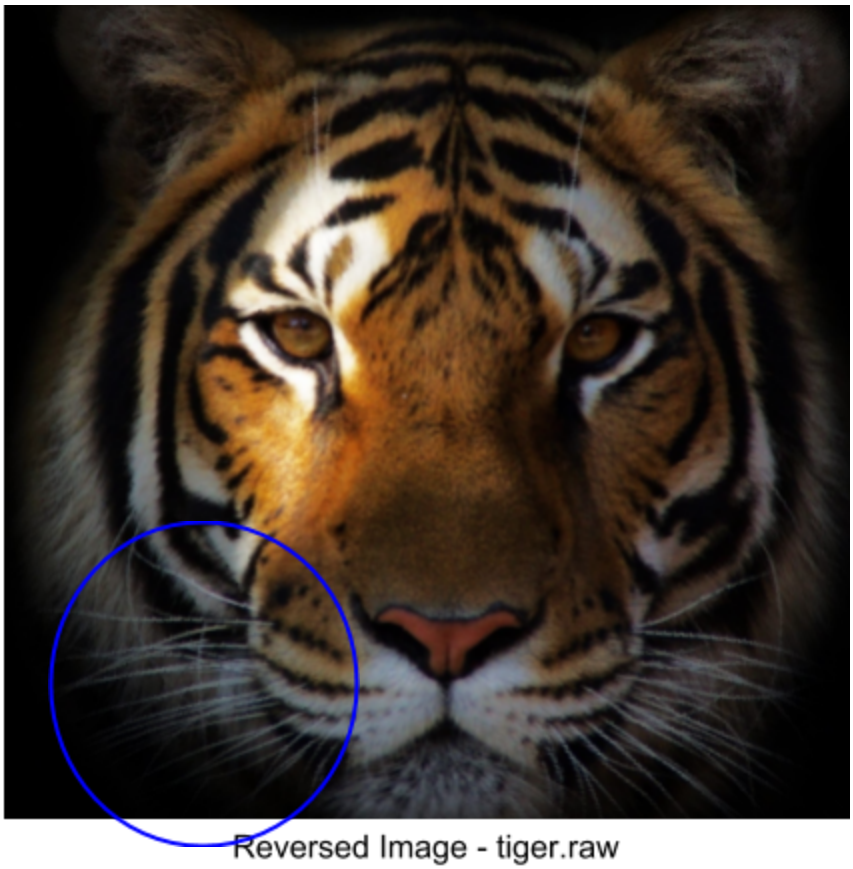
Original Image - panda.raw

Warped Image - panda.raw

Reversed Image - panda.raw

Original Image - tiger.raw

Warped Image - tiger.raw



Reversed Image - tiger.raw

Original Image - puppy.raw

Warped Image - puppy.raw

Reversed Image - puppy.raw

The above images (panda.raw, tiger.raw and the puppy.raw images have been converted from the square image to a disc shaped image, by performing geometrical warping, where the pixels in the original image are mapped to a different set of points in the output image, without any modifications made to the pixel intensity values. We see that for the 3 input images, during the square to disc conversion, multiple points from the square image may get mapped to a single point in the disc image, due to the floating point values being converted to integers during the warping operation. As a result of this, since multiple points from the square image have been mapped to the disc, during the reverse mapping from disc to square, the exact values of the square will not be obtained back, hence we might see some distortion in the reversed output images as highlighted above. In the puppy.raw image for example, the whiskers are not as fine and smooth as compared to the puppy.raw original image, they have jagged edges (notice the part within the circle). Similarly for the Reversed outputs of tiger.raw and panda.raw, we can observe the same things for the highlighted portions.

1 B. HOMOGRAPHIC TRANSFORMATION AND IMAGE STITCHING

ABSTRACT AND MOTIVATION

The concept of homographic transformation and image stitching comes into picture, when we want to capture an entire scene. In such a case, the modern day cameras take a sequence of shots and stitch them together to form one huge wide panoramic image. Panoramic photography, also known as wide format photography is the technique of stitching multiple images captured on a single camera placed at different angles. If the image is captured at different angles then we have to perform some operations on the image to make sure we can look at the image and information from it without having to tilt our heads in different directions. The homographic transformations take care of this, by performing transformations to the image, such that the output image conforms to our field of view.

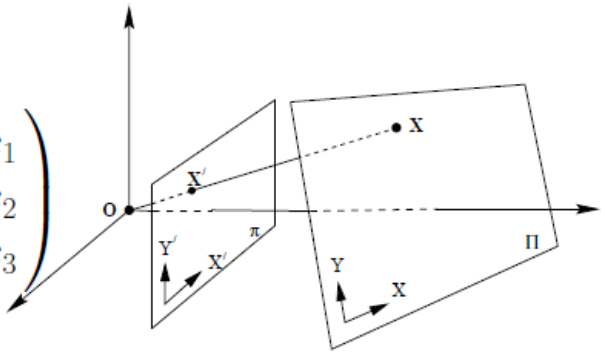
The panoramic output image is obtained by performing homographic transformation and image stitching. In homographic transformation, the image is either rotated, translated, projected, affine transformed to ensure the image from one plane maps to the desired plane of the image taken in thes straight direction. After the transformations are done, the transformed images are embedded in a bigger image or are stitched together to give the panoramic view.

APPROACH AND PROCEDURE

The problem we wish to solve here is to create a panorama using the given left.raw, right.raw and middle.raw images. We will perform the homographic transformation and the image stitching operations to get the output composite image.

In homographic transformation, we use a homographic matrix to transform the image into the desired form such that it can be stitched with the image captured along the straight field of view.

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$



$\mathbf{x}' = \mathbf{H}\mathbf{x}$, where $\mathbf{H}$ is a 3×3 non-singular homogeneous matrix.

The homographic transformation transforms the image from one plane to another using the above concept

$$x' = Hx$$

x → Coordinate Points in the image plane to be transformed
H → Homographic transformation matrix
x' → Coordinate Points in the Output image plane, the desired transformed image plane

Since we are projecting the points in 3D space onto a 2D space, we use the third quadrant to divide the x and y coordinates.

The homographic matrix is obtained by calculating 4 points in the left image and the middle image with which the left image has to be stitched, common to both the images and having the same intensity values. These 4 points are substituted into the set of linear equations, which are solved to get the homographic matrix.

$$H = \begin{bmatrix} u0 & v0 & 1 & 0 & 0 & 0 & -(u0*x0) & -(v0*x0) \\ 0 & 0 & 0 & u0 & v0 & 1 & -(u0*y0) & -(v0*y0) \\ u1 & v1 & 1 & 0 & 0 & 0 & -(u1*x1) & -(v1*x1) \\ 0 & 0 & 0 & u1 & v1 & 1 & -(u1*y1) & -(v1*y1) \\ u2 & v2 & 1 & 0 & 0 & 0 & -(u2*x2) & -(v2*x2) \\ 0 & 0 & 0 & u2 & v2 & 1 & -(u2*y2) & -(v2*y2) \\ u3 & v3 & 1 & 0 & 0 & 0 & -(u3*x3) & -(v3*x3) \\ 0 & 0 & 0 & u3 & v3 & 1 & -(u3*y3) & -(v3*y3) \end{bmatrix}$$

where
(u0,v0), (u1,v1), (u2,v2), (u3,v3) are the 4 coordinates in the left image to be transformed.
(x0,y0), (x1,y1), (x2,y2), (x3,y3) are the 4 coordinates in the middle image .
We then solve the above set of linear equations to get the values for the Homographic matric.

We then transform every point in the left image or the image to be transformed and stitched,by using the homographic transformation. One the homographically transformed points are obtained, the corresponding pixel intensities are mapped to the middle / bigger image which is already embedded in a bigger image. The above steps are repeated for the right image too and all the transformed images are stitched together through bilinear interpolation.

Procedure for homographic matrix generation in Matlab:

1) Read the left and the middle image into two matrices
2) Using the cpselect tool, select 4 spatially separated control points which have the same intensity values in both the images.

3) Get the coordinates of the 4 points in both the left image to be embedded and the middle image which has already been embedded in a bigger image.
4) Create the 8 set of linear equations which upon solving will give the Homographic transformation matrix (H) values.
5) Compute the inverse of the H matrix obtained above
6) Export the H matrix from Matlab to C++

Procedure for homographic transformation and image stitching in C++:

1) Read the left.raw, right.raw and the middle.raw images into separate 2D arrays with the R, G, B channels separated.
2) First embed the middle image into a bigger empty image of size 4 times the width of the middle image(which has the same dimensions as the left and right images) and the height to be 2 times the height of the middle image.
3) Send the middle image embedded into a bigger image and the left image to Matlab to pick out the control points for the H matrix generation
4) Once the H matrix is obtained from Matlab, we need to perform transformations of every pixel in the left.raw image, such that it can be written to the right locations in the image
5) Traverse through the entire image and for every input pixel coordinate in the 2D plane from the left image, take the value as [i j 1], forming a 3 dimensional point.
6) Multiply every point with the H matrix to get the corresponding transformed points in the three dimensional space.
7) Since our transformed left image is to be embedded into a two dimensional image, we divide the x and y coordinate values by the weight of the z-coordinate.
8) We then map the corresponding pixel intensities from the left image to the middle image, one pixel after another.
9) At the end of the step 8) we will be done with transforming and stitching the left image in the middle image.
10) The above steps are repeated to embed the right image too in the middle image.
11) We will have to perform averaging with the middle image to get a uniformly embedded panorama image and possibly embed the middle image again to a uniform and good looking image.

EXPERIMENTAL RESULTS:

We see that the left, right and middle raw images can be stitched together to form the following panorama image. The control points were selected by using Matlab's cpselect tool by considering points which were common to both the images and had the same intensity. The points were taken such that they were spatially distant from each other forming a quadrilateral similar to a trapezium. If more points are chosen we might get a better matched image, but the computational complexity will increase drastically.

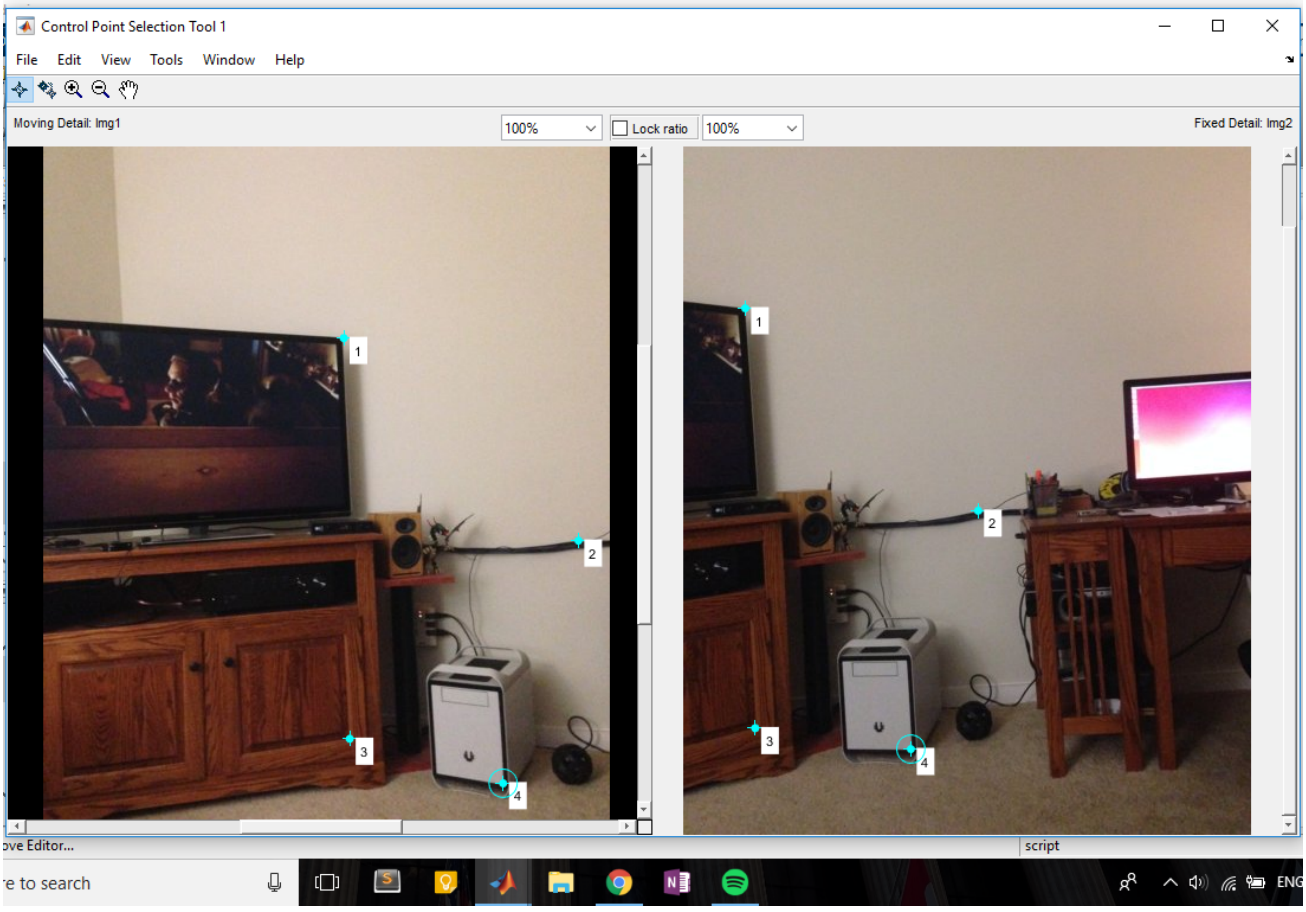
Right image control points :

u0 = 52.9375000000000 v0 = 196.062500000000
u1 = 249.812500000000 v1 = 366.937500000000
u2 = 61.3750000000000 v2 = 550.375000000000
u3 = 192.937500000000 v3 = 568.062500000000

Middle Image control points

x0 = 975.062500000000 y0 = 522.187500000000
x1 = 1173.81250000000 y1 = 693.187500000000
x2 = 980.125000000000 y2 = 860.375000000000
x3 = 1110.06250000000 y3 = 897.937500000000

Middle and Right images with the control points (Matlab cpselect)



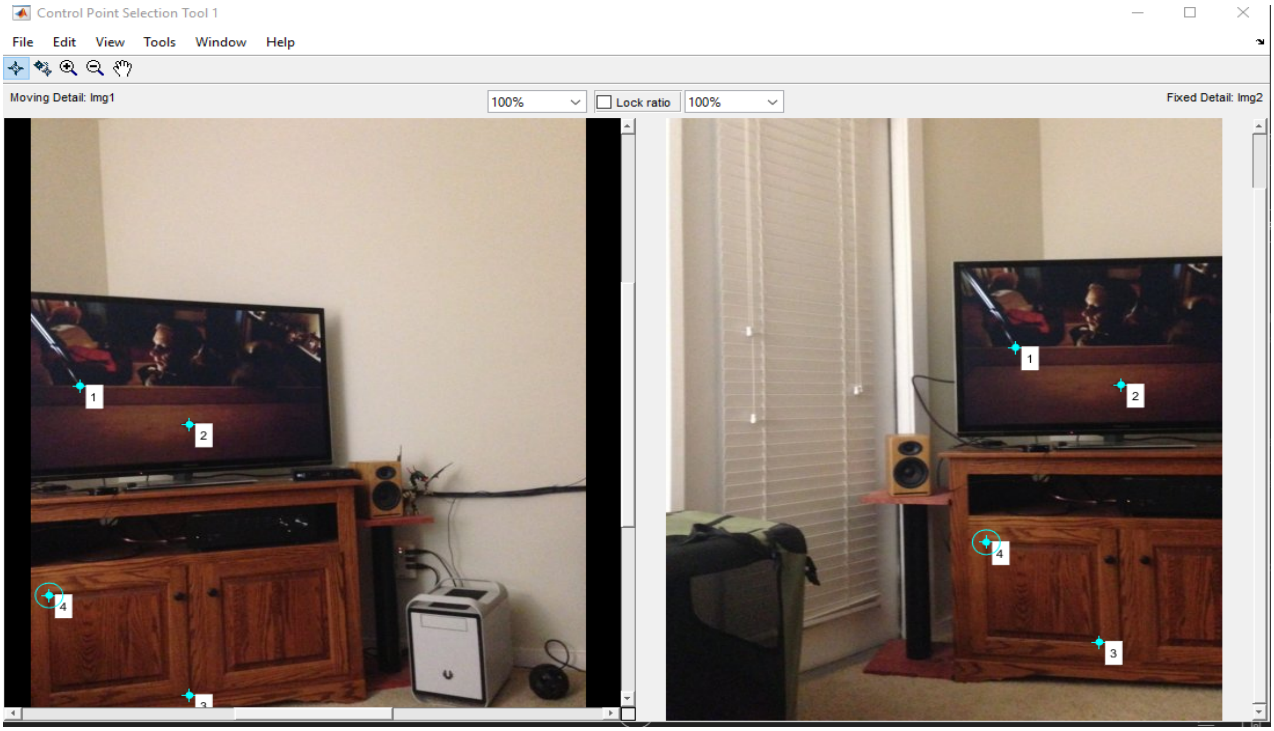
Left control points

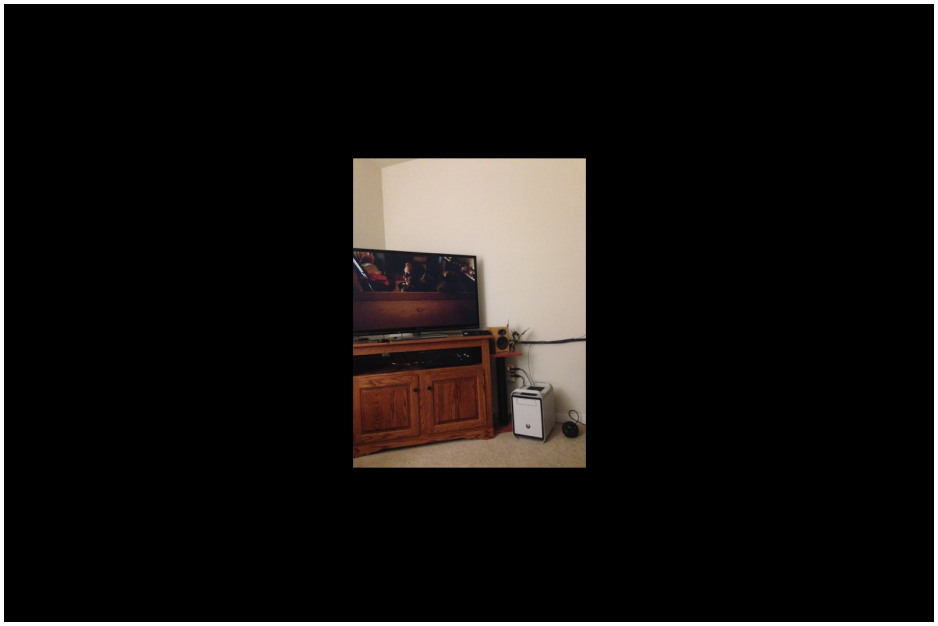
u0 = 302.125000000000 v0 = 280.625000000000
u1 = 393.125000000000 v1 = 316.625000000000
u2 = 374.375000000000 v2 = 564.625000000000
u3 = 276.875000000000 v3 = 468.375000000000

Middle control points

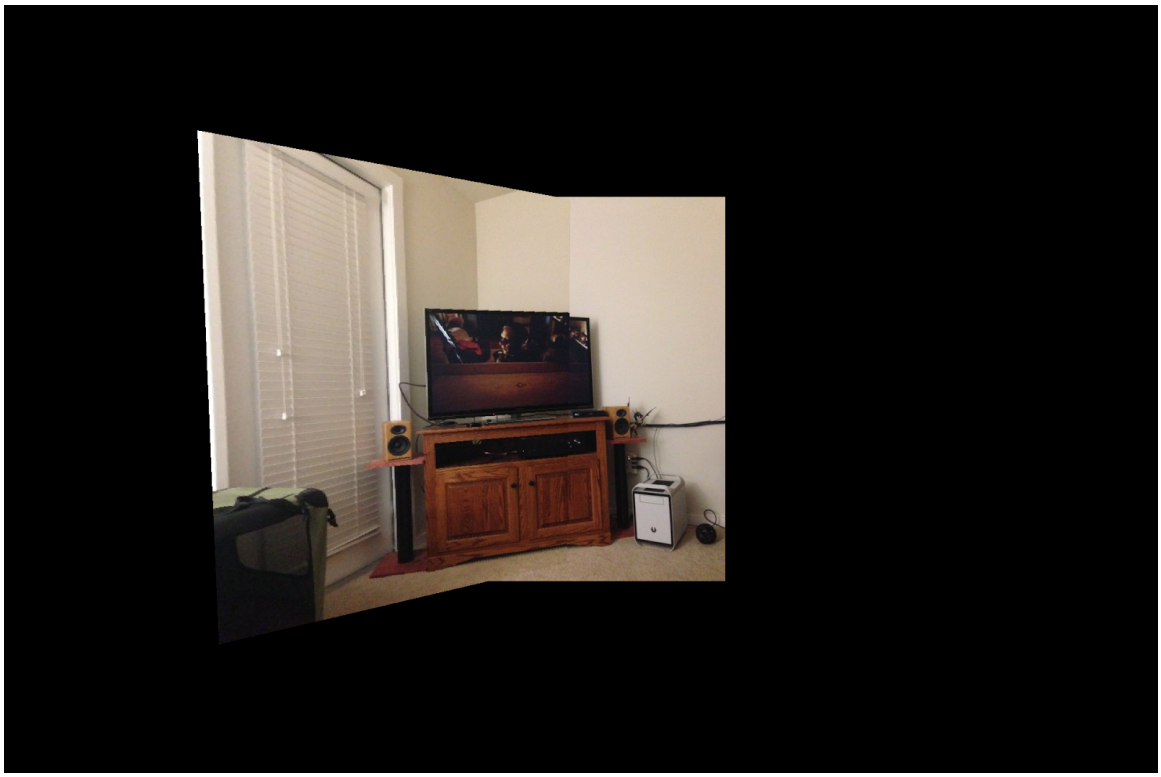
x0 = 763.125000000000 y0 = 596.625000000000
x1 = 857.625000000000 y1 = 633.875000000000
x2 = 857.625000000000 y2 = 894.875000000000
x3 = 736.375000000000 y3 = 798.625000000000

Middle and left image with the chosen control points

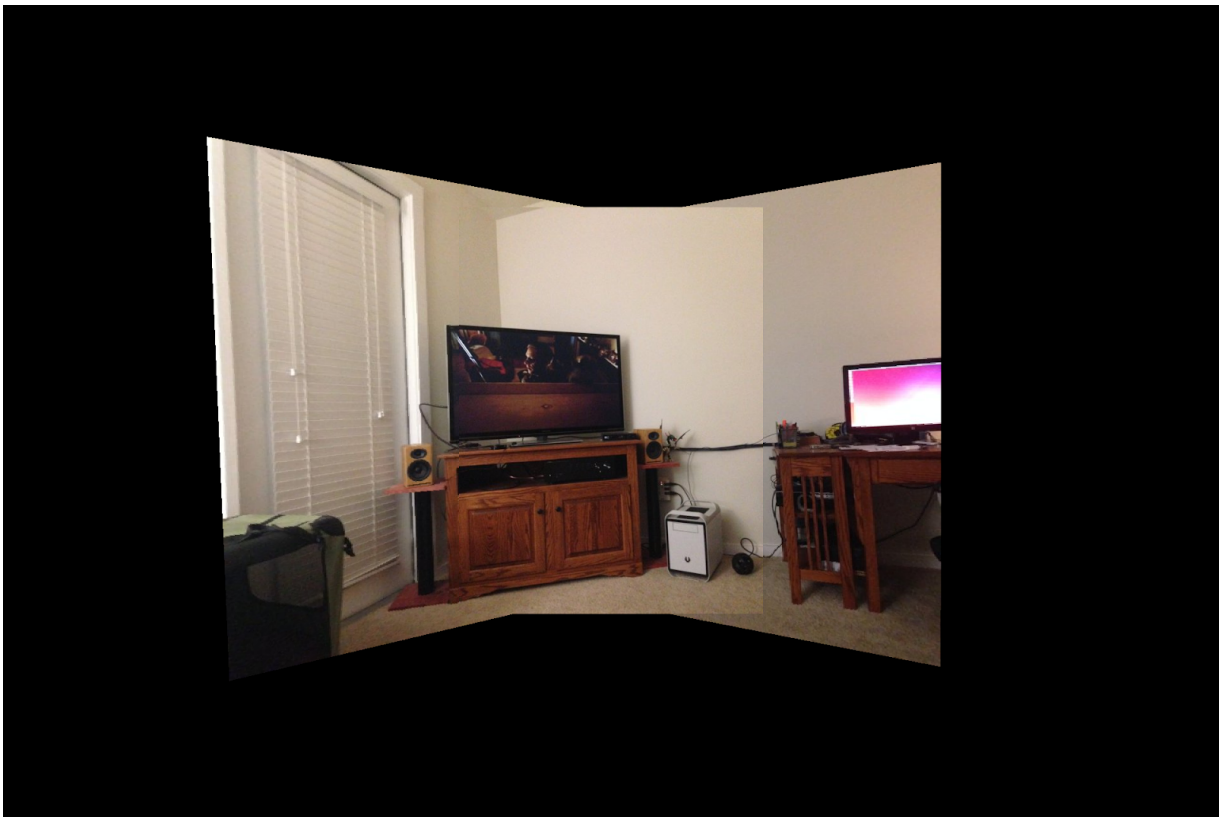




Embedded middle.raw image



Left image embedded in the middle image



Final Panorama output

PROBLEM 2 : DIGITAL HALFTONING

2a. DITHERING
MOTIVATION AND ABSTRACT:

Halftoning is the process of simulating the gray shades in an image by varying the size and distribution of black dots in an image for the purpose of printing or displaying on the screen or monitor. Halftoning mainly originated from the publishing industry and is used in printers too. Our eyes perceive an image by blending the fine details and recording the overall intensity. The concept of halftoning takes advantage of this. Digital halftoning, used while displaying an image, can be performed by dithering or error diffusion.

When we want to display an image on a monitor or a screen where lesser number of colors are supported, when compared to the colors in the image, we may run into problems. One of the main problems is losing details in the image and important details in the image might get replaced by huge blobs of color leaving the image almost unrecognizable. If you try to display a color image on a monitor which displays only black and white images, without doing any preprocessing, all one might see is blobs of black and white, where majority of the information in the image is lost. This is where the concept of dithering comes into picture.

Image Dithering is the process of adding noise intentionally to the image in order to replicate the unavailable colors in the image using available colors by distributing the quantization error. There are many methods to perform dithering. We shall talk about Fixed thresholding, random thresholding and the method of using Bayer dithering matrices.

APPROACH AND PROCEDURES

Fixed thresholding
In the fixed thresholding, every pixel in the grayscale input image is compared against a fixed threshold, and set to ‘0’ (black) if the pixel intensity is less than the threshold and is set to 255 if the pixel intensity is greater than the threshold. We take the threshold to be 127 in this problem.

Random thresholding
In this case, the threshold value against which each pixel is to be compared, is generated randomly, so that there is some dithering effect which can be performed on the image instead of generating a monotone image, like in the fixed thresholding case.

Dithering Matrix
The method of using an certain matrix to create a dithering effect in the image is called ordered dither. In this method, a precalculated index matrix introduced by Bayer is used to created the Dithered effect in the image. Here the Dithering matrices have sizes in the orders of two and are recursively calculated using the formula :

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

$$I_2n(i,j) = \begin{bmatrix} 4 * I_n(x,y) + 1 & 4 * I_n(x,y) + 2 \\ 4 * I_n(x,y) + 3 & 4 * I_n(x,y) + 0 \end{bmatrix}$$

The values in the index matrix indicate how likely it is that a dot will get turned on - 0 is says the pixel is most likely and 3, the least likely. The Index matrix is then transformed into a threshold matrix T, by normalizing the pixel intensities by bringing them into the range 0 to 1 using the formula:

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

This threshold matrix is then traversed through the entire image, and the pixel intensities are set to 1 if the pixel value is greater than the corresponding value in the Threshold matrix**.**

PROCEDURE:

Fixed thresholding

1) Read the grayscale input image into a 1D array.
2) Set the threshold to 127
3) Compare each pixel intensity with the threshold. Set the pixel intensity to 0 if the pixel value is less than the threshold and to 255 if the pixel intensity if greater than or equal to the threshold.
4) Write the resultant 1D output array into a file.

Random thresholding

1) Read the grayscale image into a 1D array
2) Use the inbuilt rand function from C to generate numbers randomly in the range 0 to 255. Ensure the seed for the random number generator is set to the system time, so that new set of numbers are generated every time time the function is called.
3) Compare each pixel intensity with the random threshold generated for each pixel value. Set the pixel intensity to 0 if the pixel value is less than the threshold and to 255 if the pixel intensity if greater than or equal to the threshold.
4) Write the resultant 1D output array into a file.

Dithering matrix

1) Read the grayscale image into a 1D array
2) Create 2x2 matrix to store the size 2 Index matrix and set the elements 1, 2, 3, 0 starting at location 0,0 to 1,1
3) Generate the 4x4 and 8x8 sized Index matrices by recursively using the formula $I_{2n}(i,j)$ formula mentioned above.

4) Notice that the 4x4 Index matrix is required to generate the 8x8 Index matrix and to generate the 4x4 index matrix the 2x2 Index matrix is required.
5) Generate the 2x2, 4x4 and 8x8 Thresholding matrices by using the T(x,y) formula mentioned above where N is the size of the index matrix - 2 or 4 or 8.
6) Normalize the input image intensities to lie between 0 to 1 instead of 0 to 255, by dividing each pixel intensity value by 255.
7) For each set of index and threshold matrix, traverse through the entire image and for each pixel intensity within that NxN (N=2 or 4 or 8) block, compare the normalized pixel intensity with the corresponding threshold matrix value.
8) Set the pixel intensity to 1 if the image intensity is greater than the corresponding threshold value, else set it to 0
9) Repeat the steps 7) and 8) for each of the thresholding matrices (2x2, 4x4 and 8x8).
10) Convert each of the output image intensities back into the range 0 to 255.
11) Write the output image 1D arrays back into a file and save it.

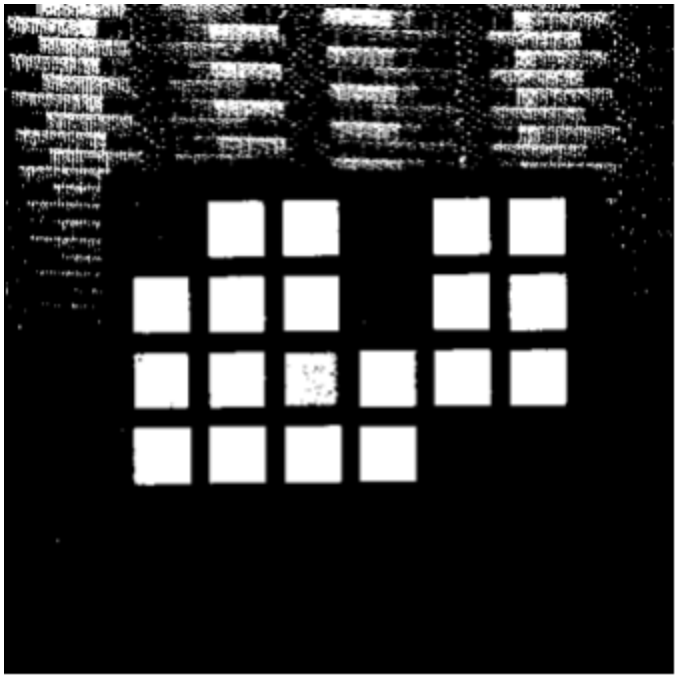
Generate display-ready image with 4 intensity levels

1) Read the grayscale image into a 1D array
2) Sort the index values of the pixel intensities in the increasing order of pixel intensity, such that the sorted address array contains index values of 0 intensity pixel values at the beginning and the entries towards the end correspond to the pixel intensities in the range 254, 255.
3) Next divide the sorted address array into 4 bins or equal parts with each part having the same number of pixel addresses.
4) Assign pixel intensity - 0 to the pixels in the first bin, 85 to the pixels in the second bin, 170 to the ones in the third and 255 to the fourth bin.
5) Write the resultant new pixel intensity values into a file to generate a display-ready image with 4 intensity levels only

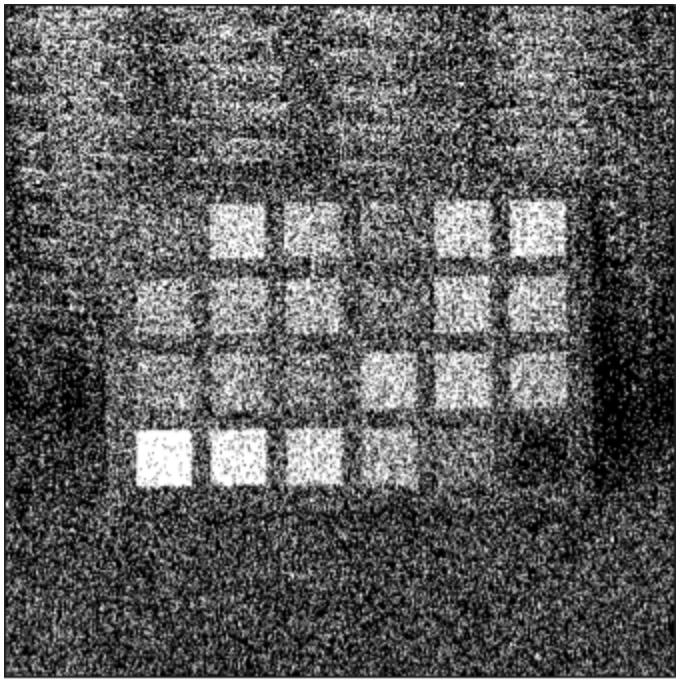
EXPERIMENTAL RESULTS:

We see that the output images generated by the fixed thresholding and random thresholding methods, are very similar and do not convey much information to the viewer due to the blobs of black and white color.

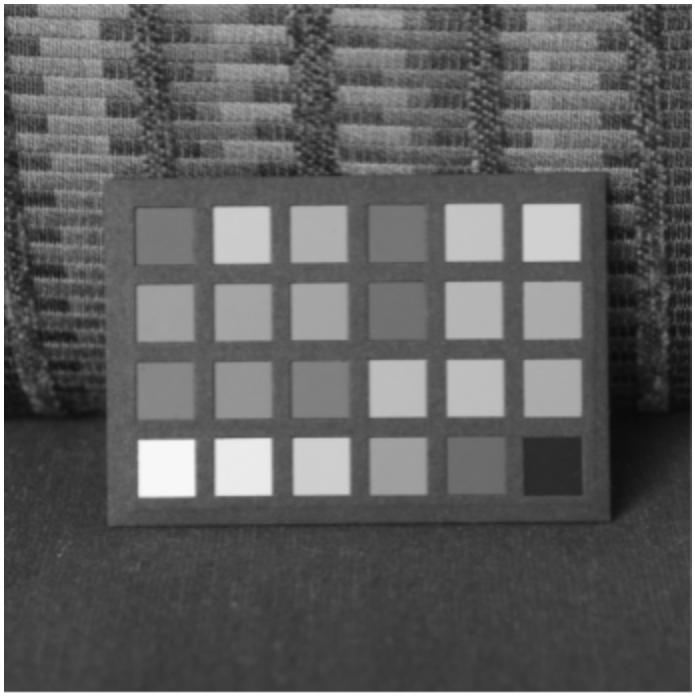
The output of the $I_2(i,j)$ thresholding matrix produces a better output compared to the fixed thresholding and random thresholding methods, and gives the perception of grayscale image. The $I_4(i,j)$ and $I_8(i,j)$ thresholding matrices give better outputs when compared to the rest and almost make us forget that the image has only 2 intensity values and not 255 values. They do have some noise due to the dithering, but both result in an intelligible and informative halftone image.



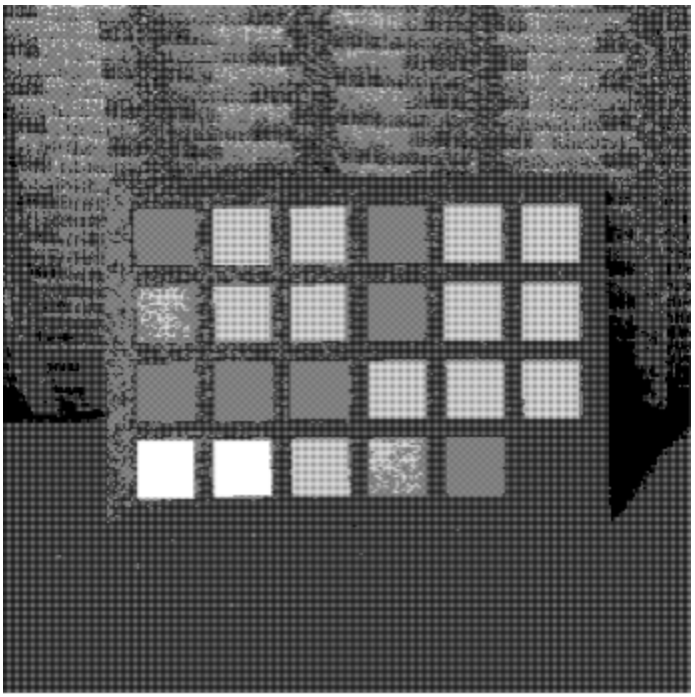
Fixed thresholding output



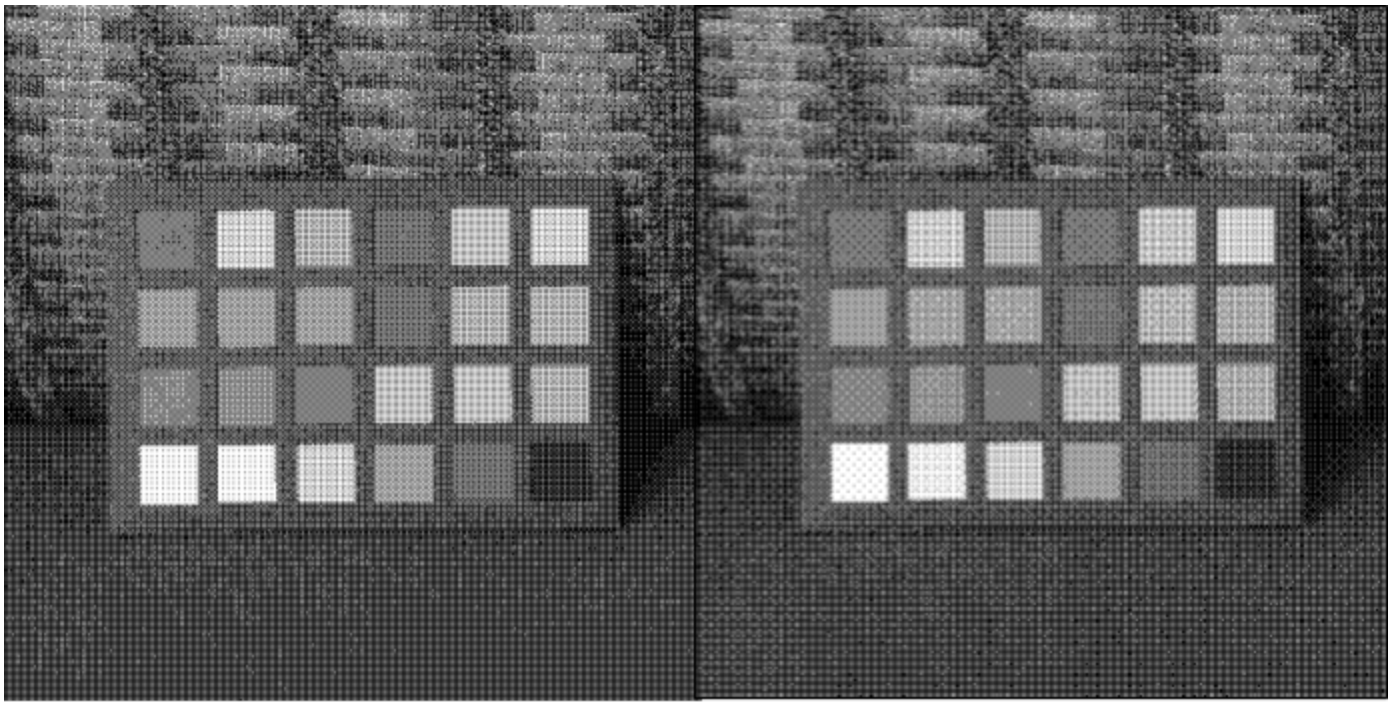
Random thresholding



Original Image - checkerboard.raw

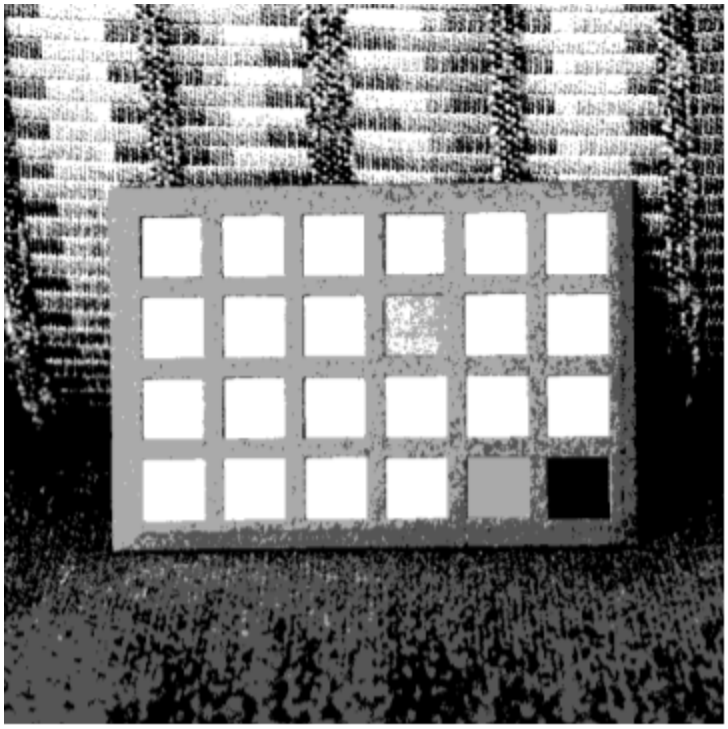


Output of 2x2 Dithering matrix



Output of 4x4 Dithering matrix

Output of 8x8 Dithering matrix



Display ready image with only 4 color levels

2b) ERROR DIFFUSION

MOTIVATION AND ABSTRACT

Another way of creating halftone images is by using the method of error diffusion. When we are converting a grayscale image into a black and white image or a halftone image, we are pushing each pixel intensity value to either 0 or 255. In the process, we are losing the quantization error value, i.e if a pixel with intensity of 123 is set to 0, we are losing an intensity value of 123. Hence we have a sequence of black pixels, if all the pixel intensities in the range 100 - 127 in a certain neighbourhood are set to 0.

In order to overcome this issue, we use the method of error diffusion where we are diffusing the quantization error to the surrounding pixel values, pixels which follow the target pixel being quantized are receiving the quantization error in proportions decided by the Error Diffusion method used. Generally 2 dimensional error diffusion method is used instead one dimension only, since this method will result in a pattern being formed in the image due the quantization errors being diffused.

We discuss the methods of Error Diffusion proposed by Floyd Steinberg, Jarvis-Judice and Ninke and Stucki in this section.

In the Floyd Steinberg method, the error diffusion is performed with serpentine scanning of the image, by diffusing the quantization error of each pixel intensity to the 4 pixels - one on the right, three below it. Floyd Steinberg method has divisor of 16 which makes it easier to compute even on older systems.

Floyd Steinberg Error Diffusion matrix = $$\frac{1}{16}\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 7\\ 3 & 5 & 1\end{bmatrix}$$

In the JJN or Jarvis-Judice-Ninke method, the 2 pixels which follow the target pixel and 10 pixels below the target pixel(in 2 rows), receive the quantization error. The Stucki method works in the same way as JJN except for the divisor being 42 and the weights being different compared to the divisor of 48 in JJN. The weights in the Stucki Error Diffusion matrix are in powers of two and hence the computation of the algorithm becomes faster. Hence Stucki is preferred over the JJN method, due to slight increase in computation speech, since both give very similar outputs.

JJN Error Diffusion Matrix = $$\frac{1}{48}\begin{bmatrix}0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 7 & 5\\ 3 & 5 & 7 & 5 & 3\\ 1 & 3 & 5 & 3 & 1\end{bmatrix}$$

Stucki Error Diffusion matrix = $\frac{1}{42}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$

APPROACH AND PROCEDURE

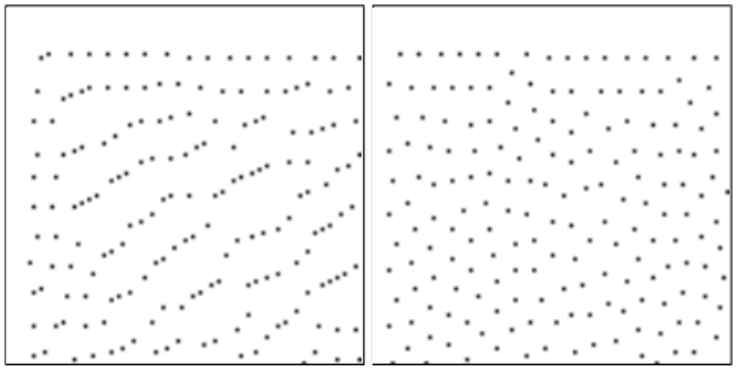
All the 3 methods discussed above use the same concept except for the change in the weights, matrix sizes and divisors for the matrices. JJN and Stucki methods have the matrix size to be 5x5 while Floyd Steinberg method has the matrix size as 3x3. The images are scanned in a serpentine manner, i.e for even rows, the pixels are scanned from the left to right and for odd rows, the pixels are scanned from right to left. The error diffusion for the pixels in the odd rows is also in the opposite direction as the even rows. In this way we ensure the quantization errors are diffused in both the directions.

Procedure:

1) Read the grayscale image into a 2D array.
2) Threshold the pixel intensities of the image, by comparing it to the threshold of 127 and setting to 0 if the pixel intensity is less than the threshold and set it to 255 otherwise.
3) Take the difference between the original pixel intensity and new monotone color intensity.
4) This error is then diffused to the pixels which follow the current pixel, by multiplying the pixel intensity of the neighbour with a certain fraction of the error and adding the product to that pixel's original value.
5) The fraction of the error is decided by choosing the value in the corresponding position in the Error diffusion matrix. Ensure every fraction of the error includes the divisor respective to that matrix i.e, the divisor is 16, 48 and 42 for Steinberg, JJN and Stucki methods respectively.
6) The scanning through the image is done in a serpentine manner, the pixels are scanned from the left to right and for odd rows, the pixels are scanned from right to left.
7) Ensure the error diffusion matrix is flipped for the odd rows, such the error is also diffused from right to left and the pixels which follow the quantized pixel receive the diffused error.
8) Once the traversal through the entire image has been completed and the quantization errors have been diffused, write the output 2D array back into a file.

EXPERIMENTAL RESULTS:

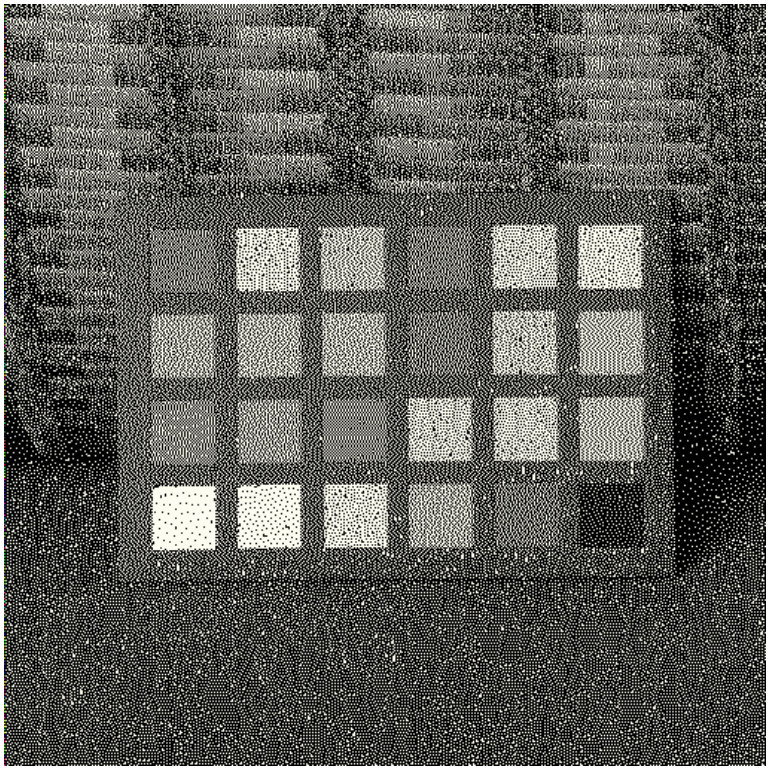
We use the serpentine scanning method to traverse through the entire image in all the methods, since if we use the traditional left to right scanning, we observe that the quantization gets accumulated on the right and at a certain point, all the pixels in the image will be pushed to black.



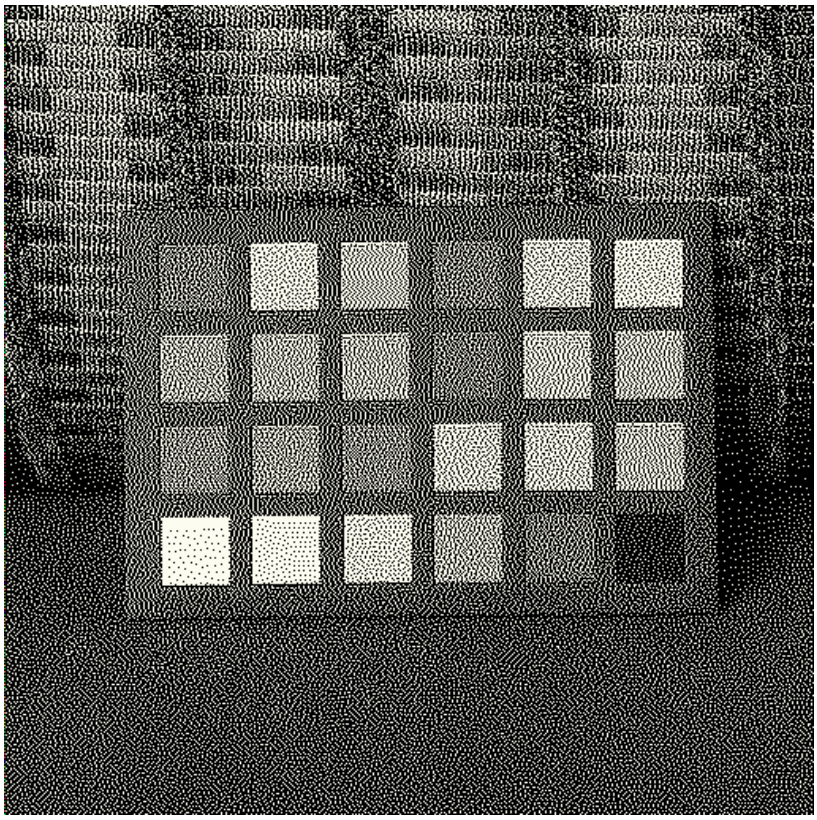
(a) Left-to-right (b) Serpentine

The above left image (a) is obtained when we perform left-to-right scanning and right image (b) is obtained when we perform serpentine scanning, the worm like artifacts are no longer observed. From the below experimental results we can observe that, the Floyd Steinberg method is less efficient in displaying a halftone image when compared to the JJN and the Stucki methods. The outputs of the Stucki and JJN are very similar, except for reduced patterns in the gray regions in the output of the Stucki method. Also since the Error Diffusion matrix used for the Stucki method has weights in orders of two, it give slightly better performance while execution.

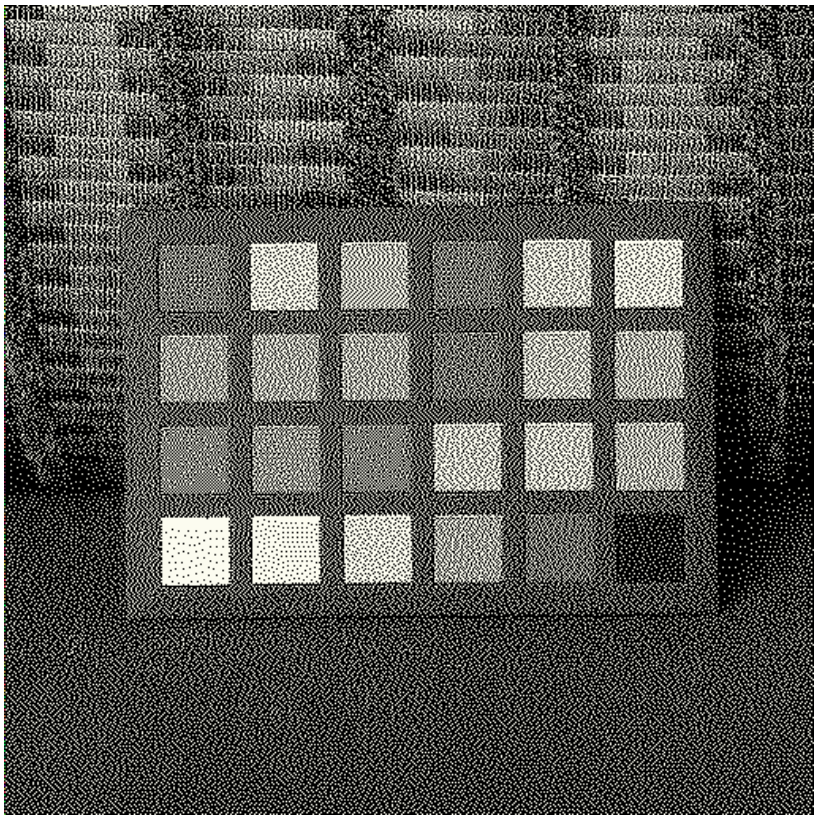
Floyd Steinberg
Error Diffusion method

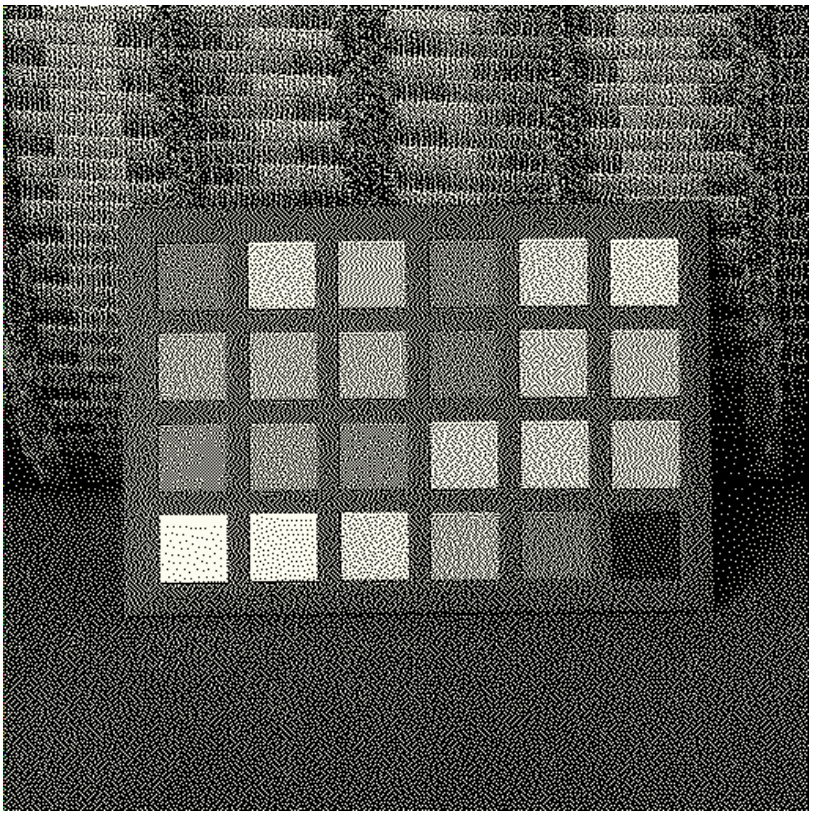


Jarvis, Judice and
Ninke Error Diffusion



Stucki Error
Diffusion method

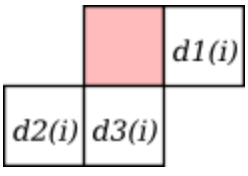




$$\frac{1}{186}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 36 & 18 \\ 9 & 18 & 36 & 18 & 9 \\ 3 & 9 & 18 & 9 & 3 \end{bmatrix}$$

Error Diffusion matrix designed by me, to see if increasing the divisor and varying the weights to higher values, changed the error diffused output (left).

I did try playing around with the weights and the divisor by keeping the size of the matrix 5x5, but was unable to improve the error diffused image quality. I propose the Variable coefficients error diffusion method, where a smaller matrix, is used and the error diffusion values vary according to the input image values. This method was proposed by Victor Ostromoukhov and he provides a list of 256 discrete valued triplets for d1, d2 and d3 values in the below mentioned matrix which [1] give pretty good results when combined with serpentine scanning. The smaller matrix, even known to introduce artifacts, reduces the sharpened aspect which generally introduced by Error Diffusion matrices. Another method to obtain a better digital halftoned image is to apply a sharpening filter before performing error diffusion.



Discrete Valued triplets **[2]**

Lena - Original Image

Variable Coefficients Diffused Image

[2]

2c. COLOR HALFTONING WITH ERROR DIFFUSION

The process of digital color halftoning involves transforming a continuous tone color image into an image with a certain set of colors which limited either due to the constraints in the displaying equipment or the printing equipment. This concept of color halftoning hold relevance due to the fact that color printers and low depth displays can print or display only a two colors or a few limited colors. The goal of color halftoning is to create the perception of a uniform and continuous tone image, by taking advantage of the fact that human visual system cannot differentiate minute color discriminations where displayed or printed after performing transformations. In this section we describe the procedure for separable error diffusion and the color diffusion methods which use Floyd steinberg method for diffusing the error to the neighbouring pixels.

Separable Color Diffusion

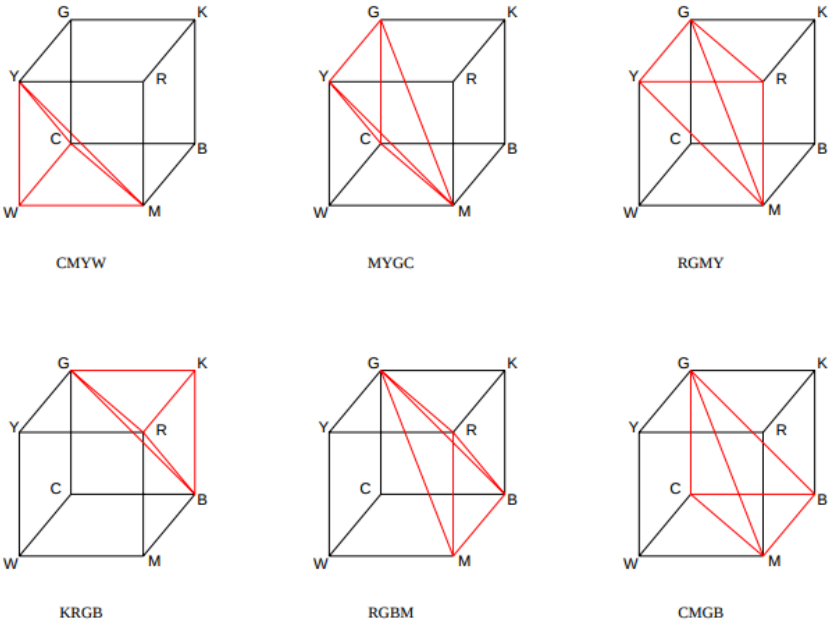
The method of separable color diffusion tries to achieve color halftoning by obtaining the Cyan, Magenta and Yellow channels of the input image and quantizing them. Then the Floyd-Steinberg error diffusion algorithm is used to quantize each channel separately. The resultant image will then have just 8 colors of the CMY cube i.e Cyan, Magenta, Yellow, Red, Green, Blue, Black and white.

MBVQ Color diffusion

In the previous problem we saw the half toning concept where algorithms are designed carefully designed with care to reduce artifacts and create a better perceived image. The reason for these artifacts is due to the variation of brightness in these dots. In the Minimal Variation

Brightness Criterion, where we perform color halftoning, we take advantage of the fact that color dots are not equally bright and hence using the appropriate colors in the appropriate locations will give the appearance of a continuous color image, despite using a limited number of color intensities.

In this MBVQ technique proposed by Shaked et al,[7] we overcome the issue of visible and distinct spotty color image, obtained upon using the Separable error diffusion method. In this method the CMYK color space is partitioned into six Minimum Brightness Variation Quadrants (MBVQ) as shown below.



[7]

Then, we find which quadrant each C, M and Y component of the image belongs to, find its closest vertex in that quadrant and then take on the intensity values of that vertex for that CMY color pixel. This quantization error is then diffused to the neighbouring pixels by apply the Floyd Steinberg method of error diffusion.

PROCEDURE:

Separable Error diffusion

1) Read the input color image into a single 1D array and separate the R, G and B channels into three 2D arrays
2) Obtain the Cyan, Magenta and Yellow components of the image by taking complement of each of the Red, Green and Blue channels
3) Apply the Floyd Steinberg method of error diffusion to every channel of the C, M and Y triplet.
4) Convert the error diffused C, M and Y outputs back to the R, G and B components
5) Combine the three color channels of the image and write it into a file.

Color Diffusion

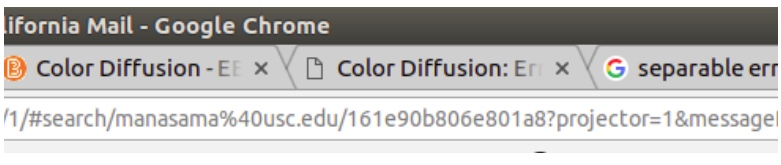
1) Read the input color image into a single 1D array and separate the R, G and B channels into three 2D arrays
2) Take the three components of each pixel and check which quadrant of the MBVQ halftone quadruple, the triplet belongs to.
3) After obtaining the corresponding quadruple, check to which vertex of the quadruple the Cyan, Magenta and Yellow components of the point are closest to.
4) Replace the C, M and Y values of that point with the intensity values of the vertex
5) Perform the steps 2) to 4) for every pixel in the image.
6) After step 6) perform Floyd Steinberg Error Diffusion to each of the newly obtained C, M and Y components of the image.
7) Obtain the complements of the C,M, Y to get the R,G and B components of the image.
8) Combine the three color channels of the image and write it into a file.

EXPERIMENTAL RESULTS

Original Image

Separable Error Diffusion output

MBVQ Color Diffusion output



We can see that the main shortcoming of the Separable Error diffusion output is that the we can see distinct color dots in the output image, making the output image appear like a discrete set of dots, and not a continuous tone image. Hence the MBVQ technique can be used to obtain a visually perceptive continuous tone image by using the Minimal Brightness Variation Quadrant technique. We see that the MBVQ technique gives a uniform color output and we are almost unable to understand that the image has just 8 colors.

PROBLEM 3 :

MORPHOLOGICAL PROCESSING

ABSTRACT AND MOTIVATION

Morphological Image processing is the field image processing where nonlinear functions are used to change the morphology or features of the image. Some of the morphological processes we discuss today are shrinking, thinning, skeletonizing. There are some other methods too like dilation, erosion, Hit and miss transform, opening and closing of an image.

The morphological processing methods take the relative ordering of pixels in an image and not the pixel intensities into consideration while performing computations and hence are able to modify a binary image effectively.

Morphological operations probe an image with a structuring element, where the structuring element, is a matrix of pixels with binary values. The structuring element is placed at every pixel location and the neighbouring pixel values are checked. If the neighbouring pixels match the structuring element values, then it is considered a hit, and further manipulations are carried out.[3] In the methods discussed below, we use 3x3 sized structuring elements and use the hit-and-miss concept to perform morphological operations. Erosion and Dilation are two fundamental operations in morphological processing on which other methods are based. In erosion, an image shrinks uniformly and in dilation, an object grows uniformly. Shrinking, Skeletonizing and Thinning are implementations of controlled erosion operation. All use the same procedure and algorithm, except for varying the conditional and unconditional mark patterns used to identify the match between the pattern and the neighbouring pixels of the target pixel.

For the counting game, in order to count the number of unique jigsaw puzzles, we use the concept of connected components where we count the number of components (puzzles) present in the image by labelling each foreground pixel belonging to an puzzle a unique label. We identify the number of components, then find out how many holes and protrusions are present in each puzzle. We then rotate and reflect each puzzle, compare it with the remaining puzzle pieces to find the number of unique jigsaw puzzles.

The applications of connected components is popular in the field of automated image analysis, where the extracting and labeling of various disconnected and connected parts of an image is crucial.

APPROACH AND PROCEDURE

a) Shrinking

In Shrinking, which is a modified implementation of erosion, smaller square structuring elements of the size 2x2 to 5x5 are used to shrink an image. This is done by stripping away layers of pixels over multiple iterations from the outer boundaries as well as the inner boundaries of sections in the image. This will result in gaps within different regions becoming larger and smaller details of the image end up getting eliminated. [4][3]

b) Thinning

In the method of thinning, we try to erase the foreground pixels in the image, by using the structuring element such that the if the image does not have any holes then the image will be reduced to a single connected stroke which is at equal distance from the boundaries of the original foreground. Thinning can used to separate multiple objects in a image by performing it over multiple iterations.

c) Skeletonizing

Skeletonizing is the process of repeatedly reducing the foreground regions in a binary image to an extent that only the skeleton or the basic structure of the region is left behind, while throwing away majority of the image pixels. Thinning produces an output very similar to a skeletonized output.

PROCEDURE FOR SHRINKING

a) To count the number of stars in the image

1) Read the input image file into a 2D array.

2) Invert the image to make the background pixels black and the foreground pixels white in order to perform the following morphological operations.

3) Perform fixed thresholding on the image if there are some pixel values which are neither black or white.
4) Convert the pixel intensities which are 0 and 255 to 0 and 1, so that the different conditional and unconditional mark patterns can be compared with the image to perform hit and miss transformations.

5) Perform the stage1 conditional mark patterns comparison which is conducted by traversing through the entire image, taking one pixel into consideration at a time.

6) While traversing through the image, consider every pixel set to 1 as the target pixel, and calculate its bond value by taking its 8 neighbours into consideration assuming there is a 3x3 window around it.

7) The bond of the target pixel is calculated based on the occurrence of a 1 in any of its neighbour positions. If the diagonal neighbours are set to 1, then their bond value contribution will be (1*Number of diagonal elements set to binary value 1). If any of the pixels, above, below, to the left or to the right of the target pixel have been set to 1, then they contribute (2*Number of pixels set to binary value 1).

8) Once the bond value is calculated, we have check if the neighbouring pixels match any of the conditional mark patterns corresponding to that bond value.

9) The implementation I have used is to append each pixel value(of type unsigned char) to a string and compare the mark patterns which are stored as strings too against the string formed by the pixels in the 3x3 window.

10) If a match is found, then the corresponding pixel is set to 1, else it is set to 0.

11) While traversing through the image, every black pixel is kept as black.

12) The output obtained at the end of the first stage of matching the conditional mark patterns, is fed as input to the second stage of unconditional mark patterns.

13) In the second stage, similar to the first stage, a pixel is taken into consideration if it is set to 1. Given the knowledge it is set to 1, we get the neighbouring pixel values and create a string from it for comparison with other unconditional mark patterns.

14) In the second stage, we need pay attention to values in the mark patterns.
    a) The ‘m’ value in the mark pattern indicates that the corresponding pixel value in the image should be 1.
    b) The ‘d’ value in the mark pattern, says that the corresponding pixel in the image can be either a 1 or a 0 - in both cases, if the image pixels are same then the pixels will be said to match, for the corresponding pixels.
    c) For the ‘a’ or ‘b’ or ‘c’ values, at least one of the image pixels in the corresponding positions, should have been set to 1.
15) If all the above values and the remaining values in the unconditional mark pattern match, then the pixel value of the target pixel will be set to the corresponding pixel value in the binary image, (provided as input for the first stage) else it is set to zero.

16) For the pixels which are 0, the values from the corresponding binary image(provided as input for the first stage) are replaced in the output image

17) Repeat the stages 1 and 2 of comparing the conditional and unconditional mark patterns, for approximately 15 - 20 times, until all the foreground pixels in the image(stars) reduce to a single white pixel and remain constant for a few more iterations.

18) Count the number of stars in the image, by counting every white pixel, surrounded by black pixels in all 8 directions (considering a 3x3 matrix) as one star.

b) To find the different square sizes present in the image

1) Repeat the steps 1) to 16) listed above for part b)
2) In addition to the above steps, while repeating the stage 1 and stage 2 of shrinking, count how many white pixels in the image are surrounded in all 8 directions by a black pixel.
3) This will give a count of the different sizes of the stars
4) Let's say at the end of the first iteration(stage 1 + stage 2), you find there are 2 white pixels surrounded by black pixels in all directions, then we can say that there are 2 stars of size 1.
5) Repeat the steps 2) to 3) to count the stars and their frequencies.

PROCEDURE - THINNING AND SKELETONIZING

1) Repeat the steps 1) to 17) listed for Shrinking above.

2) For the thinning operation, repeat the steps 1) to 16) by using the appropriate image, the corresponding conditional and unconditional mark patterns.

3) For the skeletonizing operation, repeat the above steps 1) to 17) by using the appropriate image, the corresponding conditional and unconditional mark patterns.

4) Ensure you have inverted the image to make the background pixels black and the foreground pixels white before performing the above morphological operations.

PROCEDURE - COUNTING GAME

a) To count the total number of jigsaw puzzles

1) Read the input image into a 2D array.
2) Convert the pixel intensity values from 0 to 255 values to 0 to 1, so that performing morphological operations is easier.

3) Since the morphological operations can only be performed when the foreground pixels are white and background pixels are black, invert the entire jigsaw puzzle.
4) Perform the erosion operation on the entire image. In the erosion operation, every pixel in the image is scanned one at a time. If the target pixel is surrounded by pixels of intensity value 1, in a 11x11 window, then at the target pixel's position in the output image, intensity value is set to one.
5) The next step is to perform dilatation using a 27x27 sized window.
6) In the dilation step, every pixel in the image is chosen one at a time and checked if its neighbours in a 27 x 27 window are set to the intensity value 1. If yes, then the corresponding output pixel is set to one, else it is set to zero.
7) The step which follows is the shrinking step where the 3x3 window is used and the stages 1 and 2 of shrinking are performed 40 times to finally get the jigsaw puzzle reduced to a single pixel at the end of shrinking operation.
8) All the pixels which are white are counted to give the total number of jigsaw puzzles.

b) To count the number of unique pieces in the jigsaw puzzle

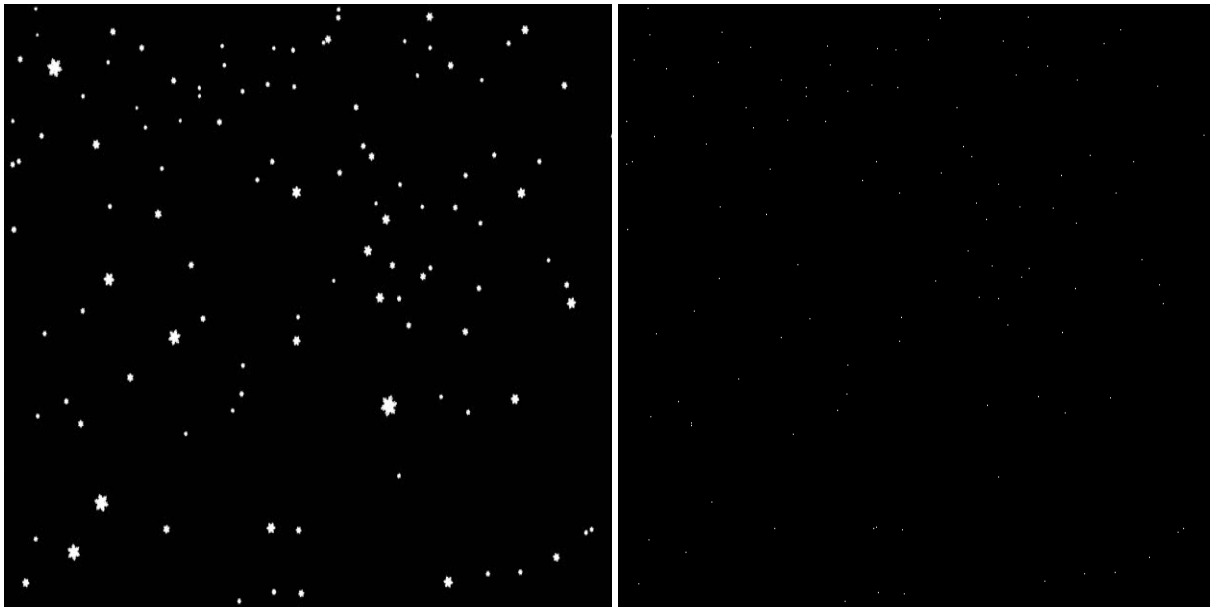
1) Read the input image into a 2D array
2) Convert the pixel intensity values from 0 to 255 values to 0 to 1, so that performing morphological operations is easier.
3) Since the morphological operations can only be performed when the foreground pixels are white and background pixels are black, invert the entire jigsaw puzzle.
4) Next, we scan through the entire image, checking one pixel at a time. If the pixel is set to one, then the scanning function checks the four neighbours preceding the image (the 3 pixels above the current pixel and the pixel to the left of the current pixel). It performs one the following operations based on the intensity values of the neighbours -
    a) If none of the neighbours have 1 intensity value, then the current pixel is assigned a new label.
    b) If one of its neighbours has the pixel intensity to be 1, then the current pixel is assigned the neighbours label.
    c) If more than one neighbours are set to 1, the current pixel is assigned the smallest valued label, and an entry is made in a lookup table (an array in this case), to remember the fact that both the pixels belong to the same connected component.

5) The step 4 is repeated until all the pixels (white) in the image have been assigned a label and an entry has been made in the lookup table for the relabelling.
6) In this step, with the lookup table as reference all the labels which had been linked together are assigned the smallest label in the neighbourhood, by checking which label each pixel is mapped to and which is the head label.
7) At the end of the relabelling stage we would have relabelled the entire image to have just N unique labels depending on the number of connected components.

8) Next the width and the height of each pixel have to be found by finding the minimum x and y coordinates for each jigsaw puzzle.
9) With the minimum x and y coordinate as the starting pixel, we apply the XOR filter where every pixel is XOR'd with 1 to remove the body of the puzzle and retain only the holes and protrusions in the image.
10) Next we count the number of holes and protrusions in each jigsaw puzzle and store it in the form of an array.
11) The arrays containing the hole and protrusion count for a certain jigsaw puzzle is compared by performing rotations and reflections.
12) If at any rotated or reflected comparison stage, if both the hole and protrusion directions match for 2 jigsaw puzzles, then they are said to be non-unique.
13) Repeat the steps 4) to 12) to get the total number of unique jigsaw puzzles.

EXPERIMENTAL RESULTS:

STAR COUNT (SHRINKING OUTPUT) :



Original Image - stars.raw

Shrinking output - stars.raw

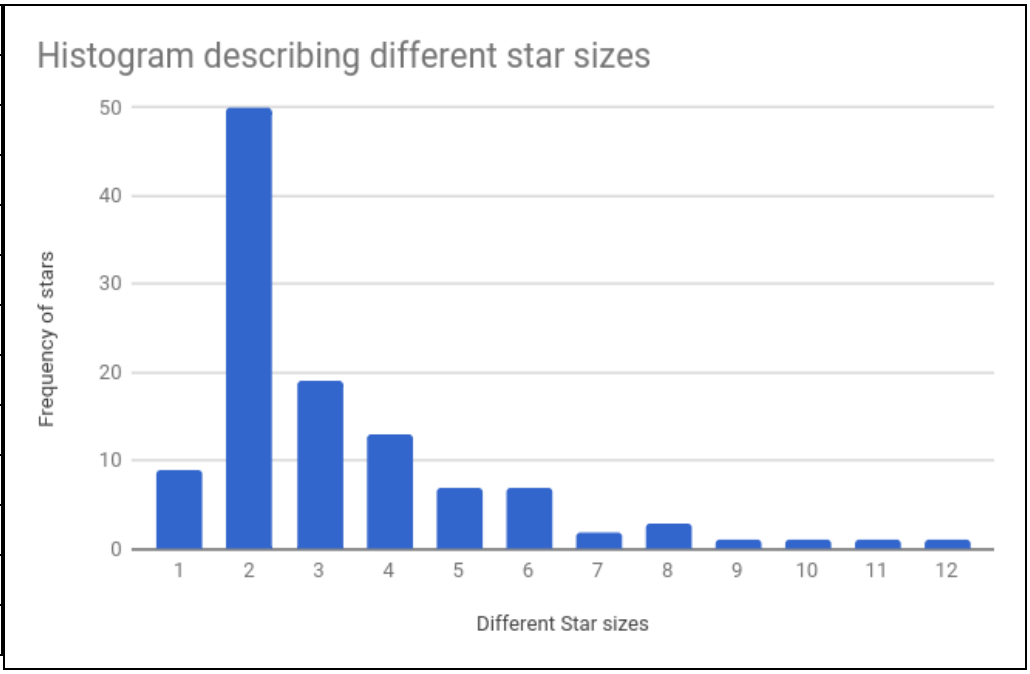
We can find that after performing the stage 1 and stage 2 of shrinking repeatedly for around 15 iterations, all the stars shrink to a single white pixel and when the number of white pixels in the black background are counted, we find that there are 112 stars. I was observing that one of the stars gets split into 2 parts in the last stage due to the shrinking and may account for an extra star.

Thus we say that the shrinking morphological operation, can be used to count the number of objects in a monotone image and has practical applications in computer vision and image processing.

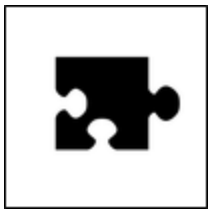
DIFFERENT STAR SIZES

When the number of stars which are being shrunk to one white pixel at the end of each iteration of shrinking are counted, we find that there are totally 12 different sized stars in the image and we find that there are more stars which are of size 2 when compared to the rest of the sizes.

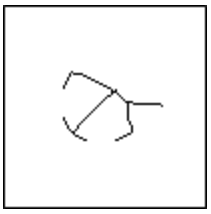
Star sizes	Frequency
1	9
2	50
3	19
4	13
5	7
6	7
7	2
8	3
9	1
10	1
11	1
12	1



THINNING OUTPUT:



Original Image - jigsaw_1.raw

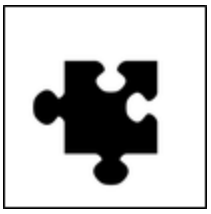


Thinned output - jigsaw_1.raw

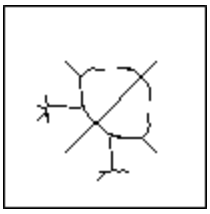
The above is obtained after applying the thinning operation on the input jigsaw_1.raw image over 30 iterations. The first boundary layer of the image is stripped away in the first stage of the first iteration, then foreground is again filled up in the second stage of the first iteration. In the subsequent iterations. On after the other, layers of the boundary are removed, leaving only the inner structure of the image.

SKELETONIZING OUTPUT:

The skeletonized output of the input image jigsaw_2.raw is obtained after performing the skeletonizing operations for 40 iterations. The output obtained for skeletonizing is similar to the output obtained for thinning when compared to the structure, but the skeletonized output is giving the actual structure of the image including the parts like the centre, indicating the direction of more black pixels, holes and protrusions.



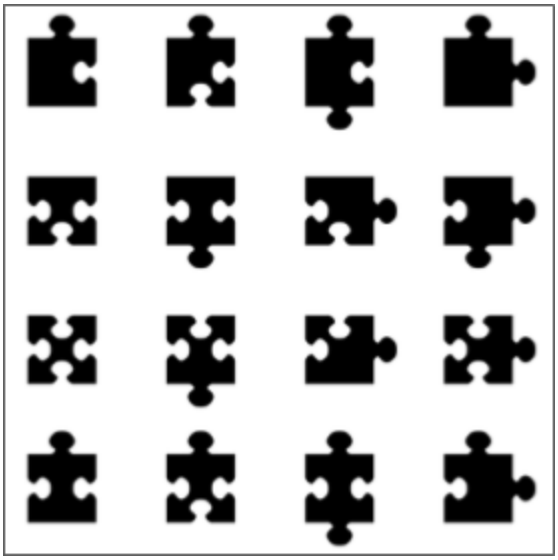
Original Image - jigsaw_2.raw



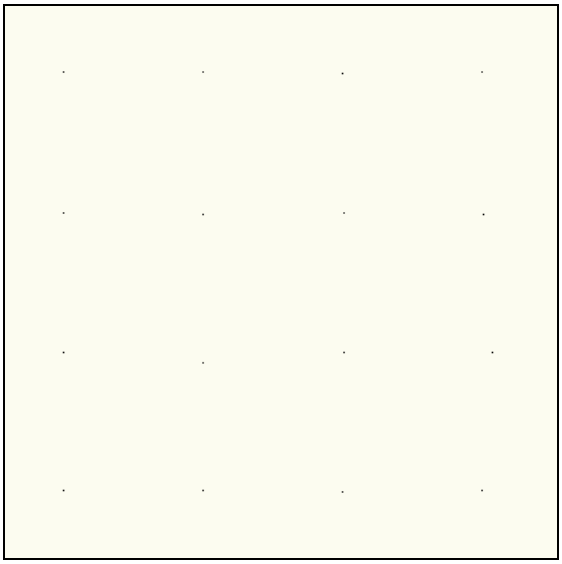
Skeletonized output - jigsaw_2.raw

COUNTING GAME (EROSION + DILATION + SHRINKING)

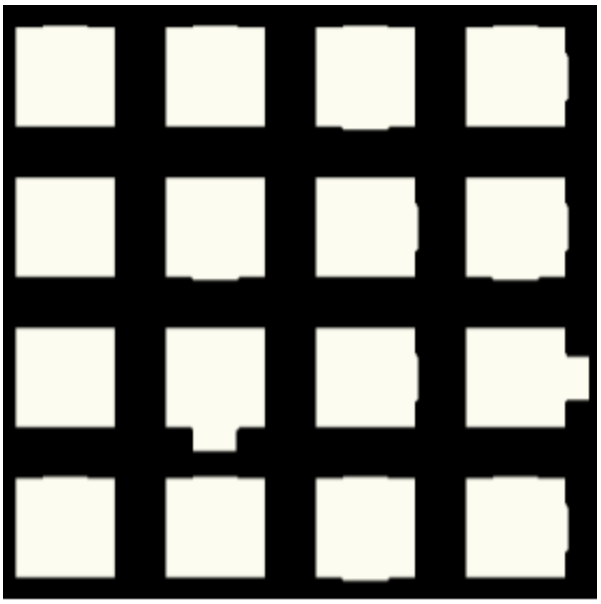
To count the total number of jigsaw puzzles : In order to count the total number of jigsaw puzzles in the image, I performed the image inversion (convert the black pixels to white and vice versa), followed by the Erosion, keeping a structuring element of size 11x11, followed by Dilation using a 27x27 window and finally performing 40 iterations of shrinking. At the end of above steps, the final output image will have 16 white pixels at the centre of each jigsaw puzzle. In the Erosion setup, we work on removing the rough edges, protrusions and holes in image to get a smoothened general structure. The dilation step, fills the jigsaw puzzle completely, giving it the almost square shape such that the repeated shrinking operation will reduce each puzzle to a pixel and it can be counted as a single point.



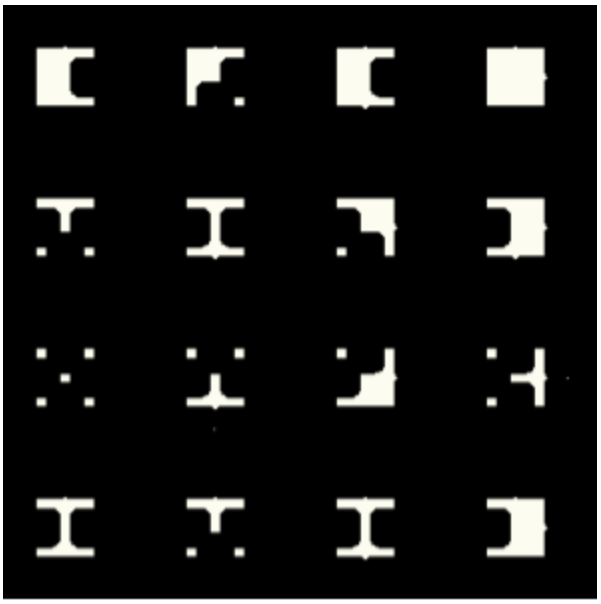
Original Input Image - board.raw



Final output - Total number of jigsaw puzzles



Dilated output - board.raw



Output after Erosion - board.raw

To count the number of unique pieces in the board

In order to count the number of unique pieces in the board.raw image, we use the concept of connected components to label the puzzles and then get the holes and protrusions in the image, by performing an XOR operation. The number of holes and protrusions in each direction are calculated. The holes and protrusion values are then compared with every other puzzle's values by rotation and reflection to arrive at the conclusion that there are 10 unique puzzles in the board.raw image. Some of the pieces which are same other puzzles are -

a) 10, 12, 14 - All three are same
b) 6 and 13
c) 2, 7 and 11
d) 8 and 16

The assumptions made which designing the counting game are:

1) The input image will have a square structure, since the XOR operation to obtain the holes and protrusions is designed taking a 45x45 sized XOR matrix
2) After the relabelling stage, a new set of consecutive labels in increasing order have to be assigned.

REFERENCES:

1) **Victor Ostromoukhov. A Simple and Efficient Error-Diffusion Algorithm. In Proceedings of SIGGRAPH 2001, in ACM Computer Graphics, Annual Conference Series, pp. 567–572, 2001**
2) **Libcaca study: the science behind colour ASCII art - http://caca.zoy.org/study/biblio.html**
3) **Morphological Image Processing - https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm**
4) **William Pratt - Digital Image Processing**
5) **Analytical Methods for Squaring the Disc - Chamberlain Fong - spectralfft@yahoo.com - Seoul ICM 2014**
6) **https://taotaoorange.wordpress.com/2011/05/09/view-geometry-basics/**
7) **D. Shaked, N. Arad, A. Fitzhugh, I. Sobel, "Color Diffusion: Error-Diffusion for Color Halftones", HP Labs Technical Report, HPL-96-128R1, 1996.**