

## **EE 569 Homework #2**

Maroof Mohammed Farooq

maroofmf@usc.edu

7126869945

# Table of Contents

1. Problem 1.....	3
1.1 Abstract and Motivation .....	3
1.2 Approach and Procedures .....	3
1.2.1 Geometrical Warping.. .....	4
1.2.2 Puzzle Matching.....	5
1.2.3 Homographic Transformation and Image Overlay.....	6
1.3 Experiment Results.....	8
1.4 Discussion.....	9
2. Problem 2 .....	10
2.1 Abstract and Motivation .....	10
2.2 Approach and Procedures .....	10
2.2.1 Dithering Matrix based Half toning .....	10
2.2.2 Error Diffusion based Dithering .....	11
2.3 Experiment Results.....	12
2.4 Discussion.....	13
3. Problem3 .....	15
3.1Abstract and Motivation .....	15
3.2Approach and Procedures .....	15
3.2.1 Rice Grain Inspection .....	15
3.2.2 MPEG – 7.....	16
3.3 Experiment Results.....	17
3.4 Discussion.....	18
4. References .....	20

# 1. Problem 1

## 1.1 Abstract and Motivation:

Geometrical image modification consists of a set of most common operations performed in image processing. Operations such as rotation, translation and scaling of the images spatially that provides the flexibility to produce a desired result. Apart from image enhancement and restoration, it is one of the important steps in image manipulation. They are extensively used in computer graphics, image morphing and panorama stitching. In this problem we will implement all the geometrical operations and utilize them in various situations.

It is sometimes desirable to manipulate an image for considerable distortion in the shape it represents. This can be achieved by properly utilizing the concept of geometrical warping. In essence, geometrical warping is a transformation that alters the spatial configuration of an image. This is mainly used for creative purposes. In this problem we will look into the implementation and discuss a warping technique using triangles to achieve such effects.

Geometrical image modification also finds application in hole filling problems. With sufficient knowledge about the location and orientation of holes and its corresponding patches, we can implement a hole filling algorithm by heavily utilizing the geometrical image operations. In this problem we discuss an algorithm to match puzzle pieces using geometrical operations on images.

By definition, an image is a 2D representation of the 3D world that is stored in a raster form. We need to study the relation between the 3D world coordinates and the 2D image coordinates for implementing interesting applications such as panorama stitching, synthesizing an image with text or other images and rotating camera effects. This relation is given by estimating the intrinsic and extrinsic parameters of the camera. The intrinsic and extrinsic parameter matrices can be reduced to a homographic matrix by assuming four points lying on the same plane. In this problem, we will explore an algorithm to estimate the homographic transformation matrix which will be utilized to synthesize images.

## 1.2 Approach and Procedures:

We prefer to use the Cartesian coordinate system to modify the image geometrically. This is because operations such as scaling, rotation and translation becomes very easy when an image is viewed on the Cartesian coordinates. Hence it becomes necessary for us to compute the Cartesian coordinates from the image coordinates and vice-versa. To make the equations linear, the input and the output coordinates are augmented by appending a '1' in the end.

When an image is initialized, the Cartesian to image and image to Cartesian is set such that it can map the values between Cartesian coordinates and image coordinates. The image to Cartesian coordinate relation is given as follows:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0.5 \\ -1 & 0 & h - 0.5 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} r \\ c \\ 1 \end{bmatrix} \text{----- (Equation 1)}$$

In the above equation, the input image row coordinate 'r' and column coordinate 'c' are augmented and multiplied by a matrix to get 'x' and 'y' Cartesian coordinates. Here 'h' refers to the image height. By taking the inverse of the above matrix and multiplying it with the Cartesian coordinates, we can obtain the corresponding image coordinates. These matrices are calculated every time an image object is initialized.

## 1.2.1 Geometrical Warping

In this problem, we will consider transforming square image into a diamond shaped image with the help of triangle warping technique. If we closely observe the input and output relation of the square to diamond mapping, we can find 8 unique triangles that can help us achieve the output. The eight triangles are shown in figure 1(a) and figure 1(b). Each triangle can be represented with three control points.

Given two triangles ABC and A'B'C', where ABC is the input triangle and A'B'C' is the output triangle, we have established a transformation matrix "T" that maps points lying in ABC to point A'B'C'. This can be given by the following<sup>[1]</sup>:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{----- (Equation 2) }^{[1]}$$

Here  $x'$ ,  $y'$  are the output Cartesian coordinates and  $x$ ,  $y$  are the input Cartesian coordinates. With the knowledge of three input triangle points and three triangle output points, we can solve for  $a$ ,  $b$ ,  $c$ ,  $d$ ,  $e$  and  $f$ . This has been achieved using the following formula:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x'_1 & x'_2 & x'_3 \\ y'_1 & y'_2 & y'_3 \\ 1 & 1 & 1 \end{bmatrix} \times pinv \left\{ \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \right\} \text{----- (Equation 3)}$$

This transformation can be clearly illustrated in figure 1(c). This is called forward warping. In this type of warping, each input value gets mapped to an output coordinate. Due to floating point numbers involved in this calculation, the output coordinate need not necessarily be a whole number. This creates a problem since we cannot access coordinates in the output that are of floating value. Additionally, truncation of the floating point coordinates to get the nearest whole number can lead to serious deterioration of the image quality.

In order to address this problem, we have implemented inverse wrapping with bilinear interpolation that maps the output coordinates to the input coordinates. Similarly, the input coordinates need not necessarily be a whole number. In the case when the input coordinates are of floating value, we use bilinear interpolation technique to determine the value of the corresponding pixel.

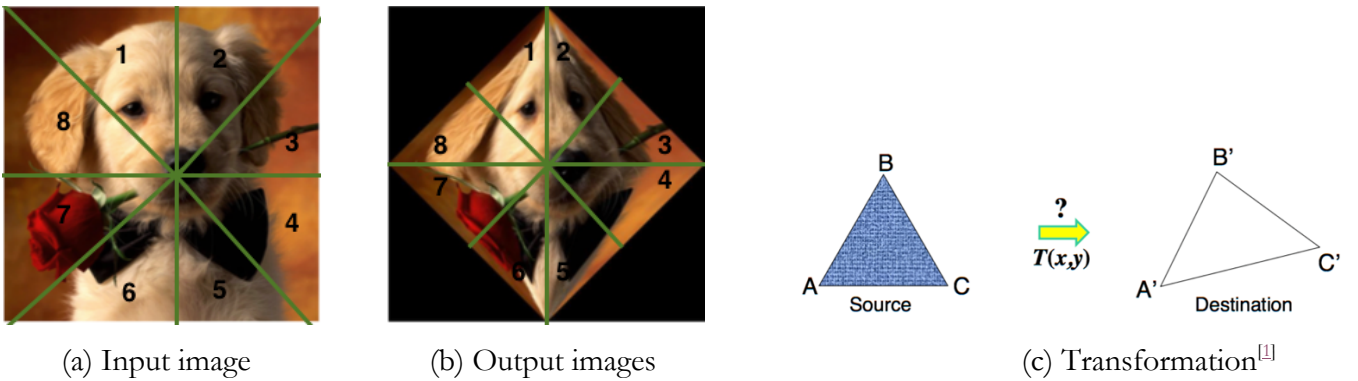


Figure 1: Triangle mapping

For each input-output triangles, the transformation matrices were obtained from MATLAB by using Equation 3 and were found to be:

$$\begin{aligned}
T_1 &= \begin{bmatrix} 2 & 0 & -150.5 \\ -1 & 1 & 150.5 \\ 0 & 0 & 1 \end{bmatrix} & T_2 &= \begin{bmatrix} 2.0135 & 0 & -152.5338 \\ 1.0135 & 1 & -152.5338 \\ 0 & 0 & 1 \end{bmatrix} & T_3 &= \begin{bmatrix} 1 & 1.0135 & -152.5338 \\ 0 & 2.0135 & -152.5338 \\ 0 & 0 & 1 \end{bmatrix} \\
T_4 &= \begin{bmatrix} 1 & -1 & 150.5 \\ 0 & 2 & -150.5 \\ 0 & 0 & 1 \end{bmatrix} & T_5 &= \begin{bmatrix} 2.0135 & 0 & -152.5338 \\ -1.0135 & 1 & 152.5338 \\ 0 & 0 & 1 \end{bmatrix} & T_6 &= \begin{bmatrix} 2 & 0 & -150.5 \\ 1 & 1 & -150.5 \\ 0 & 0 & 1 \end{bmatrix} \\
T_7 &= \begin{bmatrix} 1 & -1 & -150.5 \\ 0 & 2 & -150.5 \\ 0 & 0 & 1 \end{bmatrix} & T_8 &= \begin{bmatrix} 1 & -1.0135 & 152.5338 \\ 0 & 2.0135 & -152.5338 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

The color channels were decomposed and the transformation to each output point was applied to find its corresponding input point. Bilinear interpolation was utilized when needed to obtain smooth images.

## 1.2.2 Puzzle Matching

In this problem, we will fill holes in images with its corresponding patches using geometrical transformation. This is implemented by first finding the corners of the holes and patches by Harris corner algorithm.<sup>[2]</sup> To utilize this algorithm, we first convert the RGB image to grayscale using the following formula:

$$G(i, j) = 0.299 \times R(i, j) + 0.587 \times G(i, j) + 0.114 \times B(i, j) \text{ -----(Equation 4)}$$

The output grayscale value for row ‘i’ and column ‘j’ is stored in G(i, j) where R(i, j) is the red component of the input, G(i, j) is the green component and B(i, j) is the blue component of the input.

In essence, the Harris corner algorithm checks for changes in the x and y direction in a window and classifies the center pixel as a corner if the changes in x and y direction are significant. This is achieved by computing the x and y derivative of the image. These derivatives were calculated by convolving the image with Sobel operator. The following operations were utilized to compute the x and y gradient of the grayscale image:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \quad ; \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A \text{ -----(Equation 5)}$$

In the above equation,  $G_x$  is the gradient along the x direction and  $G_y$  is the gradient along the y direction of the original image A. After computing the gradients, we pass a 3x3 window through the input and compute the values of M matrix which is given by:

$$M = \sum_{x,y} w(x, y) \times \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \text{ -----(Equation 6)}$$

The M matrix contains essential information about x and y gradients in the window. Gradient value of each pixel  $I_x$  and  $I_y$  in the 3x3 window is weighted by a weight function and summed up. The weight function can be a 2D Gaussian window or a box function whose value is 1. For the sake of simplicity we have considered the value of

$w(x, y) = 1$  for all instances. Later the M matrix is used to measure the corner response value, R. The value of R is calculated by applying the following formula:

$$R = \det(M) - k(\text{trace}(M))^2 \text{ ----(Equation 7)}$$

The value of R is associated with the Eigen values of M. Higher the value of R, higher the Eigen values. Here k is an empirically determined value that is between 0.04-0.06. For this problem the value of k is chosen to be 0.04.

Sometimes due to noise or unsharp corners we may get multiple corner points near the true corner. This can be eliminated by non-max suppression. To implement non-max suppression, we store all the points in a hash map which maps the corner number to its coordinates and R value. We then loop through all the corners and find the indices which are within the 5 pixel radius of the current corner. For two or more neighboring corners, we choose the one with the highest R value and neglect the others. This way we are more robust to noise and unsharp corners.

Similar to the mapping done in section 1.2.1, we calculate the transformation matrix using the 4 corners obtained from the patches and 4 corners from the target images. Using reverse mapping, we have mapped the patches into the desired target images.

The matrices found for puzzle 1 and puzzle 2 were:

$$T_1 = \begin{bmatrix} 1.3861 & 0.3762 & -283.1485 \\ -0.3812 & 1.3911 & -17.8589 \\ 0 & 0 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} -0.0594 & -0.7030 & 514.8812 \\ 0.6980 & -0.0644 & 33.3540 \\ 0 & 0 & 1 \end{bmatrix}$$

where  $T_1$  is the transformation for puzzle 1 to pieces and  $T_2$  is the transformation for puzzle 2 to pieces.

The problem of fractional indices that was discussed in the previous section has been solved by using bilinear interpolation.

### 1.2.3 Homographic Transformation and Image Overlay

In this problem we will synthesize two images that are taken from different camera view points. This can be done by understanding the forward projection of converting the world coordinates to pixel coordinates. The position of a real world object in an image is effected by the focal length and orientation of the camera. The relation between the real world coordinates and pixel coordinates are given by extrinsic and intrinsic parameters. Utilizing these parameters, we can relate the world and image coordinates as follows:

$$w \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & C_x \\ 0 & f & C_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \text{ ----(Equation 8)}$$

where x,y are the image coordinates, 'f' is the focal length 'r' is the rotation values and X,Y,Z are the real world coordinates.

To synthesize two images taken from different camera view points, we need to have the information about the intrinsic and extrinsic parameters of the camera. If these parameters are not specified, we need to estimate the homographic transformation matrix H. Homography assumes that the two images lie on the same planar surface in space. The estimation of homographic transformation matrix was done by considering 4 points each from the host and test images that should lie on the same plane.

To estimate the 3x3 homographic transformation matrix, we had 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. The H matrix was estimated on MATLAB by using the formula given below:

$$\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} X_1 \\ Y_1 \\ X_2 \\ Y_2 \\ X_3 \\ Y_3 \\ X_4 \\ Y_4 \end{bmatrix} \times pinv \left( \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1X_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1Y_1 & -y_1Y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4X_4 & -y_4X_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4Y_4 & -y_4Y_4 \end{bmatrix} \right) \text{-----(Equation 9)}$$

The H matrix values  $h_{ij}$  were determined by multiplying the vector containing text image coordinates  $X_i, Y_i$  with the pseudo-inverse of host image coordinates  $x_i, y_i$ . The H matrix from this calculation was found to be:

$$H = \begin{bmatrix} 0.0684 & 0.8874 & -66.3633 \\ -0.6006 & 0.4089 & 304.3954 \\ 0.0001 & -0.0028 & 1 \end{bmatrix}$$

After estimating the H matrix, we looped through the host image and found the corresponding pixel value in the text image. This was done by using:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = H \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}; \quad X = \frac{x'}{w} \quad ; \quad Y = \frac{y'}{w} \quad \text{---(Equation 10)}$$

where X , Y are the corresponding Cartesian coordinates of the text image.

When we look at the embedded text image we have a background region which is black for the tartan's text image and white for the Trojans text image. To remove this background, we have used simple thresholding on the the obtained pixel value for all the color channels after transformation. If the obtained pixel value of all the channels exceed the threshold, then we don't override the pixel value in the host image. This technique provides a very effective result.

### 1.3 Experiment Results:



(a) Kitten\_1 wrapped



(b) Kitten\_2 wrapped

Figure 2: Output from spatial warping algorithm

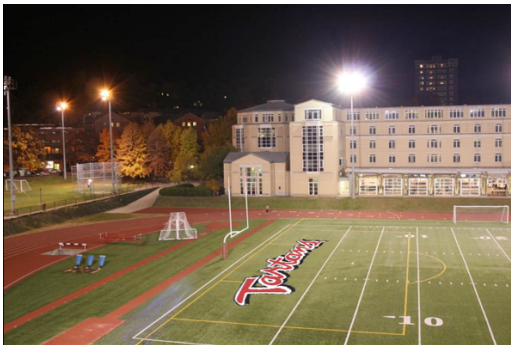


(a) Puzzle Matching 1



(b) Puzzle Matching 2

Figure 3: Puzzle matching output



(a) Embedded Tartan's image



(b) Embedded Trojan's image

Figure 4: Image overlay on host image



## 1.4 Discussion:

Various geometrical image operations were implemented in this problem. One operation that needs significant improvement is rotation. Due to rotation, few original pixels may be lost as the presence of fractional image coordinates is inevitable. As a result of this the image gets blurry. If we compare the images in figure 5, we can see the decrease in quality due to rotation. To a certain extent, bilinear interpolation helps to preserve the quality. To further improve on rotation operation, we can implement rotation by shearing.



(a) Without Interpolation

(b) With interpolation

(c) Original image

Figure 5: Rotation operation

In the output produced by the spatial warping algorithm in figure 2, we can observe that the mapping is one to one. The shape of the output image looks distorted as desired. Upon closer inspection of the output images, we find that a lot of original pixels are lost. This is mainly because we are under sampling the original image.

Looking at the output produced by the hole filling algorithm for the puzzle matching problem, we can see that the corners have been successfully detected by Harris corner algorithm. Upon closer inspection of the output of puzzle 1, we can find sharp edges near the boundaries of the hole. This is mainly because of color variation in the puzzle piece and target image. This can be resolved by applying median filter of a suitable window size across the boundaries to smoothen it.

The output produced for Trump puzzle matching doesn't look satisfactory. A lot of improvements can be done by increasing the sharpness of the puzzle piece. The blurry puzzle piece was the result of poor interpolation. A sharpening filter could be applied over the puzzle piece before merging it with the target image.

Additionally, for puzzle matching, four coordinates were used to find 6 parameters of the matrix. This resulted in an overdetermined system. Least squares approach was used to minimize the mapping error and if we closely observe, the mapping is still one to one.

The output for homographic transformation and image overlay could be seen in figure 4. Upon closer inspection we can see that the background has been removed. The threshold values for background removal were selected based on trial and error. The quality of the embedded text images was preserved. Furthermore, this implementation has the potential to be automated by computing the H matrix just by specifying the input and output points.

## 2. Problem 2

### 2.1 Abstract and Motivation

Dithering is a type of thresholding where a color or grayscale image is converted into a cluster of binary pixels. It works by approximating unavailable colors with available colors. The importance of dithering can be illustrated by looking at few of its applications.

Printing a full color image on a non color printer or displaying images with high color information on a limited color screen can cause problems. The original image may contain subtle gradients that provide details to an object. These subtle gradients may be replaced with blobs of uniform color if dithering is not performed. <sup>[3]</sup> To get the best quality in devices that support less color, dithering is performed.

This process of reducing the bit resolution causes quantization errors that can make the image lose useful information. To avoid this, dithering is done by intentionally applying blue noise to randomize the quantization error. Using ordered dithering and error diffusion method are two ways to introduce blue noise and produce a better result for less color devices. In this problem, we will explore and compare different algorithms to perform dithering.

### 2.2 Approach and Procedures

#### 2.2.1 Dithering Matrix based Half toning:

Ordered dithering is commonly used in the industry where images of higher colors are used on a display of less color depth. This algorithm achieves dithering by applying a pre-calculated threshold map on the image. These threshold maps are also know as Bayer index matrices that are square matrices with length of sides as power of 2. In this problem, we will apply 2x2, 4x4 and 8x8 Bayer index matrices individually and compare the resulting dithered image.

Bayer index matrices were calculated using the recursion formula given in the problem statement. The following Bayer matrices were obtained:

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

The obtained Bayer index matrices are then assigned a threshold value that is between 0-255 based on the likelihood of pixel activation. The likelihood of pixel activation is given by the Bayer index matrices where 0 indicates the pixel most likely to be turned on. These thresholds are calculated by using the formula given in the problem statement.

We then loop a N x N window, where N is the size of the the threshold map, through the input image. If the input value is greater than the corresponding value in the threshold map, then we output the value 1. In other cases, we output value 0.

To generalize the above concept, we have implemented a design that could generate a display ready dithering for N intensity level. Since the above implementation holds good for only two colors ie. 0 and 1. In cases where we need to dither an image to display N intensity levels, we need a generalization.

To quantize an image to 4 quantization levels, we first calculate the four gray levels by dividing the dynamic range of the image into four equal parts. Since our dynamic range is 0-255, we have used {0,85,170,255} as the four quantization levels. Then we loop through the input input values and compare the value with the 4 levels. To obtain better levels, the output value is clipped to the next highest quantization level. For example, the output for grayscale value of 90 would be 170.

### **2.2.2 Error Diffusion based Dithering**

The quantization process in ordered dithering can result in blobs of uniform color if they are not treated properly. To avoid such quality loss, we diffuse the quantization error into the neighboring pixels. This is known as error diffusion or error dispersion. Error diffusion takes a smarter approach by spreading the quantization error to the neighboring pixels at a certain proportion. The proportion is different for different error diffusion algorithms.

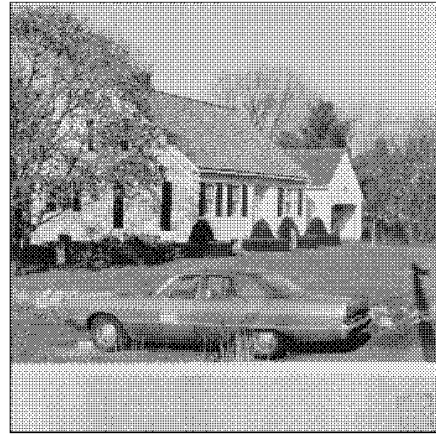
To implement this, we first initialize the error diffusion matrix. We have implemented Floyd-Steinberg, JJN and Stucki error diffusion matrices in this problem. If we observe the matrices, we can find that all the elements in the error diffusion matrices sum up to 1.

We then loop through the pixel values in serpentine manner. This can be done by reversing the column scanning in every alternate row. Each pixel in the image was binarized with a threshold value of 127. The data type 'double' was used to store the pixel information as it was expected that their value could go beyond the range that 'unsigned char' could handle. This was mainly because of error diffusing into the neighboring pixels. However, the output image consisted of 'unsigned char' type pixel values.

## 2.3 Experiment Result

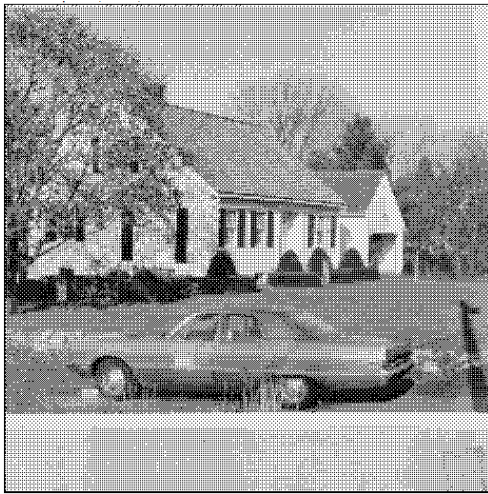


(a) 2x2 Bayer Matrix

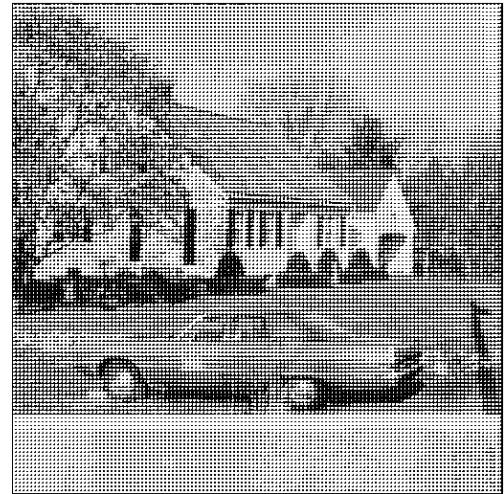


(b) 8x8 Bayer Matrix

Figure 6: Dithering Matrix based dithering



(a) 4x4 Bayer Matrix



(b) Given Bayer Matrix ( $A_4$ )

Figure 7: Dithering Matrix based dithering

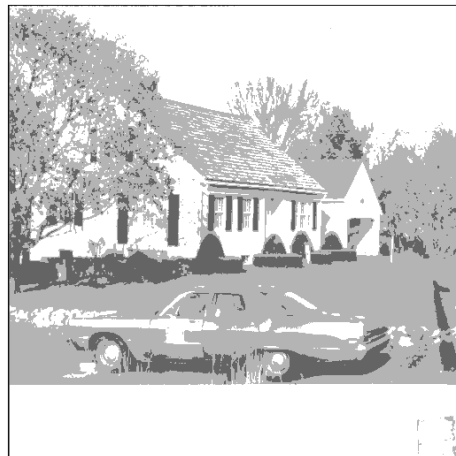
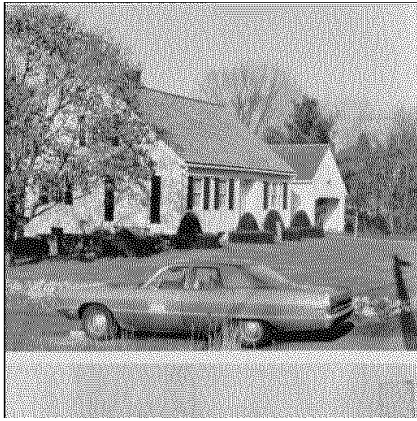
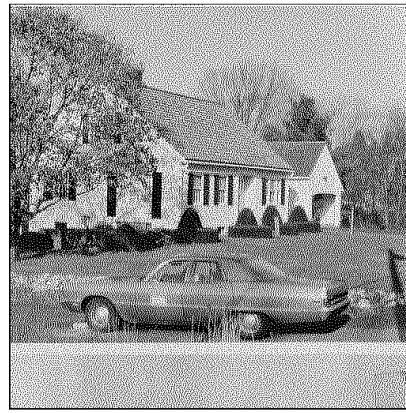


Figure 8: Four level dithering

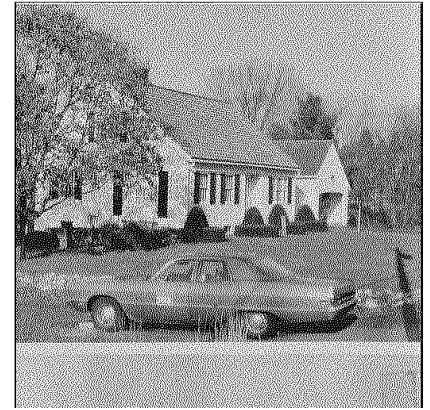




(a) Floyd-Steinberg



(b) Jarvis, Judice and Ninke



(c) Stucki

Figure 9: Error Diffusion based dithering

## 2.4 Discussion

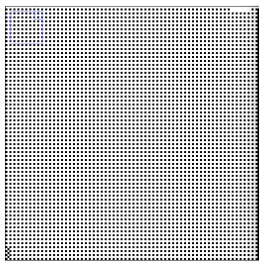
Eight different dithering methods were experimented in this problem. Figure 6 shows the output when 2x2 and 8x8 Bayer index matrices were applied. Please note that the subtitles in the output posted above may not be visible. Using the images directly produced by the code will make the differences more pronounced.

In figure 10, we can see how the Bayer matrix of different sizes distributes the quantization error for a group of pixels with same value. Upon closer inspection, as shown in figure 10, ordered dithering produces cross hatch pattern artifacts. Due to same patterned threshold mapping that occurs repeatedly throughout the image, it would create vertical, horizontal and diagonal lines producing cross hatch patterns. This is one of the drawbacks of ordered dithering. Apart from this drawback, its main advantage is speed and ease of implementation. Additionally, ordered dithering doesn't introduce any spatial distortion which is a big problem with other dithering techniques.

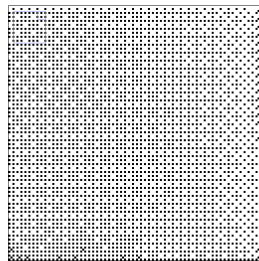
If we observe the  $A_4$  Bayer matrix, we find that the thresholding is of spiral pattern. It is evident when we look at figure 10 (c). These patterns repeat when the pixel values are uniform.

Figure 8 shows the output when the image is quantized to 4 levels. If we observe the image, we can find that the image looks more enhanced and the corner information is lost.

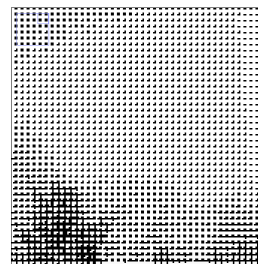
The output for error diffusion is shown in figure 9. Of all the half toning methods implemented, error diffusion generates the best results. Dispersion of quantization error over a larger area makes this algorithm more successful than others. Error diffusion method also displays a very pleasant randomness that doesn't give a sensation of rows and columns of dots. Ideally, a good error diffusion matrix produces a checkerboard pattern.



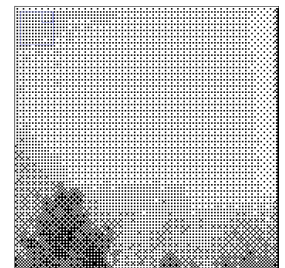
(a) 2x2 Bayer Matrix



(b) 8x8 Bayer Matrix



(c)  $A_4$  Bayer Matrix



(d) 4x4 Bayer Matrix

Figure 10: Zoomed images of dithered images

The values of Floyd Steinberg's error diffusion matrix were chosen such that they create an even checkerboard pattern. The output produced by applying JJN and Stucki error diffusion matrices were much better as compared to that of Floyd Steinberg because the error was distributed to three times as many pixels as the latter. The output of JJN and Stucki are smoother and much subtle.

The main problems faced by dithering algorithms is the presence of artifacts such as distracting band of speckles in areas with uniform values that produces grainy looking images. To reduce the grainy effect on areas with uniform values, we can propagate only a part of the quantization error to its neighbors. This technique is commonly referred to as "reduced color bleed".<sup>[4]</sup> This will result in less artifacts in areas with uniform values.

---

## Problem 3

### 3.1 Abstract and Motivation

Morphological image processing is a collection of techniques that are used for analyzing geometrical structures in an image. According to the dictionary, the word morph means “form” and ology- means “the study of”. These techniques rely on non linear operations that are related to shapes in an image. Morphological operations such as dilation, erosion, opening and closing utilize a small template known as structuring element to probe into an image. These operations can be combined and applied to basic algorithms for boundary extraction, hole filling, extraction of connected components, convex hull and so on.

In this problem we will discuss the implementation of rice grain inspection to define rice quality using morphological operations. Different thresholding algorithms were tested to obtain a perfect binary image. The final thresholding algorithm that gives us the perfect binarization is discussed in detail. Shrinking operation was used to reduce the grain to a single dot which helped us with counting the actual number of rice grains in the image.

Morphological operations are extensively used in shape classification. To extract shape information of objects in an image, a method for representing shape is required. Morphological functions like erosion, dilation and hit or miss are utilized carefully to provide solutions to hole filling and boundary smoothening problems. In this problem we will make an attempt to implement these solutions.

### 3.2 Approach and Procedures

#### 3.2.1 Rice Grain Inspection

To perform morphological operations, we need to convert a color image to a binary image. This is because morphological operations can be easily performed on binary images. In order to accomplish this, we first read the color image and extract the Luma component from the color image. This can be done by applying the following formula given in equation 4.

Different algorithms to convert from grayscale to binary have been tried out. Most of the binarization techniques failed because the objects in the image were either brighter or darker than the background. Any standard algorithm would perform badly as there was no means to separate the background from the foreground adaptively.

To achieve a high quality of binarization, we first split all the color channels and apply median filter of size 3x3 to each channel. This was done to make the boundaries smooth and reduce the impulsive noise in the image. Then a mode based binarization technique was applied. This technique has two steps to it. Firstly, it counts the mode pixel value of an image. In most cases, the mode value accurately represents the background pixel value. In the second step, we set the pixel values as 1 or 0 based on certain condition. With the knowledge of the background pixel value, we iterate through all the pixels and set the value to 0 if it lies within a certain radius around the mode value. If the pixel value lies away from the mode, then it is set to 1. The histogram of each channel is shown in figure 11 and, as expected, the mode value represents the background pixel value of each channel.

Selection of the radius was done through trial and error. The best value for radius was found to be 15. Later, we synthesize all channels with binary values using equation 4. Then a 5x5 median filter is applied to remove small sized hole in the binary image and smoothen the grain surface. Again, the values obtained after the synthesis is not binary, so we perform mode based binarization with a radius of 15 to the obtained synthesized image. This process gives us a smooth and clean binarized image as shown in figure 12. From this method, we could detect 55 rice grain.

Then thinning was applied to the binary image. To apply thinning, we first load the conditional and unconditional pattern tables in a static hash map. To reduce the run time we declare all the pattern table maps as static

as static variables are run only once. The pattern is searched for in the static hash map and if it finds a match it returns the values 1. If the pattern is not matched, then it returns the value 0.

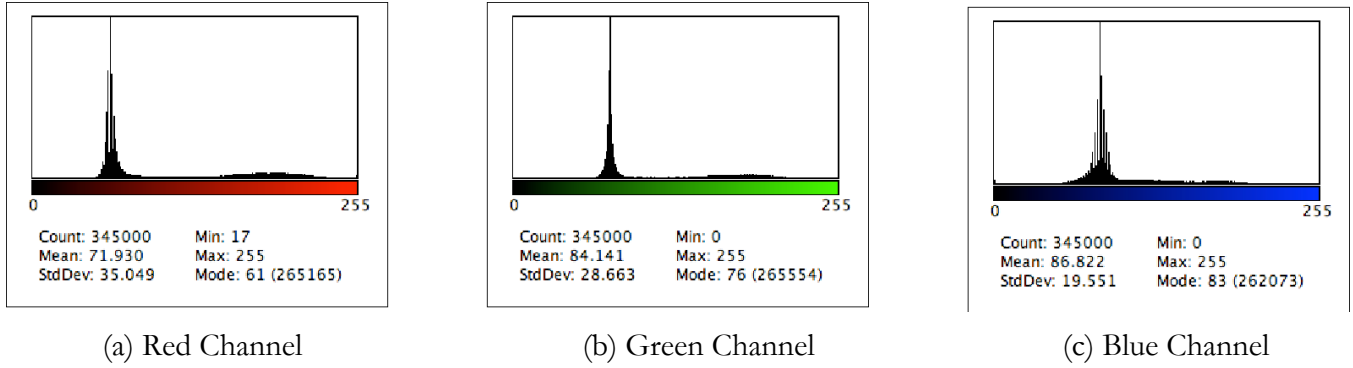


Figure 11: Histogram of rice image

### 3.2.2 MPEG 7 Dataset

The MPEG 7 data set consists of 2D shapes that can be utilized to train a classifier. In this problem we will implement pre-processing steps required to obtain accurate shape information of the binary images. Upon inspection, the necessary pre processing steps required for accurate morphological processing are hole filling and boundary smoothing.

Holes in an object can produce incorrect results when morphological operations are performed. This is because holes in the object tend to produce incorrect hit or miss result. However, this problem can be solved by implementing a very simple hole filling algorithm. This is effective done by utilizing the following formula:

$$X_k = (X_{k-1} \oplus B) \cap A^c \text{ -----(Equation 11)}$$

The output  $X_k$  is recursively found until  $X_k = X_{k-1}$  where  $k = 1, 2, 3$  and so on. The previous output is dilated with a structuring element  $B$ . The dilation is then intersected with  $A$  complement that would limit the result to inside the target region. For this purpose, the structuring element  $B$  was chosen to be:

$$B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

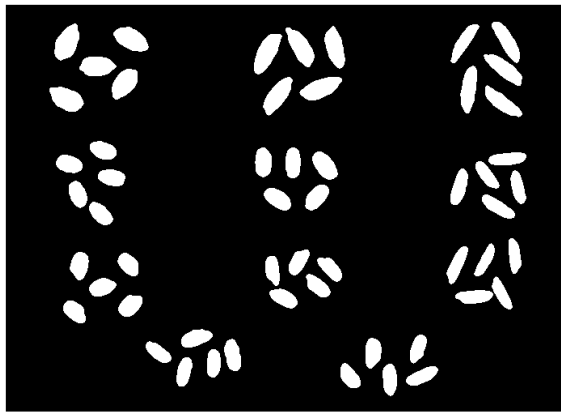
We set the pixel value at 0,0 to start with in this dilation process. The main idea behind this is to dilate the background and find the missing holes. After the background is dilated the following logical operation is performed that will result in the hole locations:

$$H = \overline{A \cup X_k} \text{ -----(Equation 12)}$$

The hole locations  $H$  is calculated by taking the complement of the union of input and dilated images. The hole locations are then filled by taking the union of  $H$  with the input. Dilation and erosion was performed multiple times in cascade to get a smooth boundary. The output for these is showed in the next section.



### 3.3 Experiment Results



(a) Thresholding output

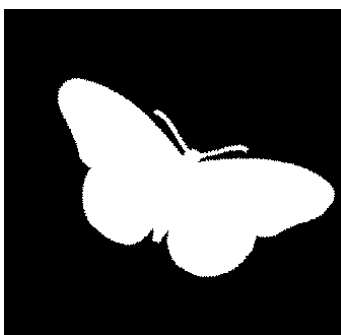


(b) Shrinking output

Figure 12: Applying morphological operations on rice image



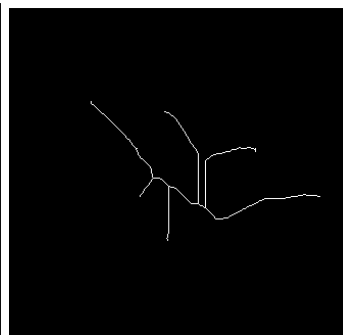
Figure 13: Thinning on rice image



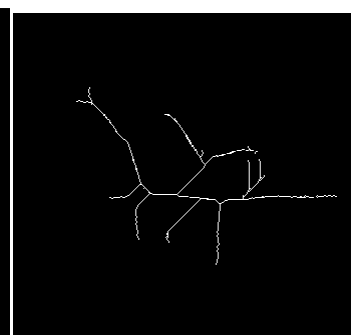
(a) Hole Filling



(b) Boundary Smoothen



(c) Thinning



(c) Skeletonize

Figure 14: Morphological operations on butterfly image



(a) Hole Filling

(b) Boundary Smoothen

(c) Thinning

(c) Skeletonize

Figure 15: Morphological operations on fly image



(a) Hole Filling

(b) Boundary Smoothen

(c) Thinning

(c) Skeletonize

Figure 16: Morphological operations on probe image

### 3.4 Discussion

Firstly, we should discuss about the different thresholding algorithms and the reason why it failed. As we see in figure 17, we can see major flaws in all the implemented thresholding algorithms. The main reason why it failed is because it could mask the background from the foreground. Due to which we can see the bottom left portion of the rice grains missing in figure 17 (a) and figure 17(c). Other methods consider windows to determine local threshold rather than global threshold. However the objects darker than the foreground were incorrectly binarized. The output for Niblack's thresholding suffers due to background noise. This can be seen in figure 17 (b)



(a) Otsu's (0.48)

(b) Niblack

(c) Fixed Threshold

(d) Adaptive Mean

(c) Adaptive Gaussian

Figure 17: Thresholding algorithms on rice

The comparison of rice grains can be made by first applying connected component labelling. Each grain in the binary image would correspond to a digit between 1 and 56. We would then proceed to count the number of such digits occurring in the image and store it in a hash map with the grain number as key and label count as its value. This would give us the grain size in terms of number of pixels. Larger grains would have larger values and smaller grains would have less number of pixels. We would then proceed with calculating the average of each type of rice grain and sort it. This would give us the relative size comparison of each grain type.

The rice grain can be categorized based on their physical parameters such as length, area and color. Morphological feature can be extracted and used for training a simple classifier. The following features can be useful to consider while classifying rice grains: <sup>[5]</sup>

1. **Area:** This calculation can be made by counting the number of pixels in the connected component labelled image.
2. **Major Axis length:** This could be found by applying thinning and connected component labelling. The number of pixels under each label gives us the value for major axis length.
3. **Minor Axis length:** This could be found by computing the minimum bounding boxes for the binarized image. The bounding boxes would enclose the grain and the breath of such bounding box would give us the value for minor axis length.
4. **Eccentricity:** This gives us a measure of how circular a rice grain is. This can be calculated by taking the ration of distance between the foci and its major axis length.
5. **Perimeter:** This can be computed by applying canny edge filter.

These features provide a rich insight about the grain types.

In figures 14, 15 and 16 we can see the effect of different morphological operations. For the butterfly image the hole filled image is dilated two time and eroded one time. This is done by trail and error. Hole filling and boundary smoothing filters are really important in morphological image processing. If we incorrectly perform them then we might get incorrect results for shrinking, thinning or skeletonizing.

The skeletonizing images for butterfly and fly gives a distinctive shape that could be used for classification. It also reduces unimportant features that could cause overfitting of the classifier. To classify unknown shape, we first need to devise a feature – feature similarity measure. We can keep each intersection in the skeletonizing output as a node and develop a graph. We can then compute a graph similarity score based on a graph similarity algorithm. The score can be computed for each label and compared.

In figure 16(a) and figure 16(b) we can observe that morphological closing operation has been successfully performed by a series of dilation and erosion.

## References :

- [1] Internet Source: [http://graphics.cs.cmu.edu/courses/15-463/2011\\_fall/Lectures/morphing.pdf](http://graphics.cs.cmu.edu/courses/15-463/2011_fall/Lectures/morphing.pdf)
- [2] Chris Harris & Mike Stephens, “A combined corner and edge detector”, UK, The Plessey Company, 1988
- [3] Internet Source: <http://www.tannerhelland.com/4660/dithering-eleven-algorithms-source-code/>
- [4] Internet Source: <http://www.tannerhelland.com/4660/dithering-eleven-algorithms-source-code/>
- [5] Rubi Kambo, Amit Yerpude, “Classification of Basmati Rice Grain Variety using Image Processing and Principal Component Analysis”, IJCTT, volume 11, number 2, May 2014