Pooja Voladoddi
6571-9160-65
voladodd@usc.edu

EE569 – Homework #2 Report
October 13, 2013

PROBLEM 1: SPECIAL EFFECT IMAGE FILTERS

1a) Colour to Gray scale conversion

I. Abstract and Motivation

It is a matter of wide recognition that conversion of a colour image to gray scale one is an essential pre-processing step in image processing so that the steps taken further for the actual processing become easier. Gray scale images are easy to work with for analysis in areas such as edge detection that helps in image segmentation based on edges/boundaries. Apart from this, it can also help detect anomalistic patterns easily, like in images of astronomical interest or medical imaging (brain scans etc). In such cases, conversion to gray scale is essential as it makes the objects of interest more prominent to observe and analyse, not to forget their importance to produce binary images for morphological processing and such.

II. Approach

Converting a colour image to gray scale varies according the source colour space. At the heart of this procedure though, lies the process of converting the 'luminance' component of the colour image to its corresponding gray scale luminance. And one of the colour spaces – the YUV colour space - takes human perception into account for this conversion. Human vision system is more perceptive to colour Green and least perceptive to Blue [1], and thus we have the equation to convert such an image to gray scale according to the equation provided in the homework.

$$Y = 0{:}299R + 0{:}587G + 0{:}114B$$

Where Y, R, G, B are Gray, Red, Green and Blue channels respectively.

In the homework, the colour image is read channel-by-channel, individual channels are extracted and are multiplied with the respective coefficients (as given above) to obtain the resultant grayscale image.

III. Results

The original 24-bit RGB images and the obtained gray scale images are as shown below –

Fig 1. Original v/s Grayscale harrypotter.raw

Fig.2. Original v/s Grayscale lena_noisy.raw

Fig.3 Original v/s grayscale naruto.raw

IV. Discussion

As far as the results show, the original images seem to have successfully been converted to gray scale images using the given formula and the image details such as contrast seem to be preserved.

1b.) Pencil Sketch using Edge Detectors.

I. Abstract and Motivation

Edge Detection is one amongst many morphological processing procedures performed on an image. Edges, as explained in class, can be defined as isolated, local points of sharp intensity changes in an image.

Closed or linked edges are called boundaries and edge detection further helps in boundary detection. Even on its own, Edge Detection is important to determine image features for scene interpretation, the application of which lies in many areas such as video surveillance, material sciences in detecting faults etc.

The goal of this problem though is the application of edge detection algorithms to obtain an image that looks like pencil sketched image. This could be used in post-processing of images captured by any camera for, for instance, artistic purposes.

II. Approach

There are multiple approaches to Edge Detection – such as First Order Derivative edge detection, Second order derivative edge detection, Optimal edge detection (using Canny edge detection).

However, in this problem the 1st order derivative and the 2nd order derivative approaches have been implemented.

1st order derivative edge detection [2]-

The simple equation is this

$$\partial f/\partial x = f(x+1) - f(x)$$

and this signifies differentiation across the x-axis, between the x+1th and xth pixel. This is known as the forward difference. To obtain the pencil sketch image, we obtain the edge magnitude map/ the gradient magnitude map in both the x- and y-directions [2].

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

which is basically for detecting the presence of sharp edges and displaying them (local maxima points are displayed).

The mask used is Sobel mask – one for horizontal direction, and one for the vertical – that is the row gradient map and the column gradient map in order to obtain the final gradient magnitude map according to the equation above.

The pencil sketch is gotten by inverting the resultant magnitude map.

2nd order derivative edge detection [2]-

The basis of this method is finding the points of discontinuities by finding the zero-crossings in the second derivative of the grayscale image.

This method is more robust in the sense that even the feeble edges are discovered but this also leaves behind the disadvantage that the mask – Laplacian mask – used for obtaining the Zero-crossing map is highly susceptible to noise.

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Laplacian masks –

0	1	0	1	1	1
1	−4	1	1	−8	1
0	1	0	1	1	1
0	−1	0	−1	−1	−1
−1	4	−1	−1	8	−1
0	−1	0	−1	−1	−1

The method is as follows –

Step 1: Apply **Gaussian blur** (in order to make the method less susceptible to noise) to the grayscale image.

Step 2: Apply **Laplacian mask** to the Gaussian blurred image (the steps above, when combined give a LoG filter- Laplacian of Gaussian).

Step 3: Thresholding on the resultant matrix to get **Zero-Crossing Image**

Step 4: Display detected edges.

III. Results

Fig 3. Pencil sketch images obtained of [L-R] – 1) harrypotter.raw 2)lena_noisy.raw 3) naruto.raw after 1st order derivative edge detection

Fig 4. Pencil sketch images obtained of [L-R] – 1) harrypotter.raw 2)lena_noisy.raw 3) naruto.raw after 2nd order derivative edge detection

IV. Discussion and conclusion

The main challenge of obtaining the pencil sketch lies in the thresholding step of 2nd order edge detection. As can be observed, Fig 3 shows that 1st order edge derivative method gives a pretty clean pencil sketch effect on all of the images.

Comment – The problem asks to perform, if needed, post processing on images. This seems to be unnecessary for the results obtained in Fig.3 . However, it is to be noted that pre-processing is absolutely crucial for one of the input Lena images as the image is riddled with salt & pepper noise. This is achieved using a 5x5 median filter before performing both 1st order and 2nd order edge detection methods. The compromise is on the accuracy of edge detection as the median filter blurs the image considerably along with the noise removal.

Thresholding in both 1st order and 2nd order methods have been done using the mean value obtained in the edge gradient magnitude map (1st order) and the Laplacian image (2nd order). In order to get a balance between unnecessary edges whilst maintaining the weaker edges

representing the more finer details of the image, this has been done. Essentially, if the value in the final edge matrix is above the mean value, it is considered to be an edge, else it is considered to be a part of the background.

Without Gaussian blur and without thresholding, the resultant edge map of 2nd order derivative looks like below –

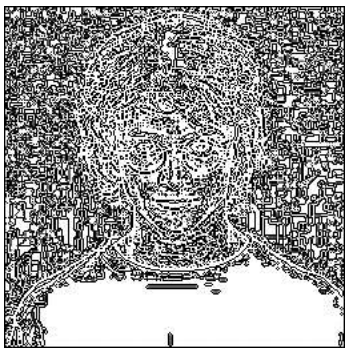


Fig.5 – pencil sketch using 2nd order derivative without Gaussian blur and thresholding for 'harrypotter.raw'

As shown in Fig.4, the result is much better when thresholding is done according to the mean value and its multiples (1.5*mean value gives results observed above) (note that the absolute of the intensity value is compared to the absolute of the mean so that the pixels with negative values are considered and not neglected).

Challenges in getting a good pencil sketch effect- As observed from Fig3 and Fig 4, the 1st order gives a pretty good effect. However, the 2nd order derivative, between being too susceptible to noise and the interest in obtaining only specific edges, gives us a very hard trade off. If the threshold is set to a multiple (either integral or real) of mean, it is a trade off between detail preservation and covering all the edges that matter. If post-processing such as denoising is performed, the minor edges will be blurred due to the averaging effect and some detail loss will occur.

For the noisy Lena image, 'lena_noisy.raw', median filter of size 3x3 would definitely give a better edge preservation but some amount of noise would still be there. Therefore, a 5x5 median filter has been used to balance the salt and pepper noise removal as well as preserve most, if not all, edges that matter.

Overall, for this problem 1st order edge derivative is the method of choice, i.e to get the pencil sketch image only. Second order edge detection in real life applications is better than First Order Edge detection. However, noise needs to be tackled in the second order edge method.

1c.) Background Special Effect

I. Abstract

The goal of this problem is to apply the pencil sketch image to both the images involved and combine them to make the sketch look more appealing or visually pleasing like in a realistic setting.

II. Approach and Procedure

The approach and procedure to obtain the asked result is fairly simple. The Homework sheet already discusses the formula

$$I'(i,j,k) = I(i,j,k) + a * G(i,j,k) + b$$

Where a = (-0.75,-0.95) are the suggested values and b = (10,30) are the suggested values.

So according to the formula, the pencil sketch image (I) and the grainy background(G) are varied with respect to alpha and beta values that results in different looking pencil sketches .

III. Results

Fig 6. Background special effect for 'harrypotter.raw' with [L-R] **1**) a=-0.8, b=20 **2**) a=-2, b= 30 **3**) a=-4, b=10

Fig 7. Background special effect for 'lena_noisy.raw' with [L-R] **1**) a=-0.8, b=20 **2**) a=-2, b= 30 **3**) a=-4, b=10

Fig 8. Background special effect for 'naruto.raw' with [L-R] **1**) a=-0.8, b=20 **2**) a=-2, b= 30 **3**) a=-4, b=10

IV. Discussion and Conclusion

Since first order derivative pencil sketch images are better than their second order ones, those images are used to get the background special effect problem.

The results using different alpha and beta values are as shown in Fig.6, Fig.7 and Fig.8 It is noted that as alpha value increases, the background's presence in the image is increased. The edges of the grainy background are darker on increasing the alpha value and hence they seem to give a better background that makes the pencil sketch look realistic.

Beta, as it is increased, improves the contrast of the whole image as it shifts the pixel value to a larger one than what existed before. This can be noted especially between the left-most (higher beta value) and the right-most images (lower beta value) of Fig 6, 7 and 8.

Observing the results, it can be concluded that both alpha and beta values to be moderate. Alpha more than 5 gives a very sharp background and Beta more than 60 gives a very bright image.
It is suggested to keep the values of alpha between 0.75 to 4 and Beta between 10-20.

1d) Transition Special Effect

I. Abstract and Motivation

As the title of the problem makes it clear, the topic here is transition about frame manipulation so that a collection of manipulated frame images when played as a video form a transition special effect.

This is useful for many practical scenarios that require production of videos for many purposes. Some examples are the film industry, animation industry to create movies, medical scans, architectural blueprints and 3D printers that can require 3D analysis. In that situation this problem will have to be blended with geometrical manipulation of the image frames to render a 3D effect in a video.

II. Approach and Procedure.

There are two types of transitional special effects created for this problem.

- Sliding transition

  This has been incorporated using replacement of pencil sketch image by original colour image in continuous frames. At a good frame rate specified during video creation, the sliding effect is perceived to be continuous and hence an animation/special effect video is created.

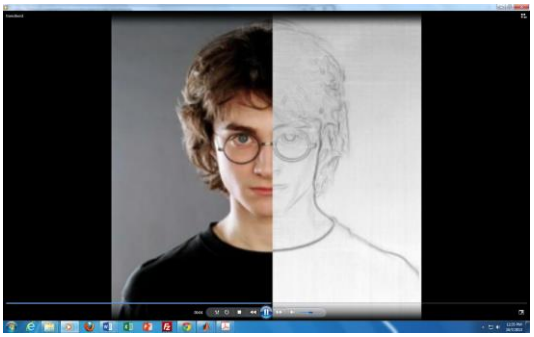


Fig.10 Sliding from left transition on 'harrypotter' images at 50%mark

  The figure above shows the screenshot of the movie at half-length of duration.

- Fade-In

  The formula for generating this effect is for creation of an additive combination of the original image and its pencil sketch image.

  $$I'(i,j,k) = (0.5+a)*I(i,j,k) + (0.5-a)*G(i,j,k)$$

Where I' is the resultant image;
'a' is the monotonically increasing parameter (-0.5,0.5) to generate the combination of the colour image and the gray sketch image.

G is the pencil sketch image and I is the original 24-bit image.

Fig.11 screenshot of Fade-in transition on 'naruto' images at 30%mark

III. Results

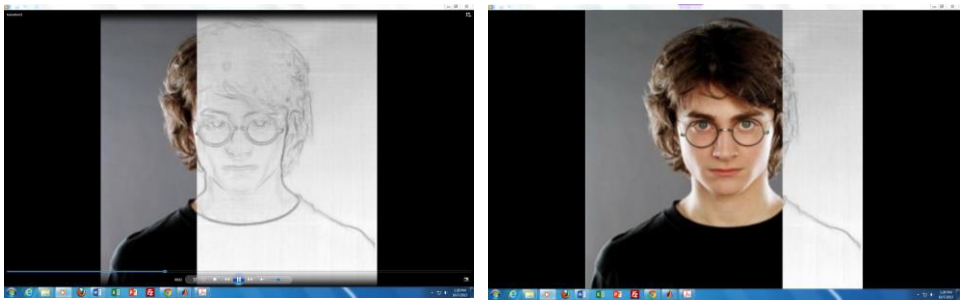


Fig 12. Sliding from left transition on Harry Potter images.

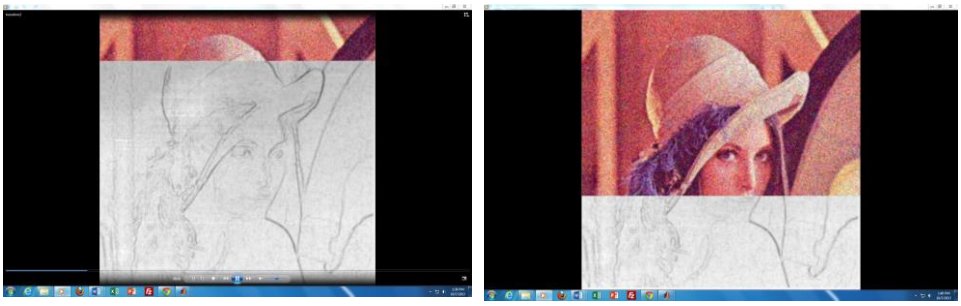


Fig 13. Sliding from top transition on Lena images.

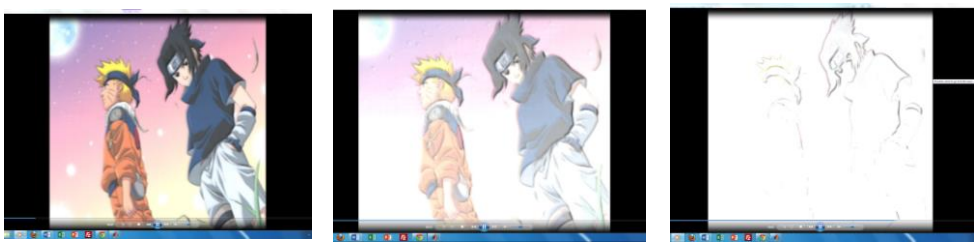


Fig 14. Screenshots of fade-in transition on Naruto images.

IV. Discussion and Conclusion

MATLAB allows video creation using the functions writeVideo( ) and getframe( ).

The output images or frames generated using the above functions are stored in a data structure of type 'struct' (which is a structure and not an array) and all the frames are written or dumped into the video object created using 'writeVideo' function.

The videoWriter( ) function allows to create a video file for display in different formats. In this problem, the videos created are stored as .AVI format videos – 'transition1.avi' , 'transition2.avi', and 'transition3.avi'for slide from left, slide from right and fade-in transitions respectively.

The challenge in this problem is mainly to figure out how to implement the logic in the context of video creation. Familiarity with relevant Matlab functions is required. The problem itself is relatively simple.

In sliding effect, the color image sweeps through row-by-row or column-by-column of the already swept pencil sketch image.

In fade-in effect, both the color image and pencil sketch are combined and the parameter 'a' decided how much weightage is given to display the color image and to display the pencil sketch in a single frame. When monotonically increased and in several frames, the fade-in effect is rendered.

1e) Colour pencil sketch

I. Abstract and motivation

The idea of this problem is to extend the pencil sketch algorithm to a 24-bit colour image from a 8-bit gray pencil sketch image.

This is particularly helpful for graphics and rendering purposes as well as digital art purposes.

II. Approach and Procedure.

The most simple approach would be to apply the edge detection algorithm to get colour sketch pencil to each of the three R,G,B channels separately and them.

There were two similar and slightly different approaches taken to do this.

The **first approach** involved using the equation provided in 1**c.)** to generate the colour pencil sketch where the grainy background is also added.

The **second** one involved applying the pencil sketch algorithm to just the original image's pencil sketch without adding the grainy background.

The comparison of results is provided as below.

III. Results

Fig 15. [L-R] Approach 1 and Approach 2 for colour pencil sketch on 'harrypotter.raw'

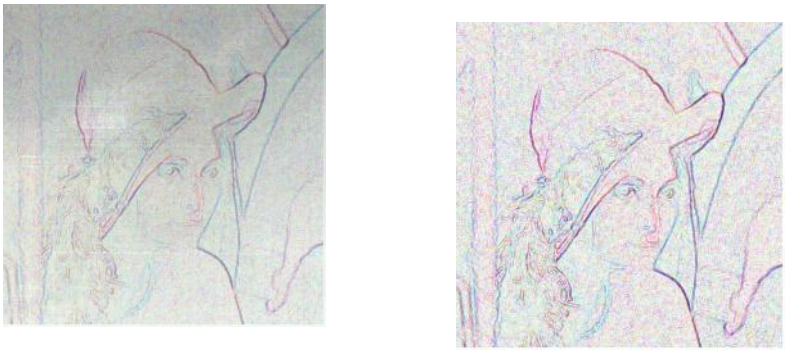


Fig 16. [L-R] Approach 1 and Approach 2 for colour pencil sketch on 'lena_noisy.raw'

Fig 17. [L-R] Approach 1 and Approach 2 for colour pencil sketch on 'harrypotter.raw'

IV. Discussion and Conclusion

Approach 1's flaw lies in the fact that grainy background is added. The colour pencil sketch obtained with the image obtained as a result of combining grainy pencil sketch and the original image's pencil sketch has lessened number of sharp edges. As a result, a few of the colour edges are lost.

And hence, the contrast and the aesthetic quality of the sketch is lower.

Approach 2 conveniently ignores the grainy background and in doing so, we can observe more details and fill in the color sketches. For example, the color in Harry Potter's cheeks; the overall color tone of Lena; and the details of the sky, moon and clothes in Naruto are clearly visible.

However, both these approaches suffer from the fact that not all the images have the same amount of edges. This is fairly obvious to observe that Naruto has the most clean, sharp number of edges so it's deducible that the edges are reasonably divided among all the channels. However, the cases where the edges are more in a particular channel(s), like in Lena and Harry Potter, the color gotten is from those particular channel (s) and hence it looks to be unpleasant to the eyes.

----------END OF PROBLEM 1----------

PROBLEM 3: DIGITAL HALF-TONING

3a) Dithering

I. Abstract and Motivation

Half Toning is representation of an image using two tones that renders a continuous image to the eyes, using a reprographic (reproduction of graphics using mechanical or electrical) technique through the use of dots belonging to those two tones, thus giving an illusion of depth to a 1-bit image like that of an n-bit image (n=8 in this problem).

Half-toning is especially important in printing, desktop publishing and graphics related areas which require photography printing. This is important because it makes the printing much easier and efficient using deposition of ink in the form of discreet dots, rendering a continuous tone to the image. Thus, cartridge ink requirements are reduced for printing jobs. With the advent of digital half toning, the same technique has been used to analyse and simulate the half-tone images on computers through dithering techniques which have improved the sophistication and aesthetic quality of half-toned images.

One of them is dithering, which is essentially a sort of random noise applied to the dots or the image created by dots, so that quantization error is reduced and hence, a more continuous image is obtained even though the image is technically half-toned, and an illusion of color depth is rendered in case of images.

II. Approach and Procedure

There are, among several other ways, three methods to perform dithering – Fixed Threshold Dither; Random Threshold Dither (Random Dither), Dithering Matrix (or Ordered Dither).

- **Fixed Thresholding**
  In this method, the ultimate image that is produced is a binary image. A manual threshold is set and each pixel F (i, j) of image J is set to 0 or 255 (for an 8-bit image) depending on the threshold. To represent this in a mathematical form,

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

- **Random Thresholding**
  - For each pixel, generate a random number in the range 0-255 using rand function in Matlab. This gives a Uniform Distributed random variable during each iteration, for each pixel. The generated number is also multiplied with a Gaussian distributed Random Variable with zero mean and standard deviation = 0.5 to improve the image, which will be discussed in section IV.

  - Compare the pixel value with that random number (T), and map it to 255 if greater than T, else map it to 0.

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

- **Dithering**

This approach is to break the monotone effect rendered on the image by the fixed thresholding, and give a less noisy appearance as rendered by random thresholding.

This is done by using an index matrix, which is later used to generate a threshold matrix to do dithering periodically over the entire image.

Step 1: Create an index matrix according to the equation

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

where 'n' is the order of the matrix window across the image and

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

The values inside the matrix are Bayer parameters that indicate the likelihood of that particular dot/pixel to be turned on or off.

Step 2: Create a threshold matrix

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

Step 3: apply it to the whole image periodically to produce a normalized output image (which can then be scaled)

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

III. Results

NOTE: It is advised to view the results after running the code as rescaling the saved images changes the resolution and image formats (PNG and JPEG) both seem to have affected the results

Fig. 18 Original [left] and Fixed Threshold (127) halftoned image of 'man.raw'[Right]

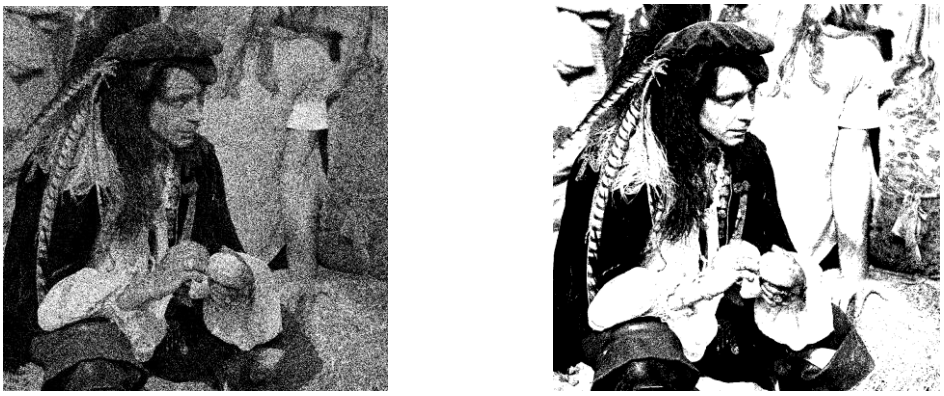


Fig19. Uniform random thresholded [Left] and Normal Random thresholded [right] halftoned images of 'man.raw'

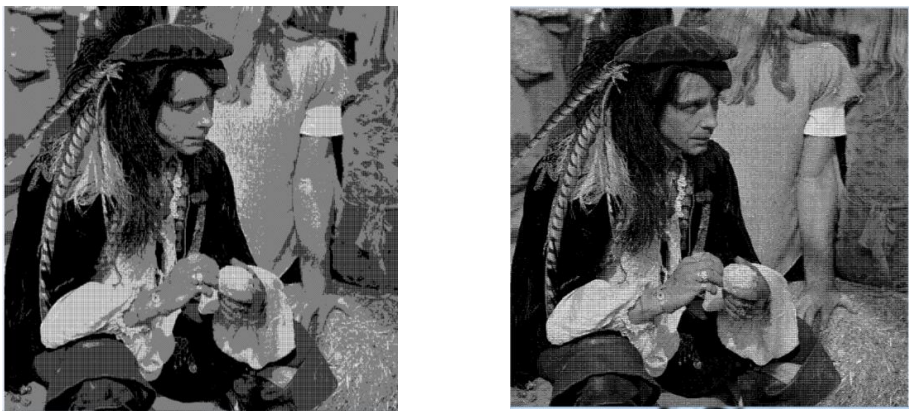


Fig20. 2x2 Dithered Image [Left] and 4x4 Dithered [right] half toned images of 'man .raw'

IV. Discussion and Conclusion

Fixed Threshold as observed from Fig18, clearly does not give a very continuous image, there is contouring and the image contrast is very bad. The half-tone rendered is a very sharp image and is not visually pleasing since quantization error is very much. The threshold chosen is 127 because it seems to be a more intuitive option to keep the threshold at half of the total number of intensity values (256 levels for an 8-bit image).

On changing the threshold to T=<127(40 and below), it is noted that the image is very bright, which is quite obvious because most of the intensity values (all the intensity values above 40) are assigned to 255. It can be concluded that as and when the fixed threshold is reduced to a value below 127, the contrast is very high and the picture appears white for the most part. The bright regions appear very bright and the slightly darker pixels from the original gray scale now occupy higher intensity values.

Fig.21-Fixed thresholding halftone image (T=40)

On changing the threshold to T=>125 (say 180), it is noted that the image is very dark, which is quite obvious because most of the intensity values (all the intensity values below 180) are assigned to 0. It can be concluded that as and when the fixed threshold is increased above 127, the contrast is very high but the picture appears black for the most part. The darker regions appear the darkest and the slightly brighter parts of the grayscale image are now black.

Fig.22 Fixed Thresholding halftone image (T=180)

For **random thresholding**, as described in the procedure, the random number **rand (i, j)** generated during each iteration is fit to
→ Uniform random distribution
→ Normal distribution

It can be seen from figure 19[right] that although the image rendered in Uniform random distribution is continuous and preserves colours to a certain extent, it is a very noisy image and does not have a pleasant impact on human eyes.

The Normal distributed image on the other hand has very less "patterning" visible, mostly notable on the lighter regions of the image, and the noise is very less compared to the Uniform distributed image. Standard Deviation is taken as 0.5 as it is observed that SD<0.5 and SD>0.5 tend to give a sharp darker image and a sharp brighter image respectively.

Overall, to break the monotone effect of the Fixed Threshold, Random Thresholding gives a very good result when compared to the former.

Dithering matrices (or Ordered Dithering) – when compared to other methods, Dithering (**Fig 20)** gives the best results as there as negligent amount of noise (when compared to random thresholding) and better contrast preservation (when compared to Fixed Thresholding) – there is a fine balance between the two. However, the gray level details in 2x2 are not preserved well compared to Normal Random Thresholded image.

This seems to be corrected in 4x4 dithered halftone image **Fig20** (**right**) where the color details are preserved (observe man's face , the hair of the girl in the background).

In conclusion, random thresholding gives a very good colour preservation but has a lot of noise , where as dithering takes care of both noise handling and color preservation (the latter in the case of larger dithering matrix orders like 4x4).
Therefore, for this image, Dithering by 4x4 Bayer matrix seems to have the best result and is the closest to the original image.

3b) Error Diffusion

I. Abstract and Motivation

Error diffusion is another technique to render a half-toned image. The difference between this and the Dithering matrix is that the quantization error produced by the pixels processed is distributed according to certain weights to the unprocessed pixels.

Therefore, unlike the other point-operation based dithering techniques (where processing on one pixel doesn't affect neighbouring pixels) , error diffusion is an **area-based** operation because the processed pixels affect the processing of the other, untouched pixels.

The advantage of this method is that the edges are preserved and hence the details are preserved to a much better extent than other half-toning techniques, as we will see further.

II. Approach and procedure

As discussed in class and in Homework paper, there are three techniques or methods among many other for performing error diffusion on images.

Three of them performed here, are

- Floyd-Steinberg Error Diffusion (FS)
- Jarvis, Judice and Ninke Error Diffusion (JJN)
- Stucki error diffusion (SE)

For all three techniques,
the steps involved are the same –

1) Set a threshold T for each pixel scanned.
2) If pixel intensity is greater than T, set it to 255 else 0. This results in quantization error E.
3) E is spread to pixels lying further (that are as of now, unprocessed) using the corresponding error diffusion matrix.
4) Continue until all the pixels are scanned and error is diffused to further pixels.

Floyd-Steinberg Error Diffusion:

For this method, both straight scanning and serpentine scanning are performed to see and observe which one gives better results. In straight scanning, the pixels are scanned from left to right.

And the error matrix is given by,

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

Where the … represents already processed pixels, and * indicates the pixel currently being processed.

Serpentine scanning is also tried where the scanning is from **right to left** for **even rows only. [4]**

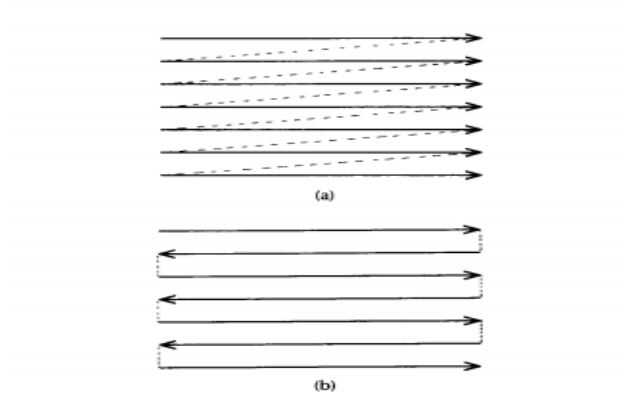


Fig. 22. Two processing path options. (a) Normal raster. (b) Serpentine raster.

Jarvis, Judice and Ninke method:

The error matrix for this is as given below.

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

It can be noted that the error diffusion is much slower here. The method involves spreading or distribution (diffusion of error) of quantization error among the next 12 pixels as opposed to only 4 pixels in FS method.

Stucki Error Diffusion :

Stucki error diffusion is quite similar to JJN in operation but the weights of diffusion employed are as below. The difference in error/ the output will be discussed in section IV.

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

III. Results

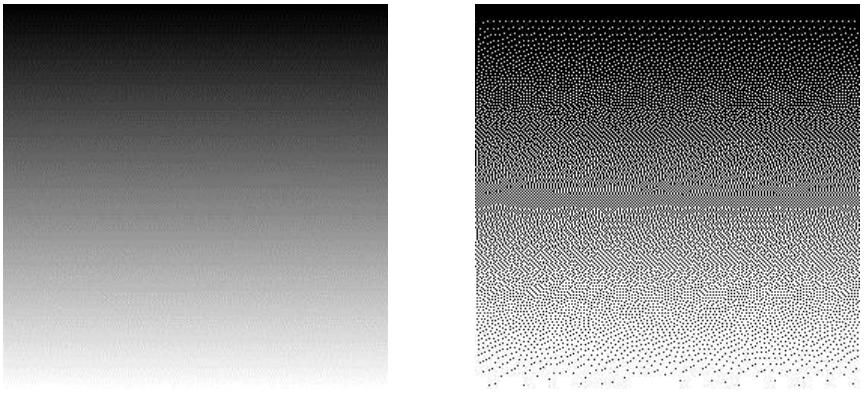


Fig.23 [L-R] Original Image ‘gradient.raw’ and ‘error diffusion using FS (straight scanning)

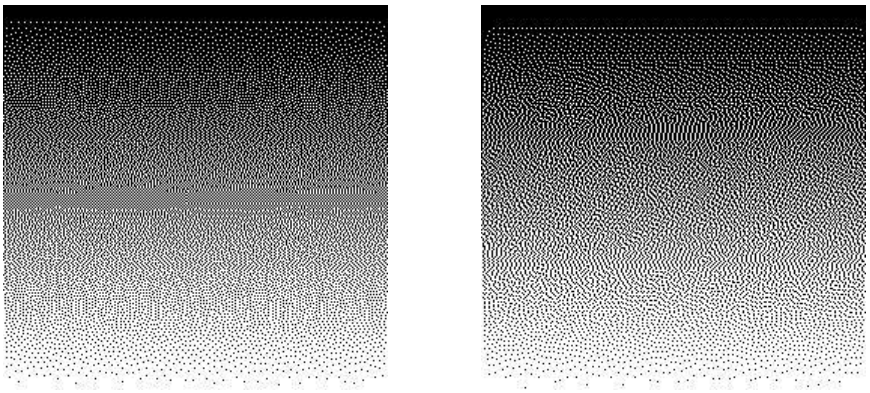


Fig.24[L-R] FS error diffusion (serpentine scanning) and JJN error diffusion (right)

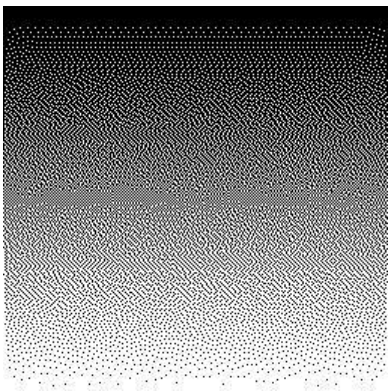


Fig.25 Stucki error diffusion

IV. Discussion and conclusion

On carefully observing Fig 23 [R] , Fig 24[L] we can observe there is a fine difference between normal raster scan and serpentine scanning results of FS (Floyd Steinberg method). Notice how the colour and transition from black to white levels are much better preserved in serpentine scanning order. In addition, there is also a smoother diffusion in serpentine scanning when compared to normal scanning. There are less "bandings" visible and as a result, the transition is much smoother. This can be attributed to the fact that the error diffusion is evenly spread out rather than accumulating in certain pixels during serpentine scanning [5].

We will shift our attention to the pictures in Fig 23,24 and 25 on the right hand side now. If we see the outputs for JJN and SE, we will notice that the pixels seem to be spread out more than in FS. This is an obvious output when one observes the error diffusion matrix – the quantization error is spread out to 12 pixels than only 4 pixels as in FS. This is also the reason why the overall tone of FS is much smoother whereas the tone of JJN and SE outputs are coarser (JJN even more so).

Banding is very less in JJN whereas the degree of smoothness is much higher in FS.

Comparing only JJN and SE (fig 24[R] and 25) we can conclude that, while the image will be rendered with a coarse halftone in both cases, the SE output is sharp and finer than its JJN counterpart. However, the continuity provided in JJN seems to be the best for the given problem.

Improvement: Blue noise is a noise which has its spectral density proportional to its frequency [6]. What this essentially means is that this produces very smooth error diffusion if applied owing to the above property, as distortions ratio is lessened. Correlation between blue noise coefficients and image frequencies will tend to be higher and hence importance of errors is reduced. Thus, blue noise could be applied to the image before error diffusion.

This is further corroborated in the paper provided in class for reference, [7] which shows that an image corrupted with blue noise renders a much better halftoned image and thus gives a better inverse halftoned picture as well.


[Left] – Image after JJN, [Right] – Image after SE error diffusion


[Left] – Image after 2x2 dithering, [Right] – Image after 4x4 dithering

Looking at the above pictures, 4x4 dithering there is a bit of patterning is left, it does not appear noisy compared to 2x2 dithering result (loss of colour) or its JJN counterpart. However, Stucki Error Diffusion has a nice balanced output while preserving the color, contrast and all the details, as well as minimalistic presence of noise. Therefore, Error Diffusion seems to give a better result compared to Dithering outputs.

3c. Inverse Halftoning

I. Abstract and Motivation

Digital Inverse Halftoning is the reconstruction of a grayscale image from its halftone rendering.

This process is equally important as the Halftoning process. Reconstructing a halftone rendering is of immense importance in digital printing and desktop publishing industry, as well as digital scanning industry – for which inverse Halftoning will provide ease of switching between, or manipulating the halftone renderings to get the required output results in an efficient and cost-effective way.

The method of implementation here is as given in the paper 'Inverse Halftoning' by Christopher Miceli and Kevin Parker of University of Rochester, New York, USA in 1992.

The approach and procedure as well as the algorithm are discussed in section II, results are observed in section III, and discussions will be made in section IV.

II. Approach and procedure

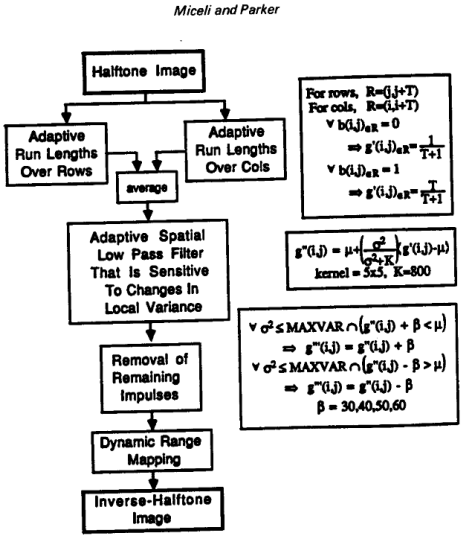


Fig. 6 Block diagram of cascaded inverse-halftone algorithm.

Fig 26. Miceli Parker Three-stage Cascaded Inverse Halftone Algorithm

The procedure in terse reproduced above in Fig 26 from [7].

Step 1: Run Adaptive Run Length Algorithm over rows of halftone image, and over columns of halftone image. Where thresholding condition is given as

$$\text{if } b(i,j)=b(i,j+1)=\ldots=b(i,j+J)=0 \ ,$$

$$\text{then } g'(i,j)=g'(i,j+1)=\ldots=g'(i,j+J)=\frac{1}{J+1} \ .$$

'b' representing the binary halftone image and 'g' representing normalised gray level approximation.

Step 2: Take the average of both row and column run length implemented images

Step3: Perform filtering using variance sensititivity, using Lee's additive filter using parameter K=800 . MAXVAR is the local variance.

Step4: Remove spurious impulses, and perform dynamic range mapping to provide utilisation of 0-255 gray levels (in this case). Note that assumptions are discussed in section IV.

III. Results

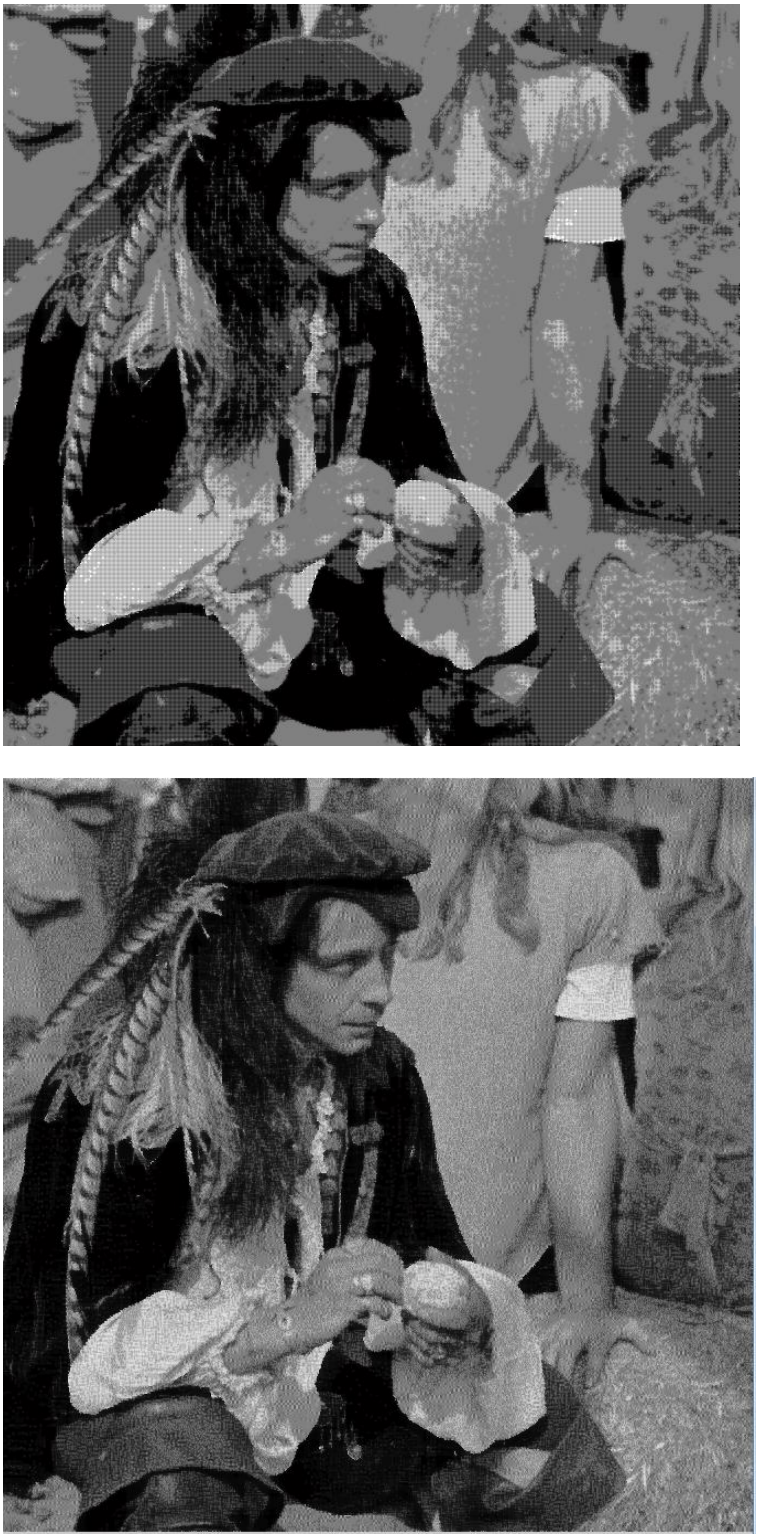


Fig 27 – Inverse Halftoning on 4x4 Dithered Image [top]
And on Floyd Steinberg Error Diffused Algorithm [bottom]

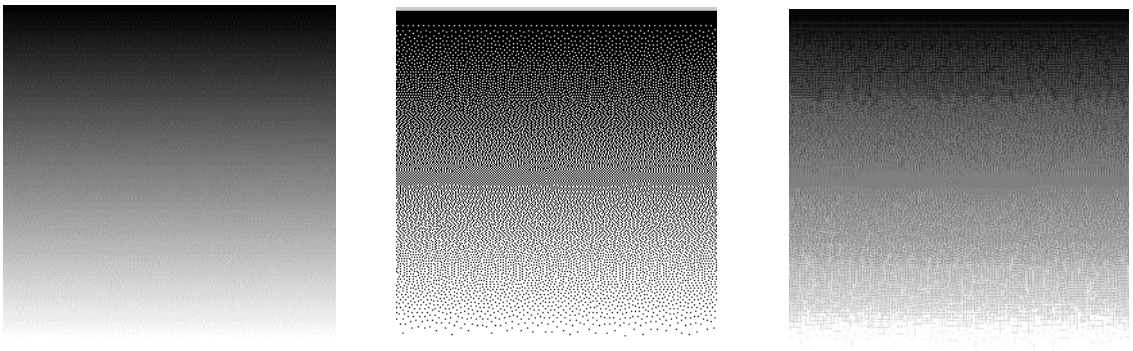


Fig 28– Original ‘gradient.raw’ image [Left] ; FS halftoned image[Center] ; Inverse FS halftoned image

IV. Discussion

The assumption made in this problem in step 4 is that there is a 40 black-pixel run length and 10-pixel white pixel run length (for dynamic range mapping).

MAXVAR is the local variance chosen as and when the neighbourhood moves according to the function created ‘variance.m’ thereby removing the need for selecting MAXVAR each time.

From fig 28, we can see that Cascaded Inverse Halftone algorithm gives a pretty close approximation of the grayscale of Floyd Steinberg halftoned rendering of ‘gradient.raw’. However, the smoothing effect produced is such that the darker regions are not preserved in colour and sharpness.

From Fig.27, it can be observed that the Cascaded Inverse Halftone Algorithm does not work that well with a Dithered image when compared to its results on the Error Diffusion halftoned image.

Notice how there is contouring and artefacts present in the [top] image of Fig 27 whereas the [bottom] image of Fig 27 comes very close to the original ‘man.raw’ image. Details such as the girl’s hair (background) and clothes contrast is preserved excellently in the latter case.

It is noted in [7] by the authors as well, that this algorithm works particularly well with blue noise corrupted halftone or an error diffusion halftone rendering, further cementing and verifying the results and observations of the above statements made in the report.

In conclusion, Cascaded Inverse Halftone algorithm works well for Error Diffusion Technique with some loss in edge details and in sharpness, and gives not so satisfactory results on Dithered halftone renderings.

----------END OF PROBLEM 3----------

PROBLEM 2: Morphological Processing

2a.) Shrinking

I. Abstract and Motivation

Morphological Processing is the manipulation of shapes of image to get desired features from it. The operations, i.e. morphological processes or operations performed on images are non-linear and depend on the relative ordering of the pixel values, not on the intensity values of the individual pixels, and are usually performed on binary images.

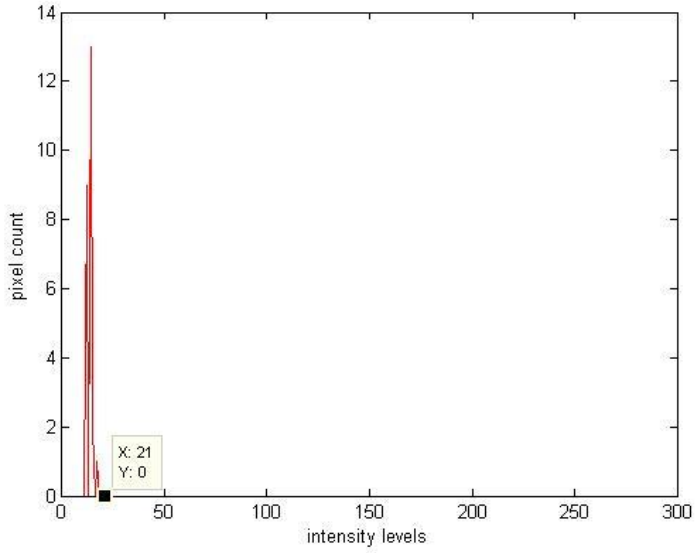
This is noticeable in some of the 'Hit-or-Miss Transformations' that have been performed throughout this problem. Shrinking is one of the 'hit or miss transformation' operations.

This will reduce the objects without holes to a single point where as objects with holes to a ring of pixels around a boundary with a common center. This feature can be especially useful when trying to process with a large number of small objects to get a count as well as observe the pattern boundaries such as in microbiological images/ astronomical images.

This has been demonstrated on the HW problem image 'stars.raw' which can be considered an astronomical image example.

II. Approach and Procedure

- First the input image stars.raw is read and converted to a grayscale image to threshold the image for getting a binary output.
- Thresholding is done by observing the output of the histogram of gray level image obtained in the above step.



Clearly the background occupies intensity levels <21. however, just to make sure that a clean binary image is gotten , the threshold is set at 30.

- Shrinking operation begins

There are four basic steps .

Step 1: Apply conditional masks to input image **F** mark the pixels to be shrunk in input image.
Step 2: **Fhat** is the conditional mask gotten in step 1. Now we apply unconditional shrink patterns.
Step3:

If **Fhat(i,j)=0**
Then output image **G(i,j)=F(I,j)**

Else if **Fhat(i,j)=1**
if **HIT , G(i,j)=F(i,j)**
if **MISS, G(i,j)=0;**

Step4: repeat this in a loop until the image is shrunk to a single dot output.

In a nutshell, shrinking can be summarised according to the equation

G(i,j)=X and [~M or P(M,M0,…M7)]

Where G is the output image, X represents the current pixel being processed and M0…M7 are neighbour pixels for unconditional pattern definitions

III. Results

Fig 29. Gray scale image of the original ‘stars.raw’

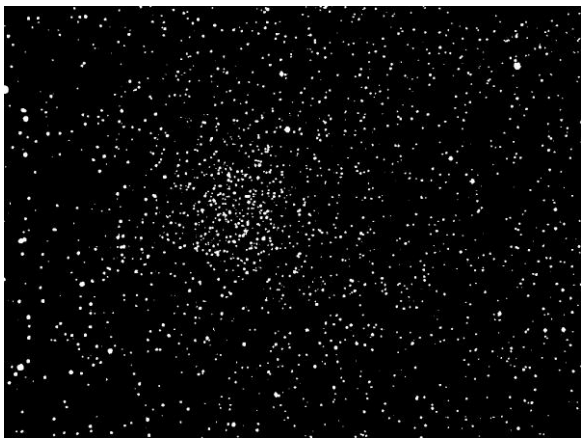


Fig 30. Binary image of stars.raw after thresholding

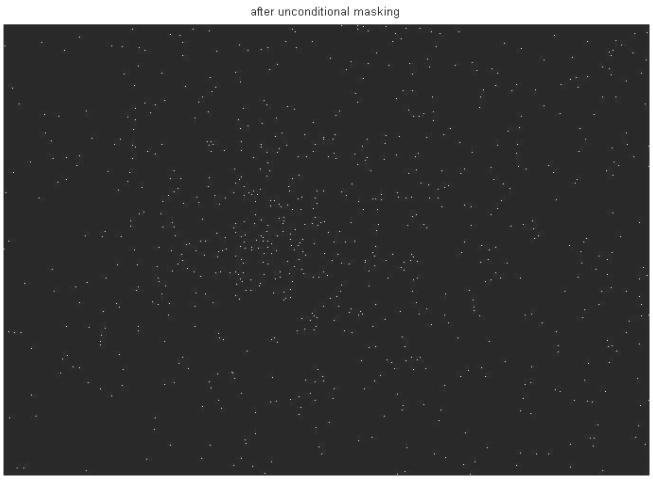


Fig.31 Output after unconditional mask in a loop to give single pixel dots

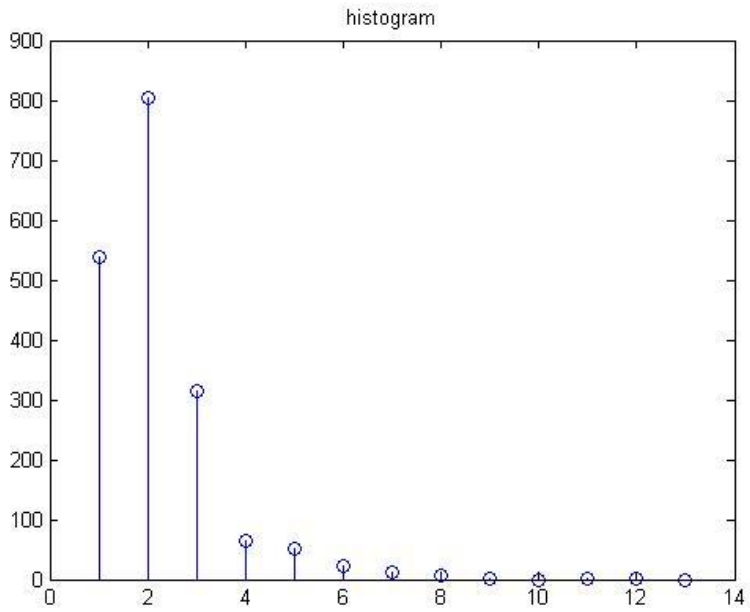


Fig.32 Number of stars (y-axis) vs. Iteration (Star size) (x-axis)

IV. Discussion and Conclusion

The threshold as discussed in section II, has been chosen as **T=30**, after observing the histogram of background v/s foreground pixels.

If pixel value is **<T** , it is made to be **0**, if it is >**T**, it is made to be **1** (binary value).

The Figure 30 shows what the thresholded binary image of the original 'stars.raw' input looks like.

Fig 31 shows the completely shrunken image after unconditional masking (stage 2 filtering) after **all** the iterations.

The logic to count the stars is basically dependent on how many isolated dots or pixels (shrunken pixels) we are counting. To keep a track of the size of stars, we count the number of iterations it takes to shrink all the stars completely and count how many stars have shrunk in each iteration,
We then plot a graph of **number of iterations** vs **corresponding shrunken pixels** which gives us the **star size(x) vs number of stars(y)** as shown in Fig 32.

Based on this algorithm,
The total number of stars = 1817

Star size(x)	**Number of stars (y)**
1	539
2	803
3	315
4	65
5	51
6	22
7	11
8	7
9	2
10	0
11	1
12	1
13	0

Note that on changing the threshold, the values are going to be different (both the number of iterations and the stars shrunk). This is because depending on the threshold , if it is very low, then the fainter stars having low brightness will also be considered thereby increasing the star count. On the other hand, the star count is inevitably going to decrease on increasing the threshold, as stars with higher intensity values are pushed into the background, meaning, they have been assigned) while binarizing.

2b) Thinning

I. Abstract and Motivation

Consider the cases of electronics hardware industry, material science and other manufacture industries. At times, it is very important to analyse the structural fitness of the products being manufactured, and often critical.

In such cases, the 'Thinning' operation can provide some help with the monitoring of regularly captured images and thereby allow the monitoring of the quality and/or the efficiency of the manufactured product/ product in manufacture.

Thinning is another 'Hit-or-Miss' Transformation performed , usually, on a binary image. According to [8], what thinning aims to accomplish can be defined as, "Erase black pixels such that an object without holes erodes to a minimally connected ring midway between each hole and its nearest outer boundary.

II. Approach and Procedure

This problem asks to apply thinning and then count the total number of digits.

For thinning, the procedure remains the same as shrinking except the unconditional (Stage I) masking patterns are different so as to keep a track of the edges.

As for counting the numbers, the procedure followed is as given in [10], but extended to 8-connected neighbours to cover the broken thin lines.

Summing up the procedure, it can be described as

1) Perform thresholding to cover most of the digit pixels.
2) Remove unwanted isolated dots.
3) Perform thinning as discussed in Problem 2(a) but with appropriate thinning unconditional and conditional masks.
4) Rectify the unconnected digits (as in cases of 3s, 0s, 6s, 9s) by connecting them.
5) Shrink the digits for counting in 6.)
6) Perform Connected Component Analysis as in [10] to count digits.

Hole filling by dilation can be tried before thinning, which is discussed in section IV.

III. Results

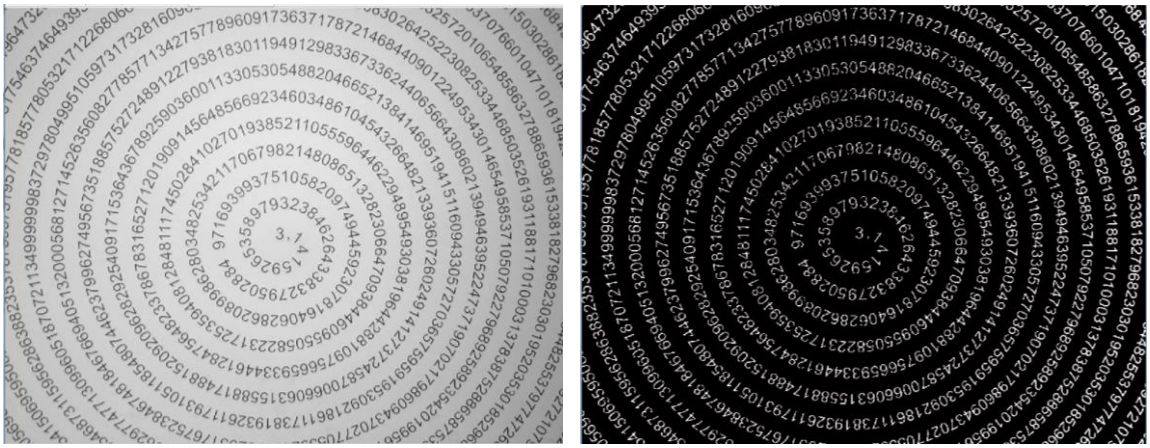


Fig 33. [L] grayscale image of 'digits.raw' and [R] thresholded image of 'digits.raw'

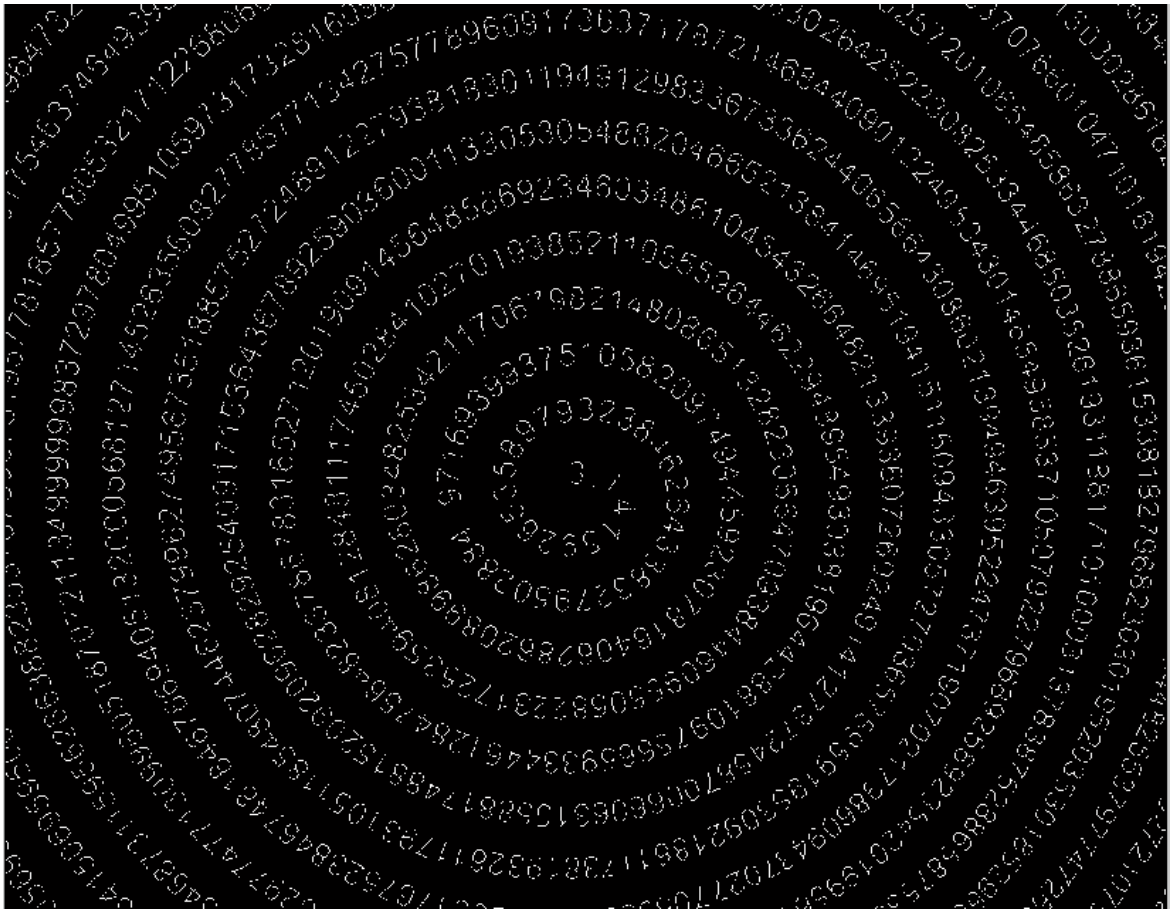


Fig 34 Thinned image of 'digits.raw'

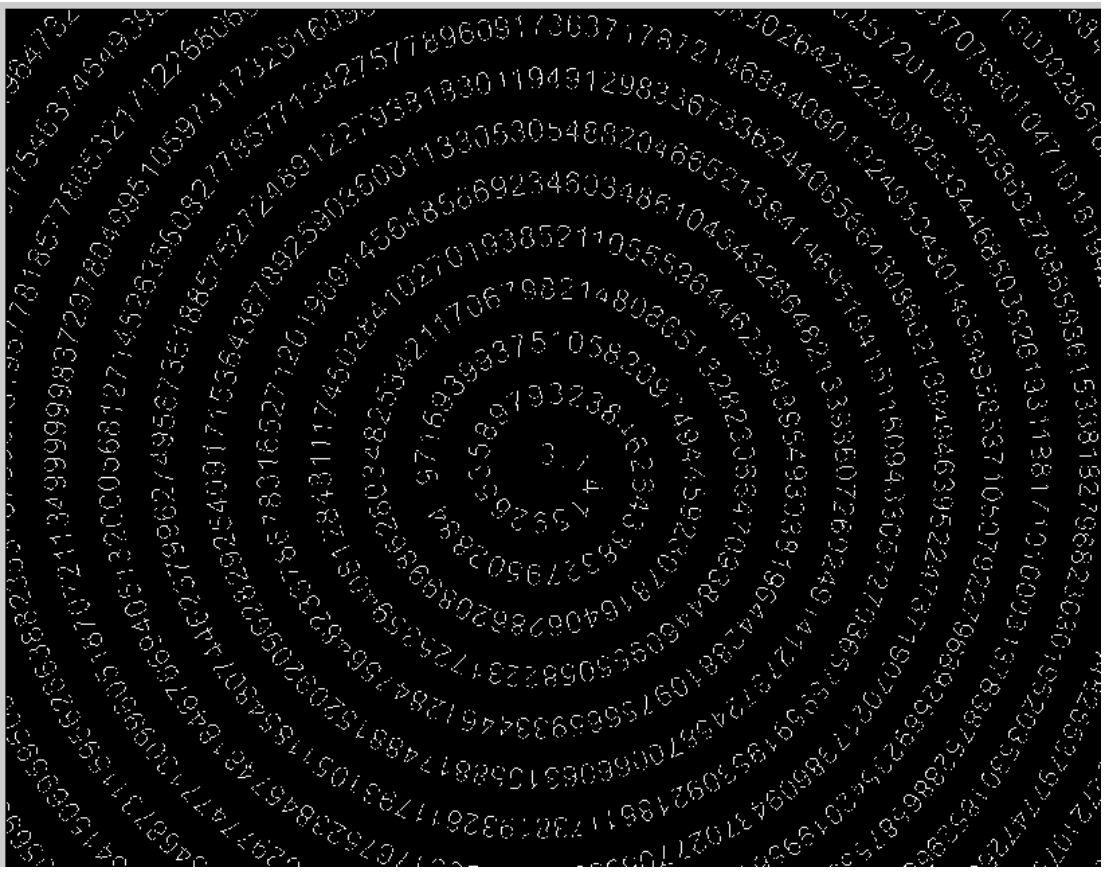


Fig 35. Corrected thinned image, after correcting connectivity issues to some extent

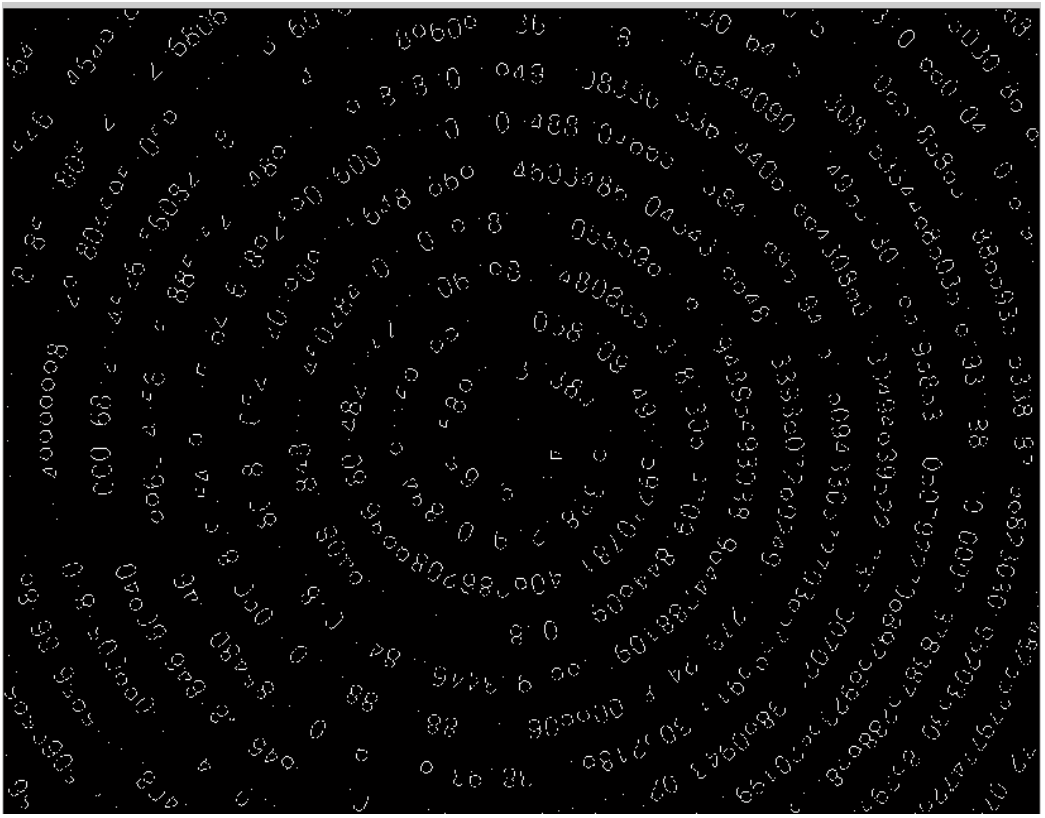


Fig 36. Figure 35 after shrinking. (notice how the 3s,0s,6s and 9s are still unconnected)

IV. Discussion and Conclusion

The challenge in this problem is to count the digits , not so much the 'Thinning' algorithm.

Thinning procedure remains the same as in Section II of Problem 2a
Step1: Apply conditional masks to input image **F** mark the pixels to be thinned in input image.
Step2: **Fhat** is the conditional mask gotten in step 1. Now we apply unconditional shrink patterns.
Step3:

If **Fhat(i,j)=0**
Then output image **G(i,j)=F(I,j)**

Else if **Fhat(i,j)=1**
if **HIT , G(i,j)=F(i,j)**
if **MISS, G(i,j)=0;**

Step4: repeat this in a loop until the image is thinned completely.

However, to count,

- The threshold is selected as about 1.5SD of the pixel values in the images so as to preserve most of the digit pixels for counting, which comes up to 123.
- After thresholding, the digits are set to **1,** if >T else set to **0,** if <T. (**Fig 33[R])**
- The digits are then thinned. However it is noticed that the digits such as 0,9,6,8 and 3 are broken on performing thinning.[**Fig 34]**
- Therefore they're tried to rectify by checking them with 16 mask conditions for curvature , for example

1	0	0
0	0	0
0	0	1

1	0	0
1	0	0
0	1	0

0	0	1
0	0	0
1	0	0

0	0	1
0	0	1
0	1	0

And so on.

- The digits are then shrunken to count, but it is noticed that some of the digits are not completely shrunk owing to holes in them.[**Fig 36**]
- There are two approaches here – to fill the holes or perform connected component analysis.
- **Hole filling by dilation** was attempted however the numbers which have no closed boundary result in unconnected/unfilled holes still, and the **count came up to 718.**
- Therefore , the **Connected Component Analysis** was done wherein

    - an array Q has the label index L.
    - L is incremented on finding a new 8-connected pixel
    - The goal is to end up with all of the pixels in each connected component having the same label and all of distinct connected components having different labels.
    - Those pixels which are assigned multiple labels even though in the same connected component, are dealt with as duplicates.
    - After finding the same label indices, those pixel values are reassigned from the original image , and count is incremented.

- Duplicates (987) are subtracted from the count gotten to get the **total count as 1310**, which is a pretty close estimation to the approximate manual count performed (~1000).

  Corner numbers are counted only if they have connected components, therefore connected component analysis may not be the most efficient way of calculating the corner numbers.

- In conclusion, after trying both Dilation and Connected Component Analysis, to get a more accurate result, it looks like more conditions of hole filling should be added, taking into account some more masks . Combining both dilation and connected component analysis is another option.

2c) Skeletonizing

I. Abstract and Motivation

As has been noted in problem 2a and problem 2b, morphological processing is particularly helpful for textural, and structural analysis.

Therefore, there are several practical areas or industries where these processes can be performed on images captured during the industrial processes.

Even otherwise, this can be used for road network analysis, agriculture to monitor health of crops, etc.

Skeletonization is more appropriate for the last two applications mentioned, given how it reproduces the structure of an object in an image, down to the most minute branchings of the same.

We demonstrate just how accurate the results are on an image 'dinosaur.raw' provided in the homework.

II. Approach and procedure

The input image 'dinosaur.raw' is first converted to a gray scale image and then converted to a grayscale image, and later thresholded with **T=mean** intensity value of the image.

Skeletonising is then performed on the image in two stages – Conditional masking and then Unconditional Masking, for several iterations until the image output consists only the structural shape of the dinosaur, and ignores all other details.

The procedure for skeletonising remains the same as in shrinking and thinning hit or miss transformations, except that the masks change.

Step1: Apply conditional masks to input image **F** mark the pixels to be skeletonised in input image.

Step2: **Fhat** is the conditional mask gotten in step 1. Now we apply unconditional skeletonization mask patterns.

Step3:

If **Fhat(i,j)=0**
Then output image **G(i,j)=F(I,j)**

Else if **Fhat(i,j)=1**
if **HIT , G(i,j)=F(i,j)**
if **MISS, G(i,j)=0;**

Step4: repeat this in a loop until the image is skeletonized completely.

III. Results

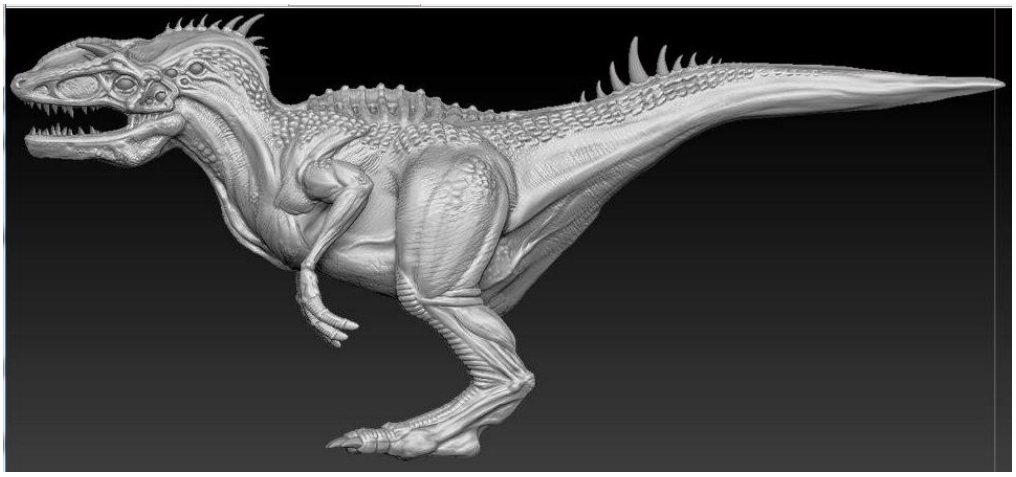


Fig 37 – Grayscale image of Dinosaur.raw

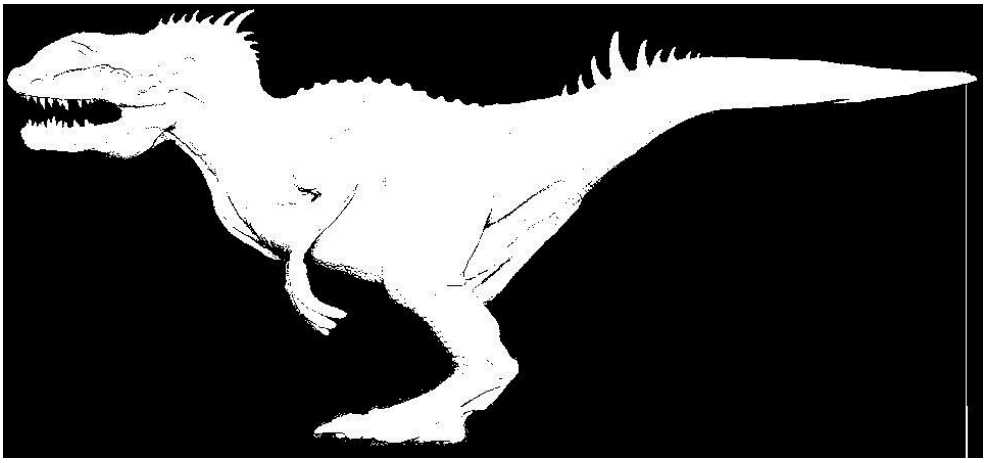


Fig 38 –Thresholded binary image of Dinosaur.raw

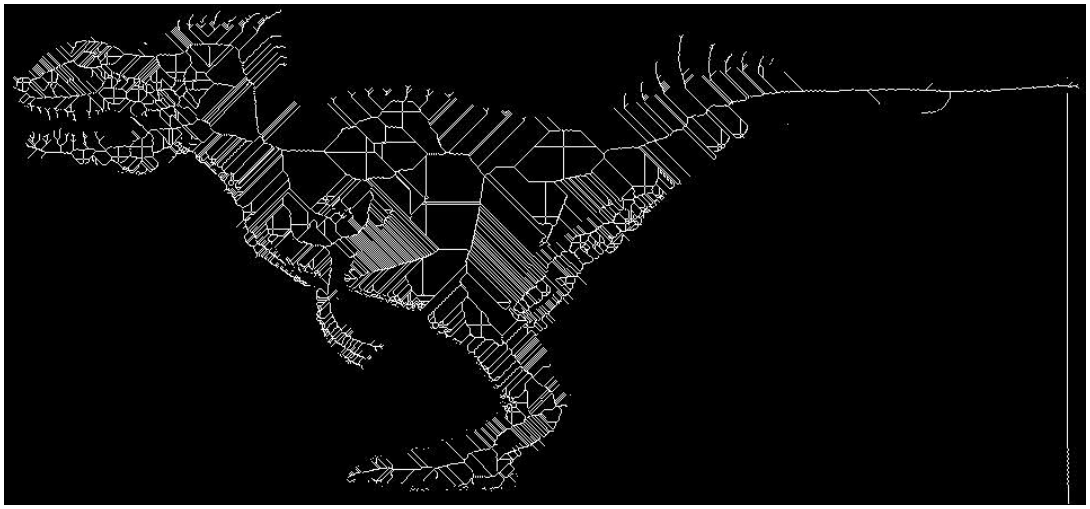


Fig 39 – Skeletonized image of Thresholded binary 'Dinosaur.raw'

IV. Discussion and Conclusion

As mentioned previously, the **threshold T** selected is the **mean** value of the image pixel intensities.

If pixel_value>T, then pixel_value=1
If pixel_value<T, then pixel_value=0

The thresholded image obtained is shown in **Fig 38.**
By selecting the threshold equal to **mean,** we are ensuring that the pixels that contribute to the very fine details of the image, and thereby its texture are preserved.

This can be corroborated by the result obtained in **Fig 39.**
It can be noticed that the details such as claws, fingers, teeth, the rimmed back and other minute textural details are preserved excellently.

Increasing the thresholds results in pushing several foreground pixels into the background, thereby increasing the speed of run-time as well as taking lesser number of iterations to give the skeletonised image . However, there is a very good possibility that all the details of the structure of dinosaur are preserved in such a case with threshold considerably larger than the mean.

No pre-processing or post-processing techniques were as such employed for this image, however they might be necessary if the image is not clean and does not give a clear demarcation between the background and the foreground, or has intermittent noise.

The output skeletonized image looks very detailed and hence no post processing has been employed.

2d) Pacman Game

I. Abstract

The aim of this problem is to combine and apply all the concepts learnt about morphological processing, and is much more realistic in the sense that the problem statement is close to real world applications of morphological processing, starting from threshold based segmentation to separation of required features for analysis, and thereby reaching a conclusion about the result obtained.

The pacman game screenshot is the input image and the goal is to count

- the point balls leaving out the bigger ones that are irrelevant to the problem statement,
- the walls that could block Pacman's path
- The turns that the Pacman can take at any intersection.

II. Approach and Procedure

The steps followed for getting the results are as follows :

1) Convert the colour image to a grayscale one.
2) Observe the intensity ranges of the point balls and blue walls using histogram or figure window to set thresholds for binary separation.
3) Separate the binary images of image containing point balls and walls.
4) To count point balls – perform shrinking until the count got is a non-zero number (so that only point balls are considered and not the bigger white balls)
5) To count walls- perform 8-connected component analysis (as in Problem 2b- for counting digits) on the binary walls image
6) To count turns – skeletonize the Walls image and perform 8-corner masking. Subtract 4 from final count to exclude 2 turns each for the 2 outer walls (2outerwalls*2 turns)

   Some examples of the corner masks are as below -

0	1	0
0	1	0
0	1	0

0	0	0
1	1	0
0	1	0

0	0	1
0	1	0
0	0	1

0	0	0
0	1	0
1	0	1

The procedure for 8-connected component analysis is briefly described in Section IV of Problem 2b) [Page 36] reproduced below

- Therefore , the **Connected Component Analysis** was done wherein
    - an array Q has the label index L.
    - L is incremented on finding a new 8-connected pixel
    - The goal is to end up with all of the pixels in each connected component having the same label and all of distinct connected components having different labels.
    - Those pixels which are assigned multiple labels even though in the same connected component, are dealt with as duplicates.
    - After finding the same label indices, those pixel values are reassigned from the original image , and count is incremented.
- Duplicates are subtracted from the count gotten to get the **total count**.

III. Results

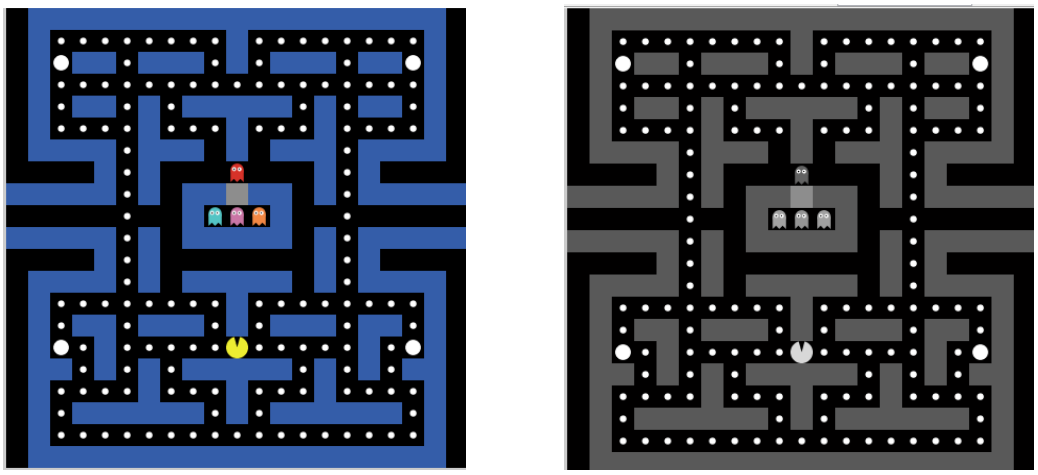


Fig 40. [Left] Original Colour Image and [**Right]**Grayscale image of Pacman.raw

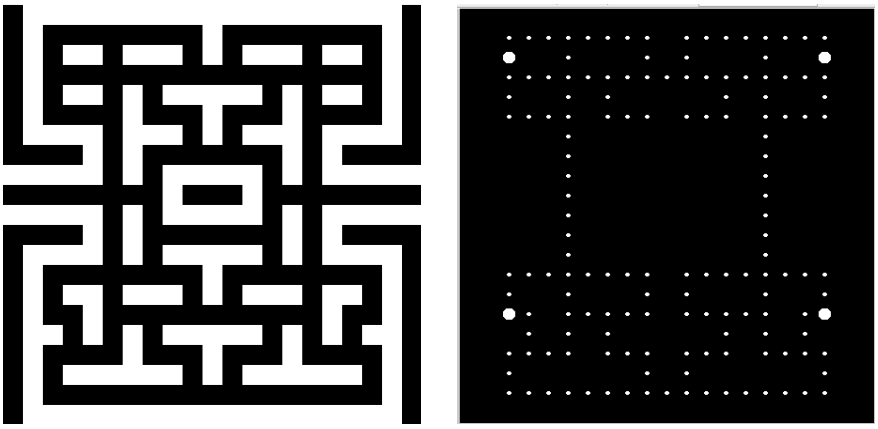


Fig 41. [Left] Binary walls image and [**Right]** Binary point ball image of pacman.raw

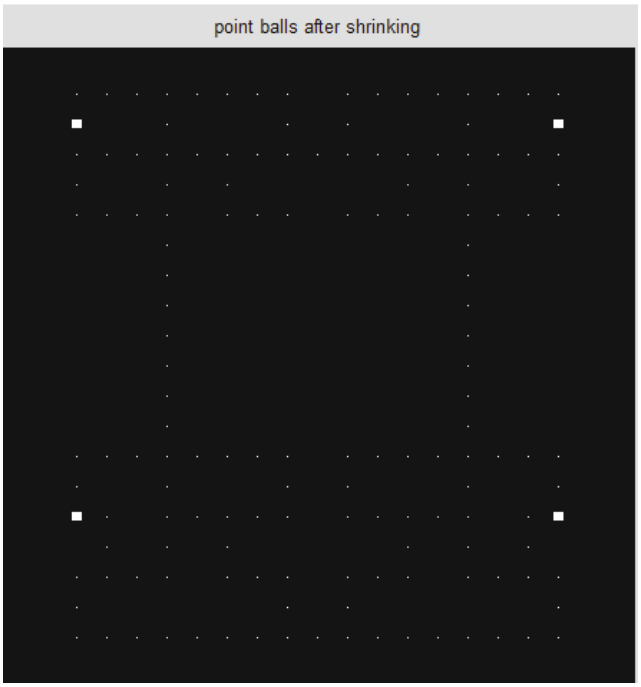


Fig 42. Point balls after shrinking completely.

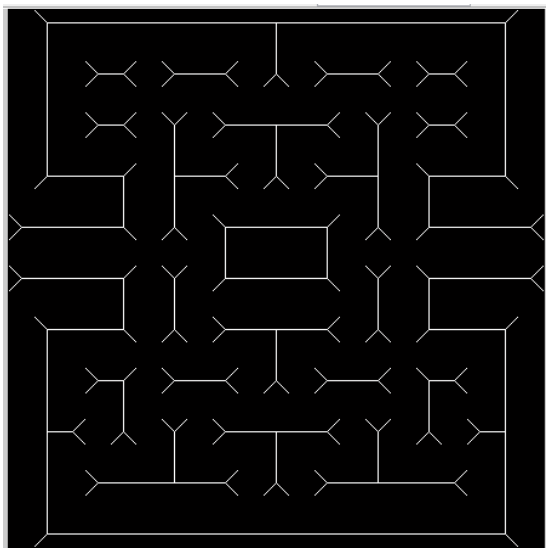


Fig 43 Skeletonized image for applying corner masks for counting turns

IV. Discussion and Conclusion

➔ After converting the colour images to grayscale, two different binary images by thresholding are obtained.

For a binary image containing only walls, the threshold is set at T= 89 or T=141 (for the center wall)

- If pixel_value > T, then binary_wall image = 1;
  Else binary_wall image = 0;

For a binary image containing point balls, the threshold is T=255.

- If pixel_value=T, then image(point_balls)=1;
  Else image(point_balls)=0;

The results for this can be seen in **Fig 41.**

➔ The binary image containing point balls is then shrunken until the smaller point balls are completely shrunk . this is done by adding a condition in the loop to check whether the count of isolated dots is greater than 0. If it is, then it implies that the smaller point balls have completely shrunk whereas the bigger ones are still left out. **Break**ing out of the loop, we have our final point ball count

Number of Point Balls = 146

➔ For counting the walls, each blue wall block is considered as one wall only, and then connected component analysis is performed on it (procedure is described as above in section II) .

Number of Walls = 22

Note: the whole blue block which the pacman cannot cross is considered to be **one** wall and the direction of the turns for this has not been considered.

This has been done in accordance with what is marked as a wall on the Homework problem statements file.

➔ For counting the turns, each corner in a wall has been considered as a turn. Again, the directions have **not** been taken into consideration, for example, if there is an intersection, the pacman could turn either left or right. This has not been considered. Assumption made is that the pacman starts at one corner, and traverses the entire maze thereby counting all the wall corners encountered by the pacman on its trail, which have been considered as turns.

By this assumption, the binary walls image is first skeletonized so as to capture the corners.

The neighbourhood pattern for each pixel is then compared against the 16-corner mask patterns considered.

If it matches the Vee-corner mask, then count is incremented by two else by one.

In the end, it is seen that the outer walls give 4 corners or turns which cannot be taken by the Pacman. These are the outer boundaries. Hence, the final count is subtracted by 8 to get the exact count of turns.

Number of turns = 138

----------END OF PROBLEM 2----------

REFERENCES:

[1]http://dasl.mem.drexel.edu/alumni/bGreen/www.pages.drexel.edu/_weg22/edge.html

[2]http://www.cs.utah.edu/~germain/PPS/Topics/index.html
[3]http://www.eng.iastate.edu/ee528/sonkamaterial/chapter_4_3.htm

[4]http://www.hpl.hp.com/research/isl/halftoning/publications/DitheringWithBlue.pdf
[5]http://alamos.math.arizona.edu/~rychlik/CourseDir/535/resources/RasterGraphics_slides.pdf

[6]http://www.livescience.com/38583-what-is-blue-noise.html

[7]"Inverse Halftoning" by Christopher M. Miceli, Kevin J. Parker, University of Rochester, 1992.
[8] http://www.cyut.edu.tw/~yltang/course/image%20processing/notes-ip09.pdf
[9] "Digital Image Processing" by William K Pratt , Fourth Edition
[10] http://www.cis.rit.edu/class/simg782.old/finding_connected_components.pdf

----------END OF REPORT----------