UNIVERSITY OF SOUTHERN CALIFORNIA

EE569 : HOMEWORK 3

November 1, 2015

Name : Taruna Agrawal

Email : tagrawal@usc.edu

Student ID: 7650184685

Contents

PROBLEM 1 : GEOMETRICAL MODIFICATION

Problem 1a : Swirl Effect

Abstract and Motivation

Geometric modification is transformation of position of pixels in a image to create special effects or to morph one image into another. Translation, rotation and scaling combination refers to geometrical transformation. The motivation of this problem is to provide a swirl effect to given image and display the resulting figure.

Approach and Procedure

Following is the approach and procedure followed for swirl effect:
Step 1: Convert the image from image coordinate to Cartesian coordinate with center of image as origin.
Step2: Use the below given formula to calculate the angle of rotation:

$$angle = a * ((X - X0) * (X - X0) + (Y - Y0) * (Y - Y0))$$

where a is amount of rotation (set as 0.0059), X and Y are coordinates of image and X0, Y0 are center coordinates.
Step3: Calculate the new transformed coordinates:

$$u = cos(angle) * x + sin(angle) * y$$

$$v = -sin(angle) * x + cos(angle) * y$$

Step 4: Do inverse transformation to find coordinates in image coordinate system or original image coordinates.
Step 5: Perform bilinear interpolation on the coordinates obtained in step 4 to obtain the pixel values.
Step 5:Display the output.

Experimental Results

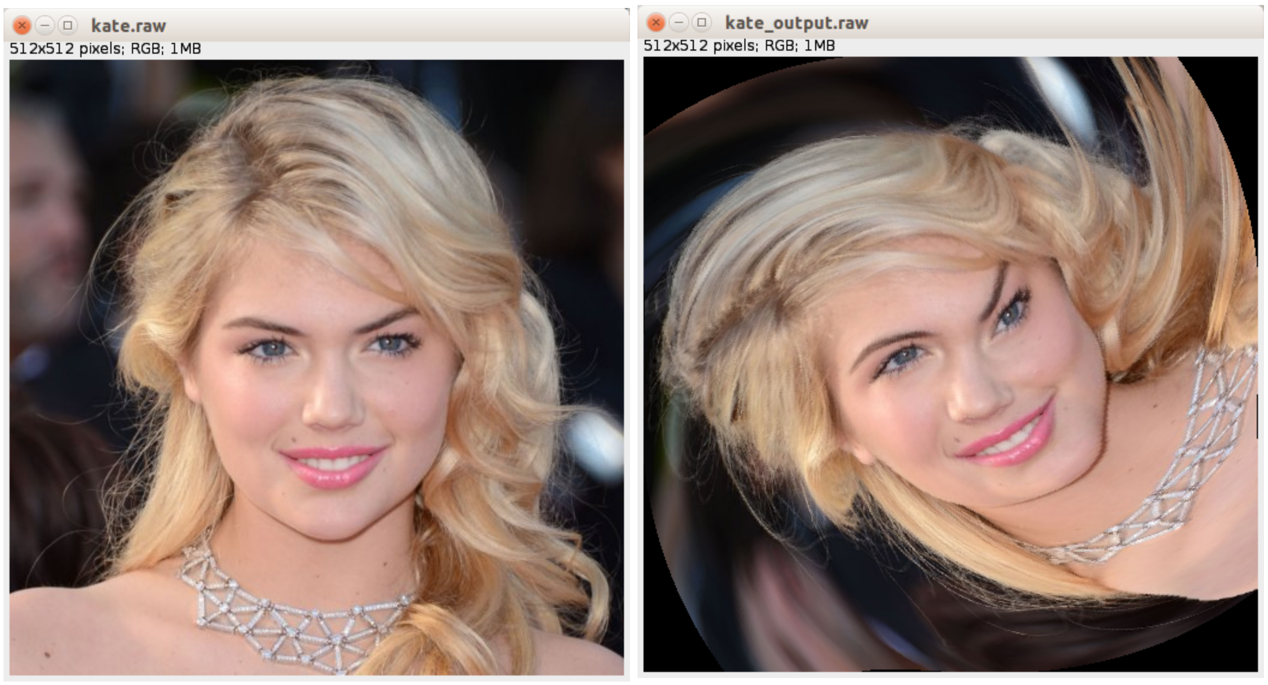


Figure 1: (left) Original Image, (right) Kate with Swirl Effect

Discussion

Swirl is essentially a rotation, where the angle of rotation depends upon the distance from the center of the image.The key idea is the swirl effect is to taken coordinates of distorted image and map it to original image coordinates. In this problem the amount of rotation increases as we move towards the edge of image, but there can be other type of swirl effect in which amount of rotation decreases as we move towards the edge. In the resultant image at the corners we see black pixels, these pixels signify the coordinates which after transformation were found to lie outside the actual image boundary and hence their value was set as zero.

Problem 1b : Perspective Transformation and Imaging Geometry

Abstract and Motivation

The goal of this problem is to use perspective transformation and imaging geometry to capture image in 3D world coordinates and project it onto a 2D image plane. We have been provided six images of dimension 200*200 and as part of this problem we have been asked to display three images on three sides of the cube using perspective transformation and Imaging Geometry. The details of the implementation are explained in next section.

Approach and Procedure

Pre-Processing

Step1: Read any three images of given dimension.

Step2: Map the coordinates of image to the coordinates of the cube. For example mapping to the side 1 of the cube we can use the below formula.

$$x[i][j] = -1 + (2 \div width) * i$$

$$y[i][j] = -1 + (2 \div width) * j$$

$$z = 1$$

Similarly it can be done for side 2 and side 3. In Figure 3, I have shown the results for mapping after pre-processing. These results are only for the corners of the cube corresponding to side 1,2 and 3.

Processing

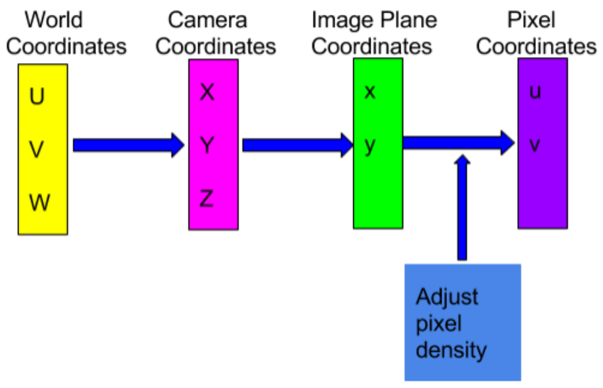


Figure 2: Block diagram for the projection method.

As we can see from block diagram first step is to obtain World coordinates, which we already got as part of pre-processing. Next step following steps need to be followed:
Step 1: Obtain Camera coordinates from world coordinates using the matrix given below:

$$[R|t] = \begin{bmatrix} -1 \div \sqrt{2} & 1 \div \sqrt{2} & 0 & 0 \\ 1 \div \sqrt{6} & 1 \div \sqrt{6} & -2 \div \sqrt{6} & 0 \\ -1 \div \sqrt{3} & -1 \div \sqrt{3} & -1 \div \sqrt{3} & 15 \div \sqrt{3} \end{bmatrix}$$

$$\begin{bmatrix} X & Y & Z \end{bmatrix}^T = [R|t] * \begin{bmatrix} U & V & W \end{bmatrix}^T$$

Step2: Now obtain the Image plane coordinates x and y. This step maps from 3 dimensional to 2 Dimensional.

$$x = f * X \div Z$$

$$y = f * Y \div Z$$

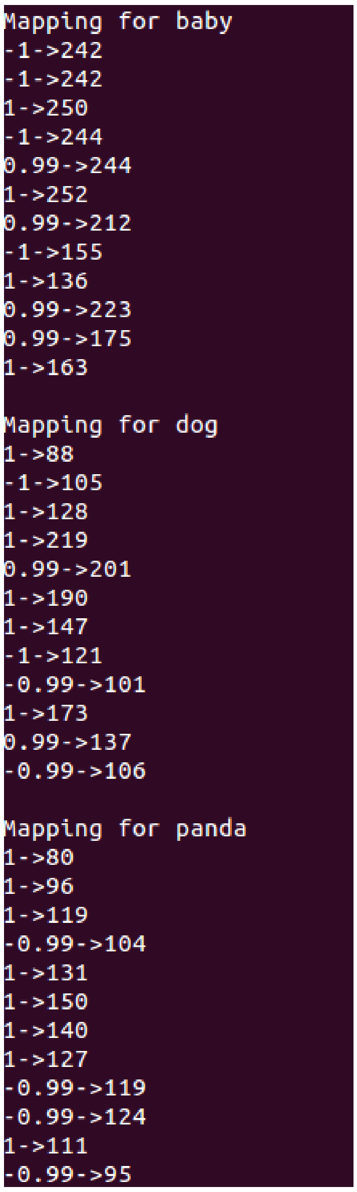
where f is focal length and value used as part of this problem is $\sqrt{3}$. X and Y are the values obtained as part of step 1.
Step 3: Now from the resultant values of x and y it can be seen that most of the values are concentrated around origin with very small range. Now in order to display this in 200*200 cube , this x and y should be multiplied by pixel density and width $\div 2$ should be added in order to adjust negative pixel coordinates.

$$u = x * 255 + (width \div 2)$$

$$v = y * 255 + (width \div 2)$$

This value of 255 was concluded after several experiments.
Step4: Now since we have got u and v value for the pixel coordinates of cube, fill these pixel coordinates with corresponding image pixel values.
Step 5: Display the resultant image after repeating step 1 to step 4, including pre-processing for other two images.

Experimental Results



```
Mapping for baby
-1->242
-1->242
1->250
-1->244
0.99->244
1->252
0.99->212
-1->155
1->136
0.99->223
0.99->175
1->163

Mapping for dog
1->88
-1->105
1->128
1->219
0.99->201
1->190
1->147
-1->121
-0.99->101
1->173
0.99->137
-0.99->106

Mapping for panda
1->80
1->96
1->119
-0.99->104
1->131
1->150
1->140
1->127
-0.99->119
-0.99->124
1->111
-0.99->95
```

Figure 3: Left side shows cube coordinates and right hand side shows corresponding red, green and blue value for respective images.

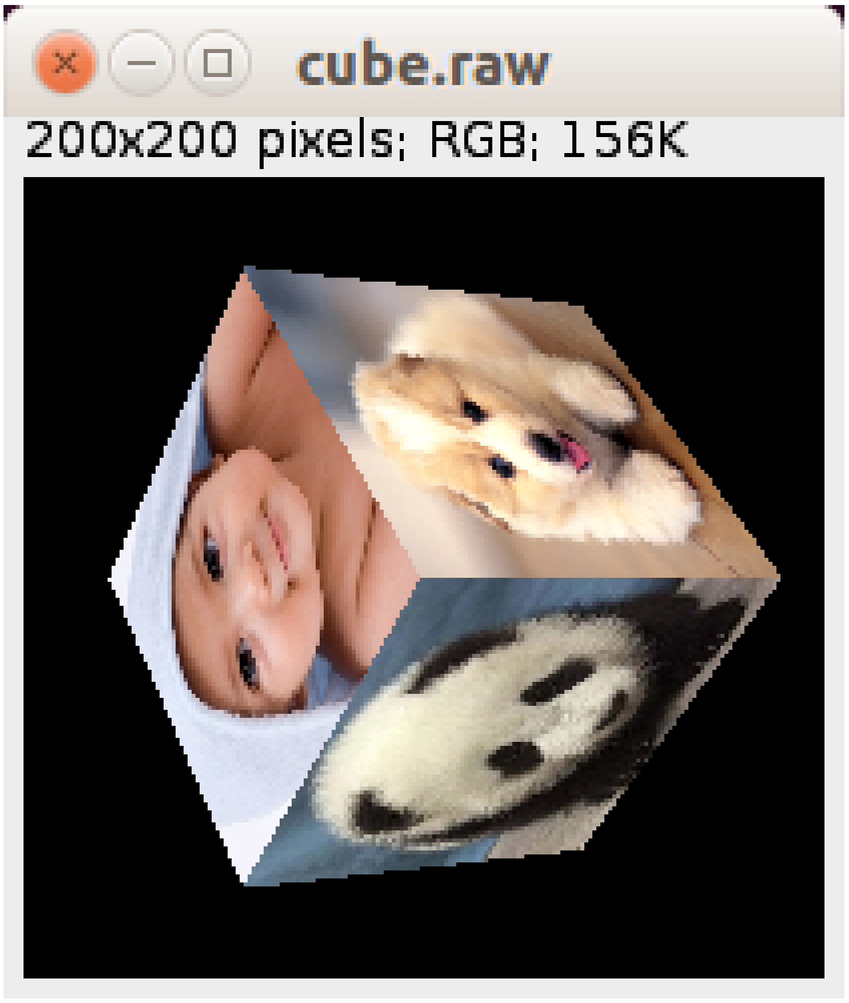


Figure 4: Resultant image of cube.

Discussion

Figure 4 shows the result obtained for forward mapping using perspective transformation and Imaging geometry. I believe the challenging part of the problem was to find the right pixel density for the problem. The result shown above is obtained with pixel density of 255. This value of 255 was reached after several experiments. I decided on this value by finding the range for x and y image plane coordinate. Say, for example we are looking at image plane coordinates obtained for side 1. Maximum value of x is -0.301 and +0.301 , also if we look at the pattern of x values obtained for each row , we will see that values decrease with each row which is expected if we look at the diagram of cube for side 1. Based on minimum and maximum values of x, I tried various pixel density values , for example 100, 200, 300, 350, 400, but as the values were increased artifacts were observed on the image. This is due to the fact that image is being scaled directly by multiplying. In my opinion we can improve the result

obtained if we use bilinear interpolation.
Also, at the edges of the resultant images we see some artifacts. The edges obtained are not very sharp and it has zigzag pattern. This can be due to the fact that image plane coordinates are floating point value and scaling them and rounding it off to nearest integer might lead to some pixels at width of image taking background pixel values (i.e. 0).

Question : Is there any way to improve the result? If so, describe your own algorithm.
We can improve the above algorithm by using bilinear interpolation during mapping from image plane coordinates to resultant image coordinates i.e. while obtaining u, v coordinates.
Following can be steps for the same:
Step 1: Find ratio for row by dividing cube height with range of combined x for all three images. Similarly find ratio for column.
Step2: u , v will be equal to ratio of row*u and ratio of col*v respectively.
Step3: Calculate absolute value of u and v and subtract this value from original u and v to get a and b respectively.
Step4: Now apply the interpolation formula to get the new cube coordinates.
Step5: Fill the pixel value with corresponding image coordinates.

PROBLEM 2 : DIGITAL HALFTONING

Problem 2a : Dithering

Abstract and Motivation

Dithering can be described as a technique in which a computer program tries to approximate the color from a mixture of colors when there are limited number of colors available. As part of this problem we have been provided 8 bit gray level image which needs to be converted to 2 bit image using dithering method. The main motivation behind this idea is to save the number of colors used in generating the image. As we all know that in order to print a gray level image with all the possible colors in it, printer might need large amount of different colors, thus increasing the cost of printing. Digital half toning is a technique which uses only one color that is black to print the images.These black dots are intelligently put on paper in order to create various shades of gray. There are various algorithms to decide the order in which the dots need to be placed on paper. In this section we will discuss about the dithering algorithm.

Approach and Procedure

Experiments were done to generate image with 2 and 4 intensity levels. Dithering matrix of size 4 and 8 is used for the same.
Following are the steps involved in dithering method with intensity level 2.
Step 1: Generate a dithering matrix using the following index matrix. In this problem we will use size 4 and size 8 matrix.

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

Above index matrix can be used to generate the size 4 and size 8 matrix using the below recursion formula.

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

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

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

Step2: Calculate the threshold using the calculated matrix of size 4 and 8. Below formula can be used to calculate the threshold.

$$T(x,y) = (I(x,y)+0.5)/N^2$$

Step3: Normalize the image and check if value of pixel is greater than threshold than output of final image is 1 else take the final value as 0.

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

Step4: Repeat step 1 and step 2 for whole image.

For creating the 4 intensity level image with gray levels 0, 85, 170, 255 follow the same steps as described above but modify step 3 as follows:
Step5: Normalize the image and check the value of pixel and assign the output value based upon below cases.

$$G(i,j) = \begin{cases} 1 & \text{if}, ifF(i,j) >= 1.5 * T(x,y) \\ 0.66 & \text{if}, ifF(i,j) >= T(x,y) \\ 0.33 & \text{if}, ifF(i,j) >= 0.75 * T(x,y) \\ 0 & \text{otherwise} \end{cases}$$

Experimental Results

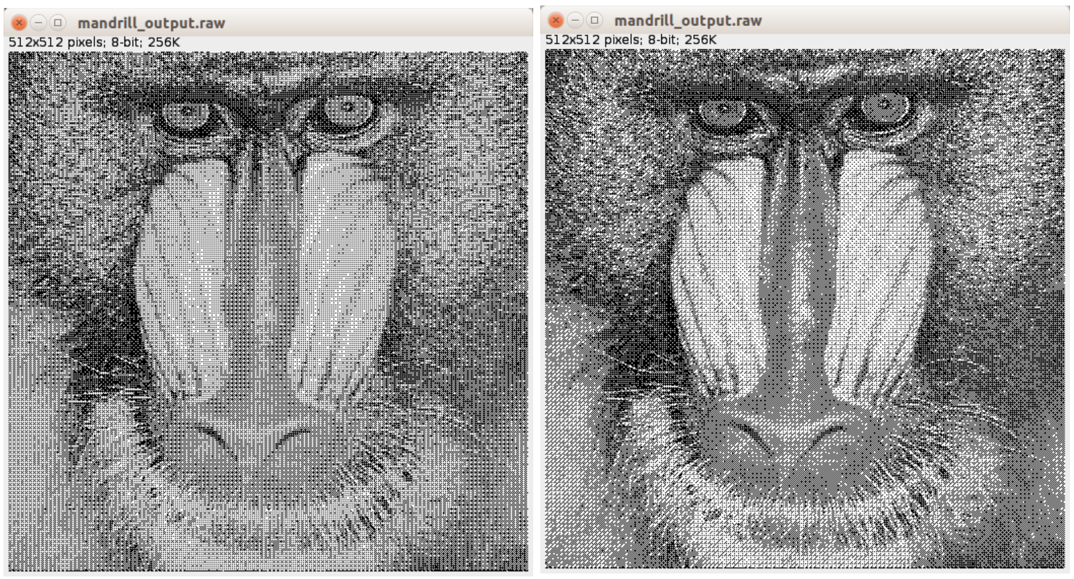


Figure 5: (left figure) Dithering image with matrix size 4 and intensity 2, (right figure) Dithering image with matrix size 8 and intensity 2

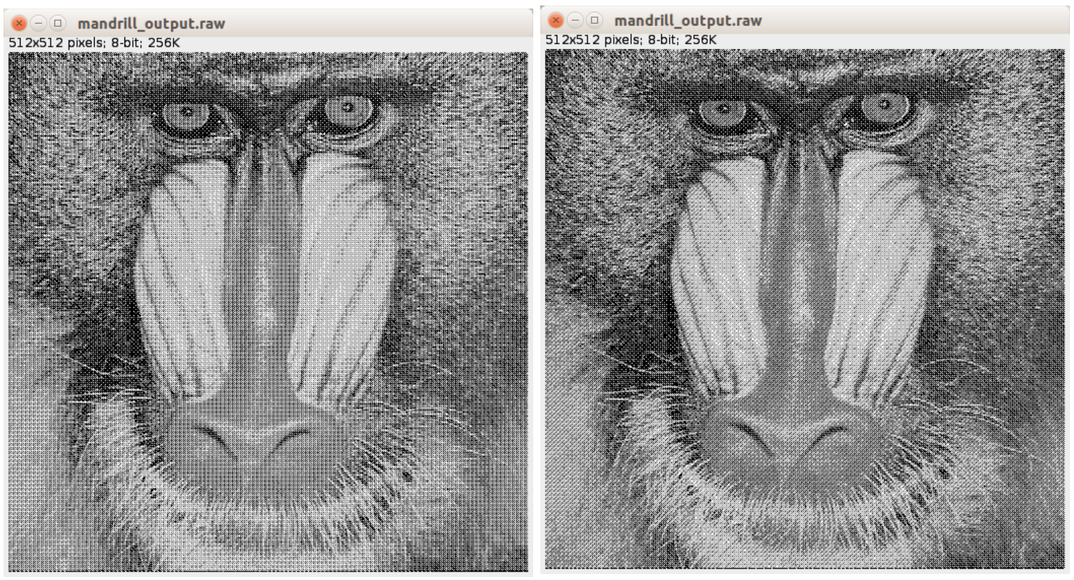


Figure 6: (left figure) Dithering image with matrix size 4 and intensity 4, (right figure) Dithering image with matrix size 8 and intensity 4

Discussion

As we can see from figure 5 and figure 6 above that as the matrix size increases the result gets better. As the matrix gets bigger there are fine variations and involves more details. Figure 7 shows the zoomed in version of mandrill nose.If we look carefully at the image it shows an undesired black dot patterns which get introduced in the image if we use dithering matrix. This effect reduces as we increase the matrix size and number of intensity levels. The reason for the improvement moving from left figure to right is because of the fact that larger matrices have more specified threshold. If the threshold is more specified then more black and white pixels can be juxtaposed and hence can create an illusion with more levels of gray colors.

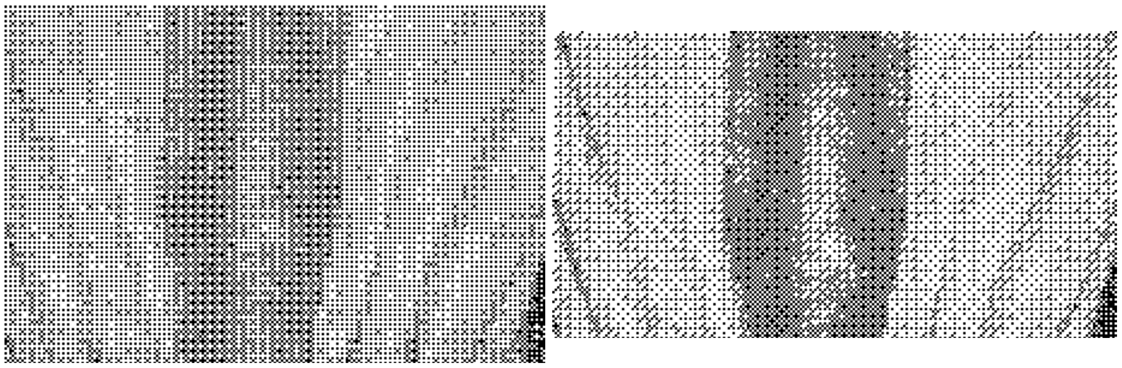


Figure 7: (left figure) I_4 and intensity 2, (right figure) I_8 with intensity 2

The procedure used to generate image with four intensity levels is discussed in approach and procedure section. From figure 6 we can clearly see that results are much better as compared to Figure 5. Image with four intensity levels contain very less black/white dots around the nose region and hair region. Also we can see that final image is very close to original image. The reason for the improvement in image is due to the fact that more number of intensity levels are being used and hence more diverse gray level illusion can be generated. Also, in next section we can see that better results are obtained with binary image if we use error diffusion method instead of dithering method.

Problem 2b : Error Diffusion

Abstract and Motivation

Another method that can be used for digital halftoning is Error Diffusion method. This method gives much better results as compared to dithering method. The main idea behind this method is to distribute the quantization error introduced in current pixel to its neighboring pixels. In this section we will be using Floyd Steinberg, Jarvis et. al. , Stucki methods to obtain half toned images. All the three algorithms basic idea is to use fixed thresholding but diffuse the error difference between quantized pixel and original pixel to the neighbouring pixels which have not been quantized yet. Next section explains in more detail about the algorithm.

Approach and Procedure

Below is the block diagram for error diffusion method. F is input image, G is output image. Threshold is fixed to 0.5, for pixel value between 0 and 1.

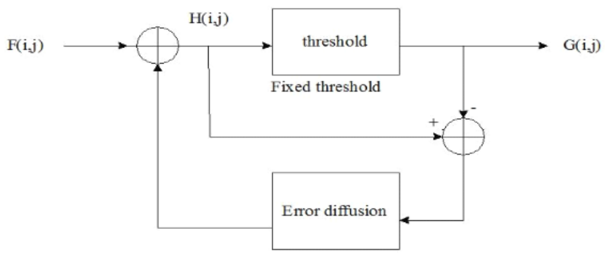


Figure 8: Error diffusion block diagram

We are also using serpentine method to diffuse the error, which uses left to right and right to left method to diffuse the error.
For all odd lines use left to right diffusion matrix

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

For all even lines use right to left diffusion matrix

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

Step by step procedure for error diffusion method is described below.
Step 1: Calculate the error for each pixel using the threshold value of 0.5. Step 2: Use the Floyd Steinberg's, Jarvis, Judice and Ninke or Stucki Error diffusion matrix to distribute colors to future pixels.

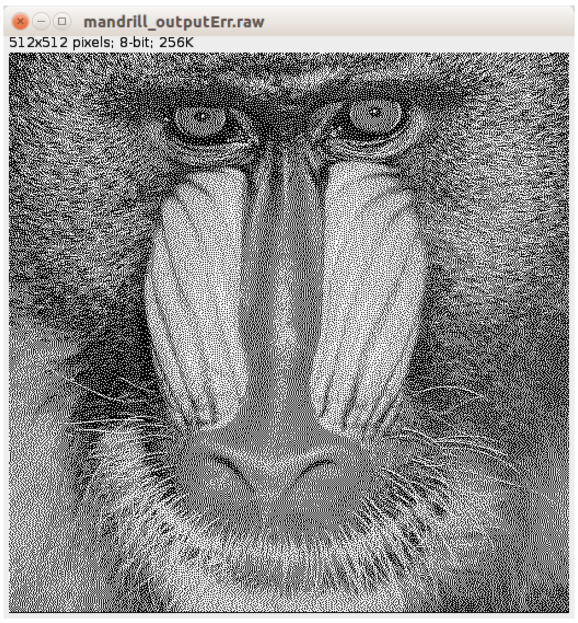
$$1/16 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix} \qquad Flyod - Steinberg$$

$$1/48 \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix} \qquad JJN$$

$$1/42 \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix} \qquad Stucki$$

Step 3: Repeat the procedure for whole image in serpentine manner that is left to right and then right to left.
Three images are generated based upon the three matrices. Results sections shows the result for all three different methods.

Experimental Results



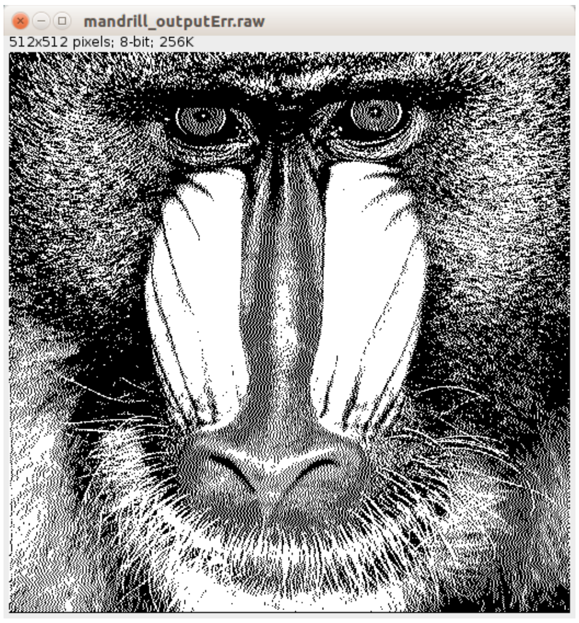


Figure 9: (left figure) Error diffusion with Steinberg method, (right figure) Error diffusion with Jarvis et. al. method

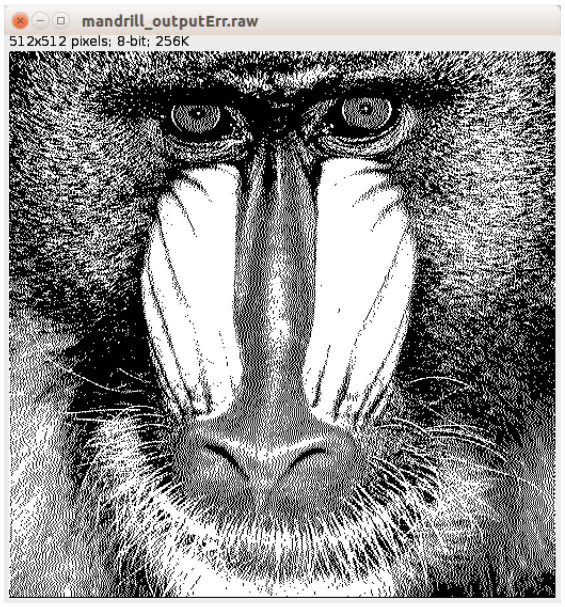


Figure 10: Error diffusion with Stucki method

Discussion

From a human eye point of view it can be clearly seen that images obtained with JJN and Stucki method are much better than the one obtained with Steinberg method and also the image we obtained using dithering matrix discussed in section 2a. If we look at the images carefully we see that around the nose region the output obtained with JJN and Stucki method is much better, also the contrast of image obtained is better with JJN and Stucki method. Since in the above two methods error gets accumulated at the corner pixels therefore we use serpentine scanning due to which error propagates from left to right and right to left. One drawback of these methods is fixed thresholding, say if pixel value is 126 then it gets mapped to 0 and the error gets distributed to its neighbor, due to which neighbor that falls in one region might move to another region due to which we see black dots in the resultant image. As per results it can be seen that this behavior is more prevalent in Steinberg method as compared to other methods. As we can see from figure 11 below area around the nose contains lot of black dots in case of Steinberg method due to which image appears noisy, whereas in case of right figure we see that image is more sharp.

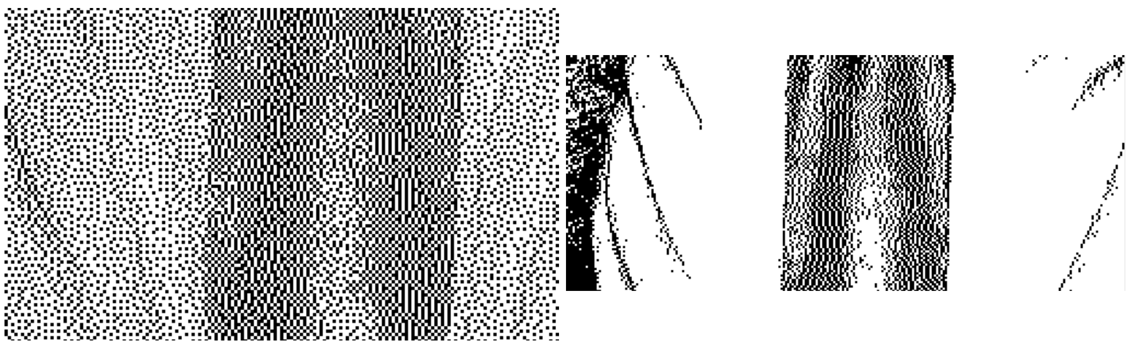


Figure 11: (left figure) Black dot pattern with Steinberg method, (right figure) and Jarvis et. al. method

One way in which we can improve existing algorithm is by using adaptive thresholding instead of fixed thresholding.

Problem 2c : Scalar Color Halftoning

Abstract and Motivation

Scalar color halftoning techniques is same as error diffusion method but instead it is used in colored images. Color error diffusion is mainly used for rendering colored images on devices with limited number of color palettes. These devices can be low cost displays or printers. In this assignment we are given the colored raw image and have been asked to perform scalar and vector halftoning. In next section I have described in detail the procedure followed for scalar halftoning.

Approach and Procedure

This is very simple method which involves converting each RGB channel to CMY and then apply error diffusion on each channel separately . After applying error diffusion to all the three channel convert the image back to RGB for viewing purpose.
Step by step procedure:
Step1: Convert RGB image to CMY by taking inverse of image.

$$C = 255 - R$$

$$M = 255 - G$$

$$Y = 255 - B$$

Step2: Perform error diffusion on each channel using Flyod Steinberg matrix as described in section 2b.
Step3: Convert back to RGB.
Step4: Display the image.

Experimental Results

The result of scalar halftoning

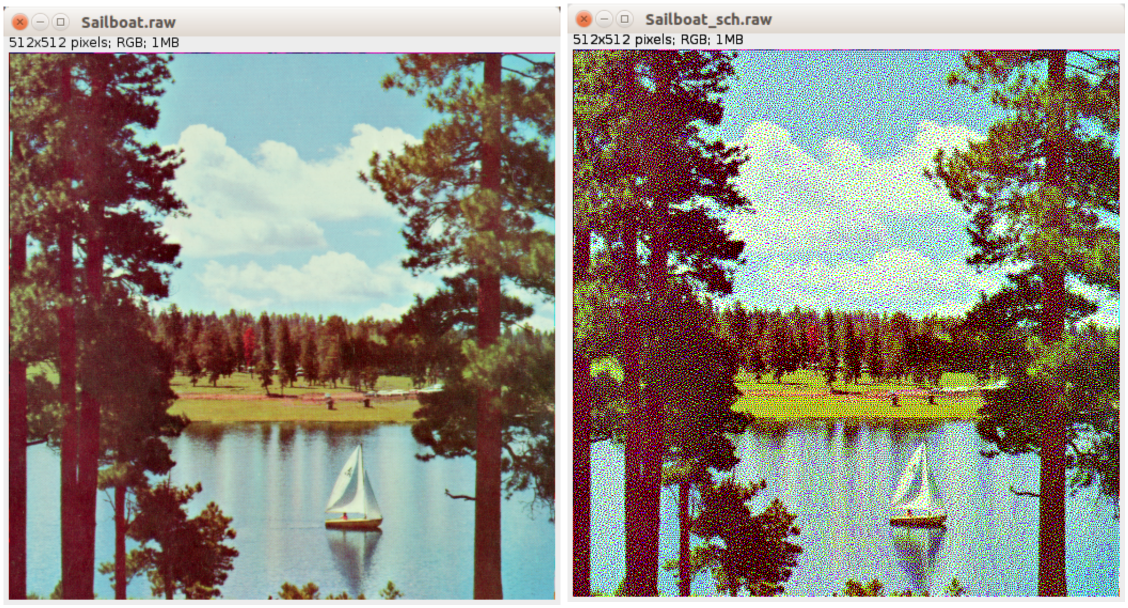


Figure 12: (left) Original Image, (right) Scalar half toned image

Discussion

As we can see from the result scalar halftoning results in lot of color artifacts and poor color rendition since it does not exploit the correlation between color which is the basic idea behind color perception and halftoning quality.
Below image shows the low frequency noise like patterns which are seen in scalar halftoning image.

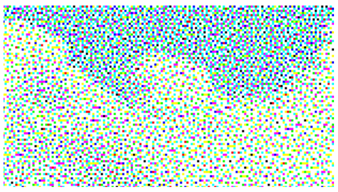


Figure 13: Noise like pattern in scalar halftoning result

In order to overcome this shortcoming of scalar halftoning vector halftoning was introduced which we will discuss in next section.

Problem 2d : Vector Color Halftoning

Abstract and Motivation

Vector error diffusion was introduced to solve the problem of scalar error diffusion. The main difference between the two procedure is the color space used for halftoning process. During quantization, vector error diffusion treats the three channels as a vector and then carry out the process instead of carrying out quantization on each channel separately.
Vector error diffusion uses eight primary colors(Cyan, Magenta, Yellow, Black, Red, Green, Blue and White) to perform the error diffusion process. The minimum distance between the eight primary color patches and the error-corrected vector is chosen and this primary color patch then becomes the output image. I have used euclidean distance to find the distance between color patch and error corrected vector. Figure 10 below shows the block diagram explaining the process involved with vector color halftoning.

Approach and Procedure

Figure 14 below shows the flowchart used in vector error diffusion method:

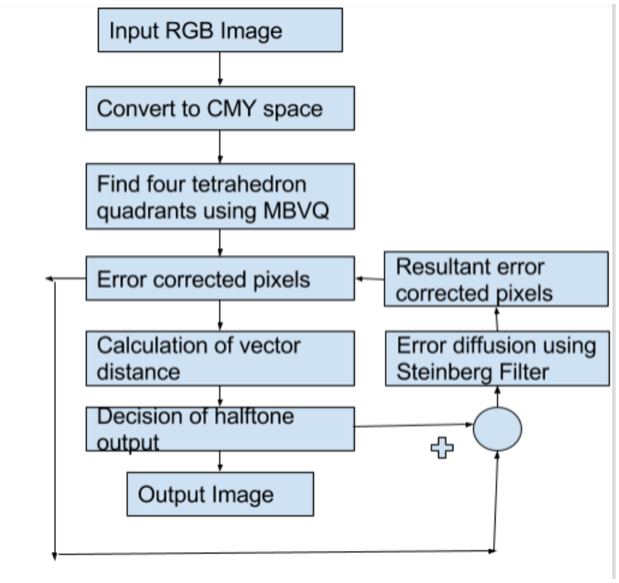

Input RGB Image
Convert to CMY space
Find four tetrahedron quadrants using MBVQ
Error corrected pixels
Resultant error corrected pixels
Calculation of vector distance
Error diffusion using Steinberg Filter
Decision of halftone output
Output Image

Figure 14: Flowchart for vector error diffusion

Above flowchart explains in detail the steps taken to perform vector error diffusion. For calculating the MBVQ quadrant following algorithm should be used[2].

```
pyramid MBVQ(BYTE R, BYTE G, BYTE B)
{
  if((R+G) > 255)
    if((G+B) > 255)
       if((R+G+B) > 510)     return CMYW;
       else                  return MYGC;
     else                    return RGMY;
   else
     if(!((G+B) > 255))
       if(!((R+G+B) > 255)) return KRGB;
       else                  return RGBM;
     else                    return CMGB;
 }
```

Figure 15: MBVQ algorithm

Experimental Results

The result of Vector halftoning

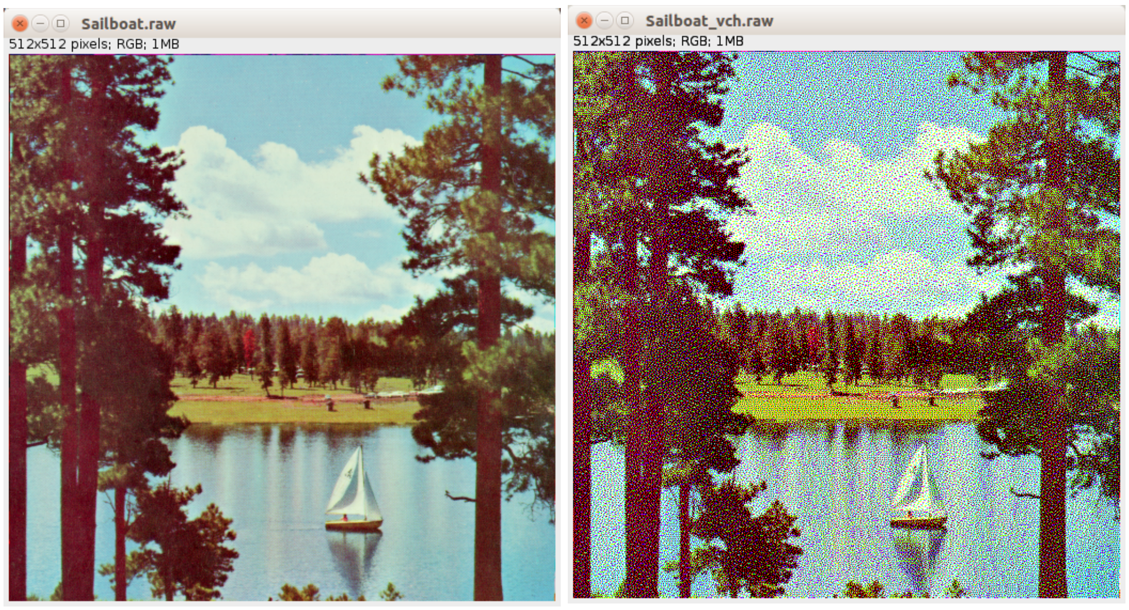


Figure 16: (left) Original Image, (right) Vector half toned image

Discussion

As per result obtained in vector half toning we don't see much difference between scalar and vector results. However, since this technique is mainly used for printing it is expected that the printing result for vector halftoning will be much better than scalar halftoning.
Vector halftoning method used above limits the number of ink colors which are used to render a specific pixel. As per authors argument by limiting the number of colors, a smaller range of brightness in the colors are used to create each color area, which leads to minimization

of the visibility of halftone patterns. This is known as Minimum brightness variation criterion(MBVC) and resulting halftone quadruple is known as Minimum brightness variation quadruple(MBVQ). This criteria was developed based on the observation that human eye is more sensitive to luminance than chrominance.
The main advantage of vector halftoning over scalar halftoning is that the halftone noise reduces to considerable amount and also the halftone pattern is more regular since vector halftone uses only four colors instead of all eight.

PROBLEM 3 : MORPHOLOGICAL PROCESSING

Problem 3a : Shrinking

Abstract and Motivation

Morphological processing is widely used in the fields which require the study of shapes and forms of image.For example finger print recognition, word recognition etc. As part of problem 3 we were asked to implement shrinking, thinning and Skeletonizing of given binary images. In next few sections I have discussed the procedure and result obtained.

Approach and Procedure

In this part of problem we have been asked to perform shrinking on horseshoe and count the number of holes, number of nails and number of white objects in the horse shoe image. This problem has two parts.
Part 1: Implementation to count number of holes(black circle) and number of nails(white circle) in the image.
Step1: For the given image set pixel of interest as 1 i.e white pixel.
Step2: Create an array M with same dimension as original image.
Step3: Scan the image with 3*3 window.
If X(i,j) = 0, then M(i,j) = 0;
else if X(i,j) = 1, then M(i,j) = 1;

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

Figure 17: [Bound $= 2 * (X0 + X2 + X4 + X6) + 1 * (X1 + X3 + X5 + X7)$]

Calculate the bound of image.
Step4: If bound is 0 or 12, set M(i,j) as 0.
Step5: if bound lies between 0 and 11, compare binary image string pattern X0-X7 with all the patterns given in reference table[4].As part of my implementation I am storing the values in bit format and then I check if binary string pattern in image is same as pattern in the reference table.
Step6: If there is pattern match set M(i,j) as else set M(i,j) as 0.
Step7: Once M array is complete , scan the whole M array with a 3*3 window.
If M(i,j) = 0, then M(i,j) = F(i,j) where F(i,j) = output Image;

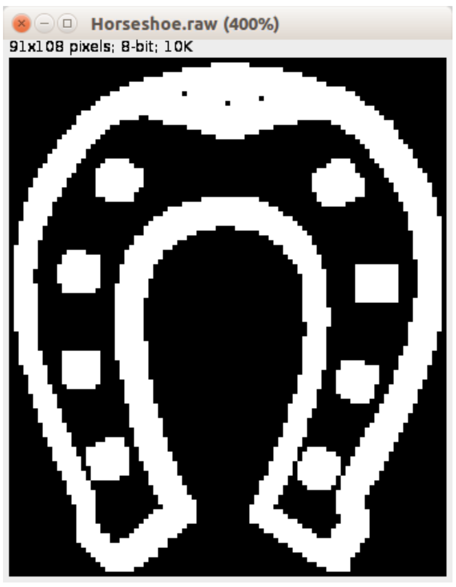
else if M(i,j) = 1, goto step 8;
Step8: If M(i,j) = 1, compare the string pattern with table 14.3.2 in given pattern table. If there is hit then F(i, j) = X(i,j) where X(i,j) is input image else put F(i,j) to 0.
Step9: Repeat step 2-8 until there is no change in image i.e. image converges.

Counting number of holes and nails:
Step1: Scan the output image with a 3*3 window.
Step2: Check if the center pixel of 3*3 window is 1 and all surrounding pixels are 0 then increment the count by 1.
Step3: Output the number of counts.
For counting number of holes invert the image.

Part 2: Implementation to count number of white objects in the image.
Steps of shrinking are same as described above but counting of number of white objects is different. In order to count the number of loops I have implemented the following:
Set any point in the horse shoe boundary of resultant shrinked image as start point and then check its neighbors for the pixel having value 1. Set this new pixel as the next point and mark the current point as 0. Move to this new point and keep repeating this process until all neighbors are zero.
Since nails get shrinked to a single pixel of white they can be counted by the same method as described in part 1.

Experimental Results

Following is the result obtained for shrinking when applied on Horse shoe image



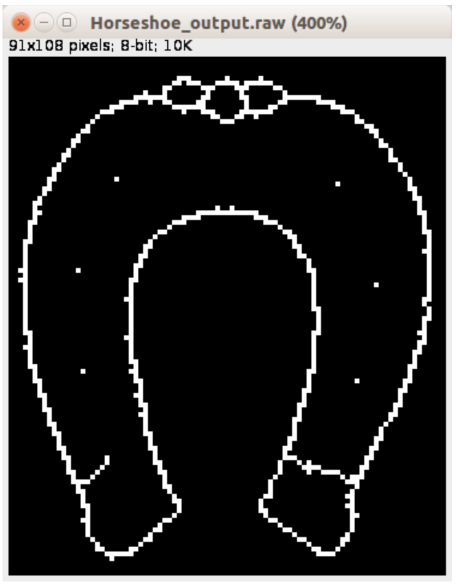


Figure 18: (left)Original Image, (right) Final horseshoe shrinked image

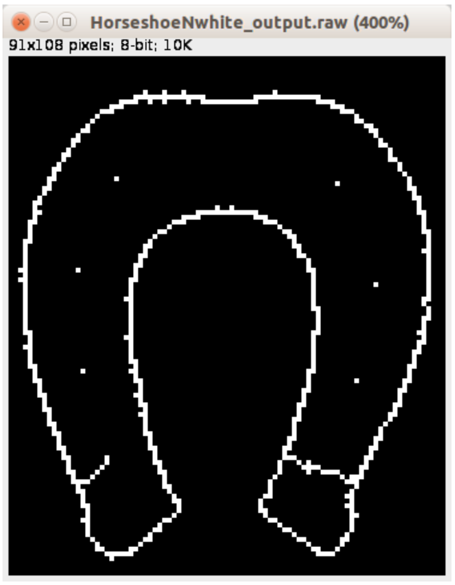
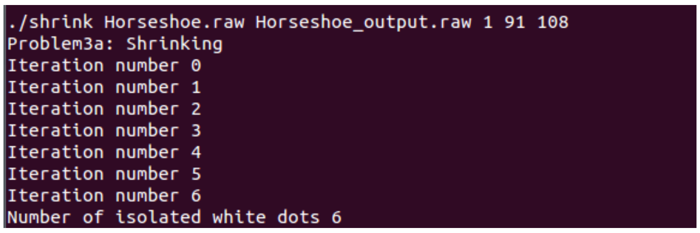
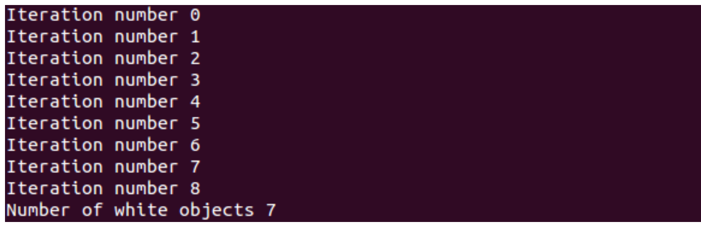


Figure 19: Shrinked horseshoe image after filling out holes

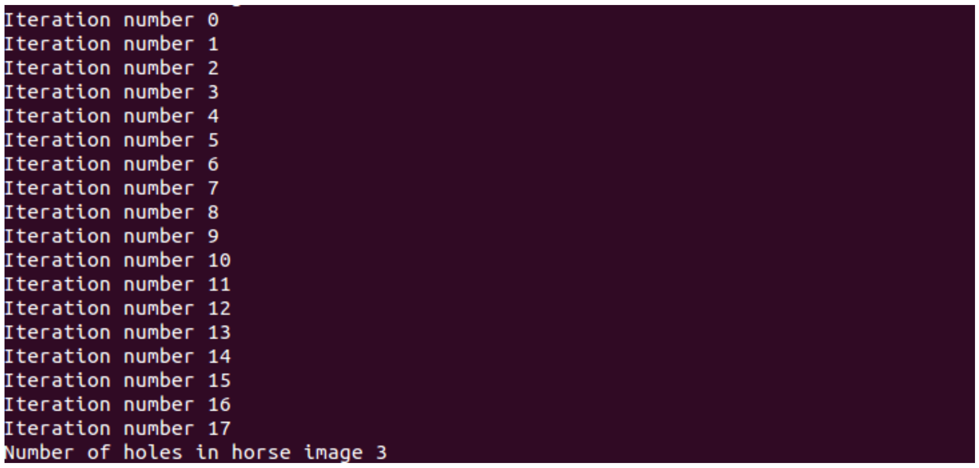


```
./shrink Horseshoe.raw Horseshoe_output.raw 1 91 108
Problem3a: Shrinking
Iteration number 0
Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Number of isolated white dots 6
```



```
Iteration number 0
Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Number of white objects 7
```

Figure 20: (left)Iteration result for counting number of nails, (right) Iteration result for counting number of white objects



```
Iteration number 0
Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Iteration number 9
Iteration number 10
Iteration number 11
Iteration number 12
Iteration number 13
Iteration number 14
Iteration number 15
Iteration number 16
Iteration number 17
Number of holes in horse image 3
```

Figure 21: Iteration result for counting number of holes

Discussion

As results show above original image converged in 6 iterations and it counts the number of nails as 6. If we look at original horseshoe image we see that there are 8 nails but in result we obtain 6 as output. This is because if we zoom in original horseshoe image we see that for last two nails of horse shoe the pixels are connected to boundary. Due to this the lower right horseshoe nail, which is connected from both sides gets shrinked to a line and lower left which is connected from one side gets shrinked to a small boundary. Below figures shows the zoomed in version of horseshoe which shows the behavior I explained above.

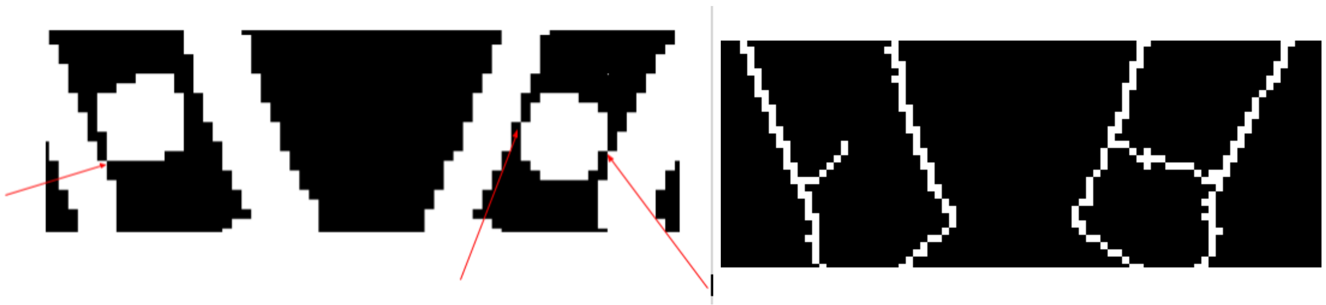


Figure 22: (left)Joint pixel pointed by red arrow, (right) result due to joint pixels

For counting the number of holes in image, image was first inverted and then fed into the shrinking algorithm. As we can see from result above the image converges in 17 iteration with number of holes as 3. This result is as per expectation since there are three holes in original image too.
For counting the number of white objects I have already explained the algorithm in detail in section approach and procedure. For counting number of white objects I have filled the holes by checking if one white pixel is surrounded by all black pixel then mark that pixel as black. results above show both shrinked image and screenshot of the count and iteration. Image converges in 8 iteration with number of objects as 7 (6 nails + 1 horse shoe boundary). Again the explanation is same as explained above for 6 horse shoe nails instead of 8.

Problem 3b : Thinning and Skeletonizing

Abstract and Motivation

As part of this section we were asked to perform thinning and skeletonizing on the horse image provided. As part of exercise I have used hole filling algorithm followed by smoothing filter and then thinning and skeletonizing on image. In next few sections I have discussed the procedure I followed , results obtain and discussion about the results.

Approach and Procedure

As the given image of horse contains lot of holes in the figure and also the horse boundary is not smooth therefore some amount of pre processing is required before the image can be processed for thinning and skeltonizing.

Pre-processing
As part of pre-processing I carried out the hole filling operation first followed by smoothing of the horse boundary.
I tried two approaches as part of pre-processing step:
Approach 1
In first approach I separated the background pixels from the horse boundary using the seed method. Since the first pixel of image is boundary pixel I took that pixel as the starting point and using the 3*3 window I checked all the neighbors of the pixel. if any of the neighbors is same as the center pixel then I mark center pixel with some value (say 2) and move on to the neighbor pixel. I repeat this step for neighboring pixel and this step keeps on going until all black pixels in background have been marked as 2. Once this step is complete all the pixels

which are not marked as 2 will be given value 255 that is white pixel.
Now in order to smooth out the image I used spur filter introduced by professor.Figure 24 shows the result of hole filling using this approach.
Approach 2
In this approach I have used two filters of size 7*7 and 3*3 to find out if black pixels are surrounded by threshold number of white pixels. As a starting step take the original horse Image and check if the pixel value is 0. If pixel value is 0 (black) consider 7*7 filter and check if it has minimum 15 surrounding white pixels then mark that pixel(center) as white. Do this for the whole image. It is seen that after this step most of holes in the horse gets filled up and only few holes are left. Now as next step consider 3*3 size filter and apply the filter to whole of the image. Check if center pixel is 0(black) and it is surrounded by minimum 5 white pixels then mark center pixel as white. Repeat this step recursively for 10 iterations. After these two steps it is seen that all image holes are filled up and also the boundary of image is smoothed out. Figure 25 shows the resultant image with this approach.
The reason I chose to use this approach was because if we look carefully at the holes in the horse body we can see that black holes are surrounded by white pixels. Therefore if we consider 7*7 size filter we can fill bigger holes. Small holes can be filled up with filter of size 3*3. Filter of size 7*7 was applied only once on the image since it was observed that image was being smoothed out more and more if we apply recursively for 2 or 3 iterations thus loosing the original structure of image. Therefore, I decided to use 7*7 only once followed by 3*3 filter. As you can see from Figure 25 result is really good and also the resultant image gets smoothed out in a very efficient manner due to which thinning and skeletonizing outputs are very good with this approach. However, one shortcoming of this approach is that, it cannot be generalized to images, having very big holes.

Processing
Step1: Check if pixel of interest is white pixel. In this case horse is white pixel.
Step2: Create an array M with same dimension as original image.
Step3: Scan the image with 3*3 window.
If X(i,j) = 0, then M(i,j) = 0;
else if X(i,j) = 1, then M(i,j) = 1;

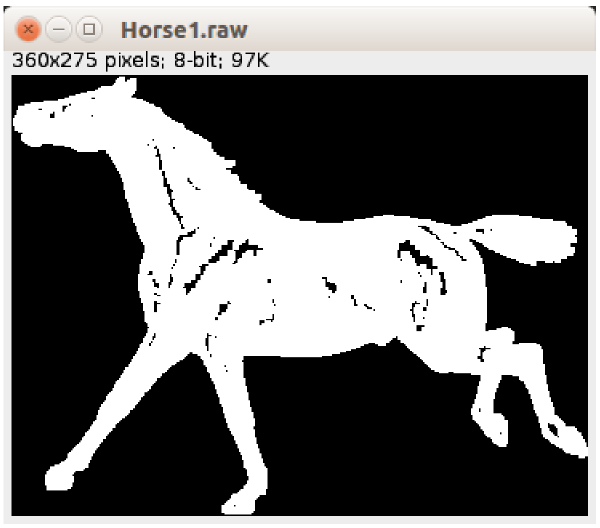
X3	X2	X1
X4	X(i,j)	X0
X5	X6	X7

Figure 23: [Bound = $2 * (X0 + X2 + X4 + X6) + 1 * (X1 + X3 + X5 + X7)$]

Calculate the bound of image.
Step4: If bound is 0 or 12, set M(i,j) as 0.
Step5: If bound lies between 0 and 11, compare binary image string pattern X0-X7 with patterns corresponding to thinning for getting thinning results or skeleton pattern for getting skeletonizing result, given in reference table[4].As part of my implementation I am storing the values in bit format andthen I check if binary string pattern in image is same as pattern in the reference table.
Step6: If there is pattern match set M(i,j) as else set M(i,j) as 0.
Step7: Once M array is complete , scan the whole M array with a 3*3 window.
If M(i,j) = 0, then M(i,j) = F(i,j) where F(i,j) = output Image;
else if M(i,j) = 1, goto step 8;
Step8: If M(i,j) = 1, compare the string pattern with table 14.3.2 for thinning in given pattern table. For skeletonizing compare with pattern table 14.3.3 in reference table. If there is hit then F(i, j) = X(i,j) where X(i,j) is input image else put F(i,j) to 0.
Step9: Repeat step 2-8 until there is no change in image i.e. image converges.

Experimental Results

This section shows result of pre-processing only. Thinning and skeletonizing results are shown and discussed in discussion section.



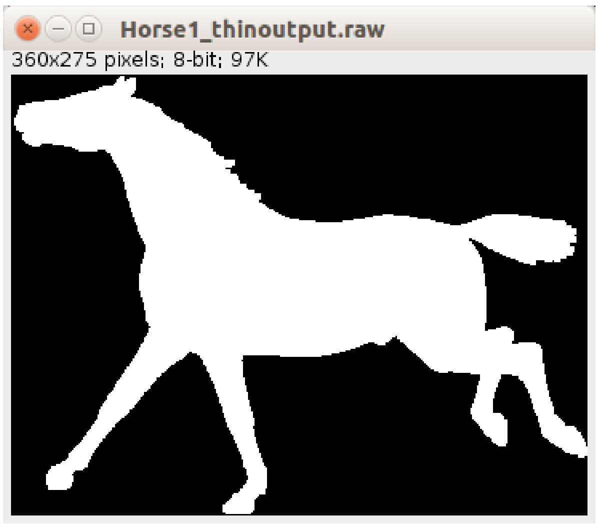


Figure 24: (left) Original Image, (right) Result of pre-processing with Approach 1

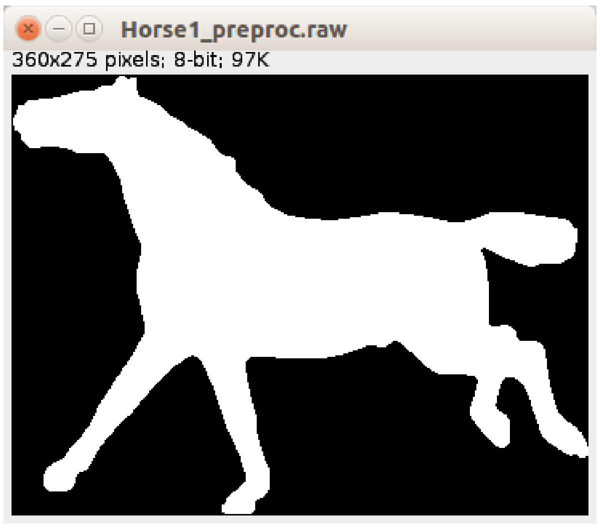


Figure 25: Result of pre-processing with Approach 2

Discussion

Since in this part of problem both thinning and skeletonizing needs to be performed on the preprocessed horse image therefore I will be discussing thinning first followed by skeletonizing results.

Results for Thinning

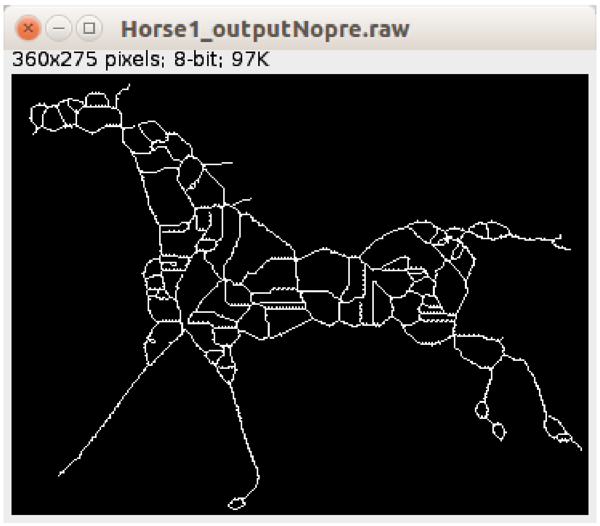


Figure 26: Result of thinning without preprocessing

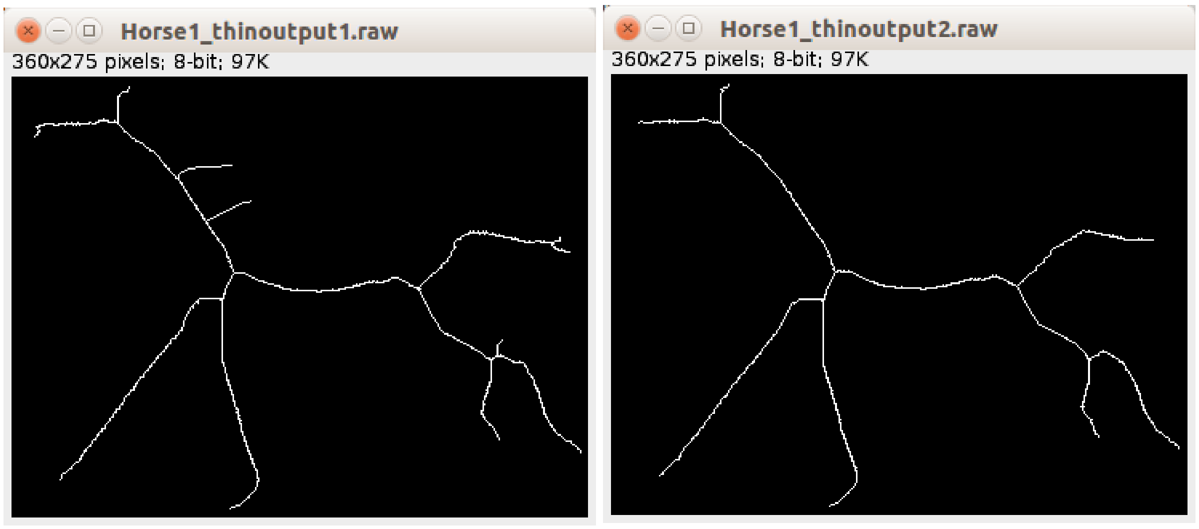


Figure 27: (left) Thinning after preprocessing with approach 1, (right) Thinning after preprocessing with approach 2

Thinning is an erosion based process which removes the selected foreground pixel such that topology of the structure is retained. Figure 26 and Figure 27 shows result of thinning when applied on horse image without preprocessing and with preprocessing step using

both approaches 1 and approach 2. The resultant thinned image was obtained in 58 and 62 iterations with approach 1 and approach 2 respectively.
As we can see from figure 26 the result obtained without preprocessing contain lot of details about the structure. The reason for so many details is due to the fact that original image contains lot of holes and these holes are being treated by thinning algorithm as some sort of structure due to which the topology is retained.
Also if we see from the results above the two thinned images in Figure 27 are slightly different. The reason for the difference in two results is because of the fact that the resultant preprocessed image in approach 2 is smoothed out in a much better way as compared to approach 1. For example, If we look at figure 27 we see two tentacles near horse neck on left figure, these tentacles are due to the horse hair captured in original picture. Please see image below for more clarification, red arrows indicate the pixels due to which those tentacles near neck are seen in thinned image.

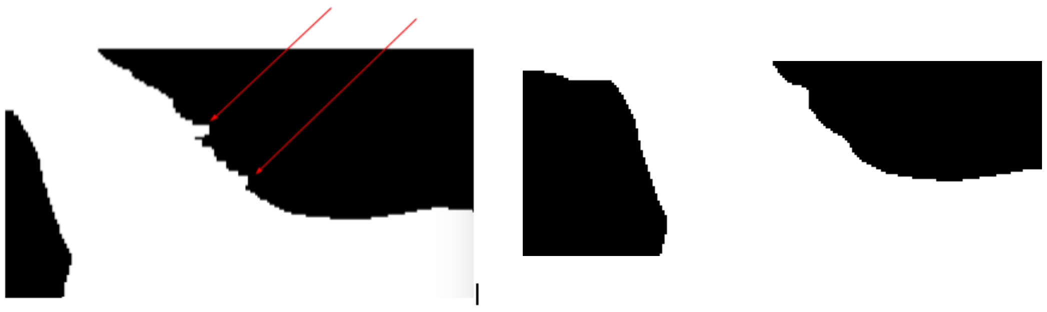


Figure 28: (left) Image with approach 1, (right) Image with approach 2

Results for Skeletonizing

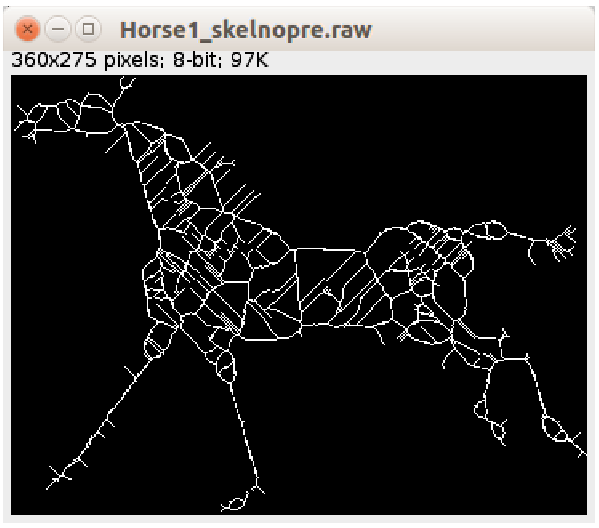


Figure 29: Result of skeletonizing without preprocessing

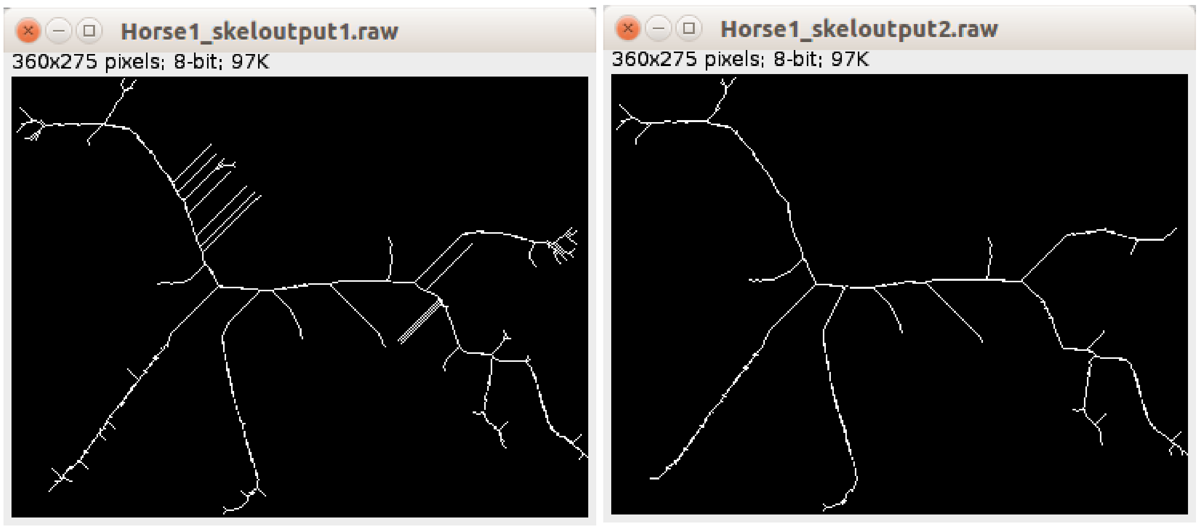


Figure 30: (left) Skeletonizing after preprocessing with approach 1, (right) Skeletonizing after preprocessing with approach 2

"Skeletonization of an image is obtained by a $medialaxistransform$, where the medial axis is the locus of points such that any medial point is equidistant to at least two points on the boundary." Figure 29 and Figure 30 shows result of Skeletonization when applied on horse image without preprocessing and with preprocessing step using both approaches 1 and approach 2. The resultant image was obtained in 45 and 46 iterations with approach 1 and approach 2 respectively.
As we can see from figure 29 the result obtained without preprocessing contains much more

details as compared to figure 26 obtained in thinning. From the skeletonization results it looks like the algorithm is very sensitive to discontinuities or small changes in object. This behavior we also see in two skeletonized images in Figure 30. Due to skeletonization being very sensitive to changes it captures even a single pixel which is not aligned with horse boundary. This results in numerous tentacles on various parts of horse skeleton in figure 30 (left). However in Figure 30 (right) when image is smoothed out these tentacles are greatly reduced.

REFERENCES

[1] B. E. Bayer, An optimum method for two-level rendition of continuous-tone pictures, SPIE MILE-STONE SERIES MS, vol. 154, pp. 139-143, 1999.

[2] D.Shaked, N. Arad, A.Fitzhugh, I. Sobel, Color Diffusion: Error-Diffusion for Color Halftones, HP Labs Technical Report, HPL-96-128R1, 1996.

[3] https://github.com/ralfcheung/Morphological-Processing.git

[4] PatternTables.pdf shared as part of homework 3.

[5] http://utam.gg.utah.edu/tomo03/03_mid/HTML/node109.html