EE569: Digital Image Processing

Homework #2

Yogesh Sanat Gajjar USC ID: 5476153636

Issued : 01/29/2020 Due: 02/16/2020

1. Edge Detection

Edge detection is a very important as well as widely used technique to find the edges or boundaries of object within images. It is an algorithm which essentially finds the region in an image where there is a sharp change in intensities or sharp change in color. A high value or dark edge indicates steep change and a low change or light edge indicates a shallow change. [1]

The motivation behind detecting edges in an image is to capture important information and change of properties of the world. Moreover, discontinuities in the brightness are likely to correspond to

- Discontinuities In depth
- Discontinuities of surface orientation
- Changes in material property
- Variations in scene illumination.

Performing an edge detection algorithm on an image significantly reduces the amount of data to be processed and may therefore filter out the information that may be regarded as less relevant, while preserving the important structural properties of an image. In edge detection, we get an edge map which is the values are each pixel represents its edge-ness score. It is widely used in the object recognition, object tracking, image retrieval and scene parsing.

Edges can be also modelled according to their intensity profiles. These are as follows :

- Step Edge
- Ramp Edge
- Ridge Edge
- Roof Edge

The three-edge detection algorithm used in this assignment are :

- Sobel Edge Detection
- Canny Edge Detection
- Structural Edge Detection

1.1. Sobel Edge Detection

Sobel Edge detection is one of the very primitive work on the field of edge detections. The Sobel operator, sometimes called the Sobel-Feldman operator or Sobel filter is a used in Digital Image Processing and Computer vision applications where it creates an image emphasizing edges. It works on the principle of first order derivative of the pixel also called as gradient of an image. Sobel operator is a 3x3 kernel for each of the two directions of an image. The sobel matrix also called as gradient matrix for x direction consists of minus numbers on the left-hand side and positive numbers on the right-hand side while preserving the data in the center pixel. Similarly, the gradient for y-direction has minus numbers on the bottom and positive numbers on top and here we are preserving a little bit on the middle row pixels. [2]

The Sobel operators in x-direction and y-direction are as follows.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Approach and Procedure

Sobel edge detector scans the image horizontally and generates an intensity function along that line. It then calculates the first derivative of the change of pixel intensities called as intensity gradient. As the detector scans, it generates peaks where it encounters a change in intensity. Here, the edges correspond to the extrema of derivative.

A gradient is calculated by a the partial derivates of the image by the partial derivative in the x & y direction. Also, the magnitude and orientation of the gradients is calculated as given below.

Gradient

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Magnitude

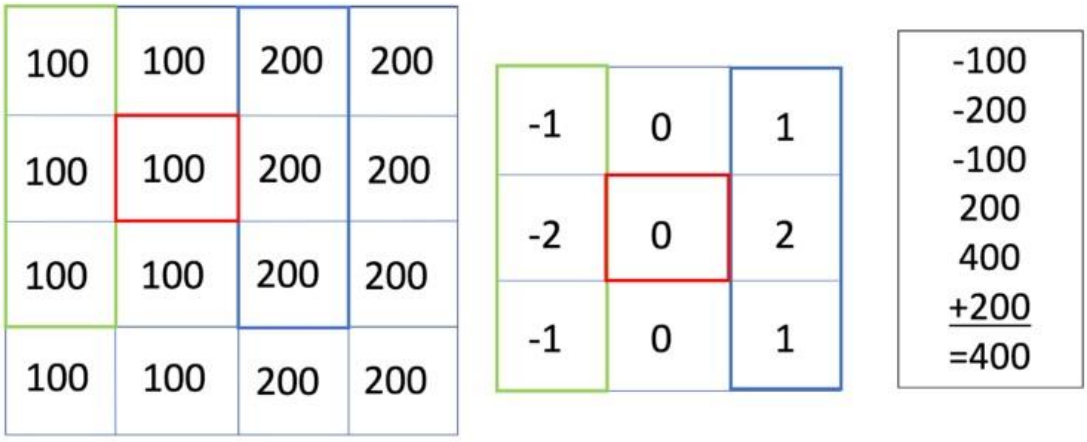
$$|\nabla f| = \sqrt{g_y^2 + g_x^2}$$

Orientation

$$\theta = \tan^{-1}\left[\frac{g_y}{g_x}\right]$$

The 3x3 kernel Sobel Operator essentially scans over every pixel of the image and finds the amount of difference between the adjacent pixels. The G_x operator calculates the x component of the gradient and the G_y operator calculates the y component if the gradient. By using Kernel Convolution, we multiply the pixel values to the corresponding Sobel operator values, add them to determine if there's an edge. We can see in the example image below there is an edge between the column of 100 and 200 values because the pixel value in the

center after convolution is positive.



Kernel Convolution: The bigger the value at the end, the more noticeable the edge will be.

If all the pixels of images were of the same value, then the convolution would result in a resultant sum of zero. So, the gradient matrix will provide a big response when one side is brighter.

Normalization : Now, in order to bring our image in the range of o to 255, we perform normalization. With normalization, we bring back the gradient components in a fixed range. This is essentially performed in both the directions as well as to the final gradient map so that we can capture all the essential features in x and y directions.

The linear normalization of a grayscale digital image is performed according to the formula

$$I_n = (I - Min)\,\frac{newMax - newMin}{Max - Min}$$

Where I_n is the normalized output, I is the input channel, Max, Min are the minimum and maximum pixel value in the image. $newMax, newMin$ are the new range of values to be normalized with.

Thresholding : Thresholding in a simple technique to create a binary image from a grayscale or full-color image. With thresholding, for a threshold, below that gradient value is found to be weak and above that provides us a good pixel edge. Thresholding is a good way to represent a continuous pixel image into a binary image consisting of either white dot or black dot.

The algorithm used in determining the edges using Sobel Operator is as follows.

1. RGB to Grayscale – This step is essential as finding edges in a grayscale image is easy because of single channel with black to white intensities. The formula to convert an RGB image to Grayscale is given as:

$$Y = 0.2989 * R + 0.5870 * G + 0.1140 * B$$

2. Boundary Extension – This step is a pre-requisite step of convolution. I used mirror reflection boundary extension method. Once the image is boundary extended, it can be used in convolution with a filter. As the sobel operator is a 3x3 kernel, I extended boundary by 1 pixel on each side. The padding is calculated with the formulae

$$padding = \frac{windowSize}{2}$$

3. Convolution – Performed convolution using both the sobel filters to calculate the gradient in both the x and y directions. The boundary extended image is used for convolution to generate three different image matrices. The three-image matrix are gradient matrix in x direction, gradient matrix in y direction and the final image which combines both the gradient matrix using the formula,

Gradient

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Magnitude

$$|\nabla f| = \sqrt{g_y^2 + g_x^2}$$

Orientation

$$\theta = \tan^{-1}\left[\frac{g_y}{g_x}\right]$$

4. Image Normalization – To normalize the x-gradient and y-gradient to values 0-255.
   To do this, the program calculates the minimum and maximum pixel value of the image. In a different loop, we do the image normalization after calculating the min and max values using the formula

$$I_n = (I - Min)\,\frac{newMax - newMin}{Max - Min}$$

5. Thresholding – Generates an output image based on inverse thresholding method. The intensity value corresponding to the pixel threshold is given as

$$OutputImage = 0\ if\ inputImage > threshold$$
$$= 255\ if\ inputImage < threshold$$

   The threshold value is calculated from the CDF. We know that the CDF of an image lies in the interval of [0,1]. So, a threshold value in the range of 0-100% is essentially a pixel value at that CDF. For example, 90% threshold means that we allow the pixel value to generate a edge about based on the CDF value above 90%.

ALGORITHM IMPLEMENTED

1. Converted the RGB image to Grayscale using the formula provided in the assignment.
2. Boundary extension on all the sides. Used mirror reflection method.
3. Initialized the sobel operators and performed convolution on the boundary extended image to find x_gradient map, y_gradient map and gradient magnitude map.
4. Performed normalization by calculating the maximum value of the pixel and dividing the map with the max value and multiplying it with 255.
5. Calculate the CDF of the normalized final image. Identify the pixel value at that value when the CDF is threshold.
6. Performed thresholding based on the CDF of the image.

Experimental Results

Part A – Normalized X-Gradient and Y-Gradient Images for "Dog.jpeg" & "Gallery.jpeg"

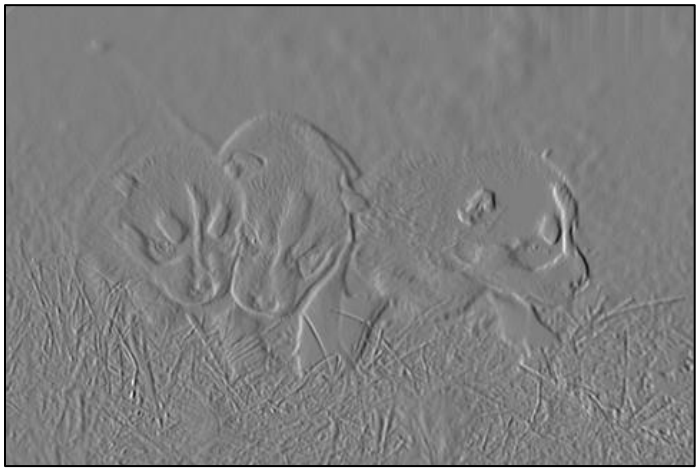


Fig. 1 X – Gradient using G_x Sobel Kernel Kernel

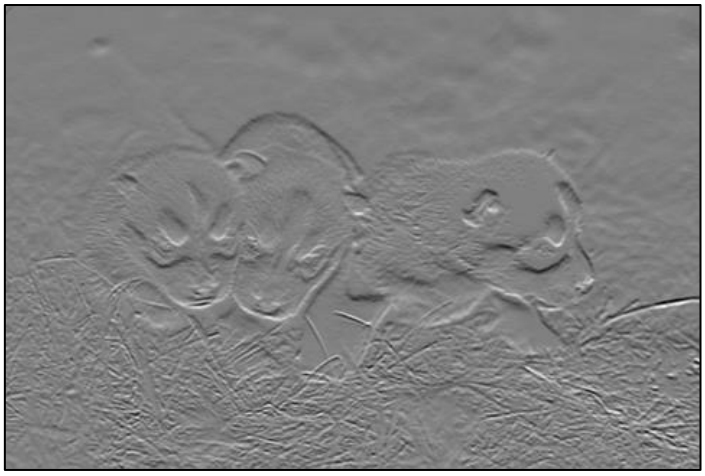


Fig.2 Y – Gradient using G_y Sobel

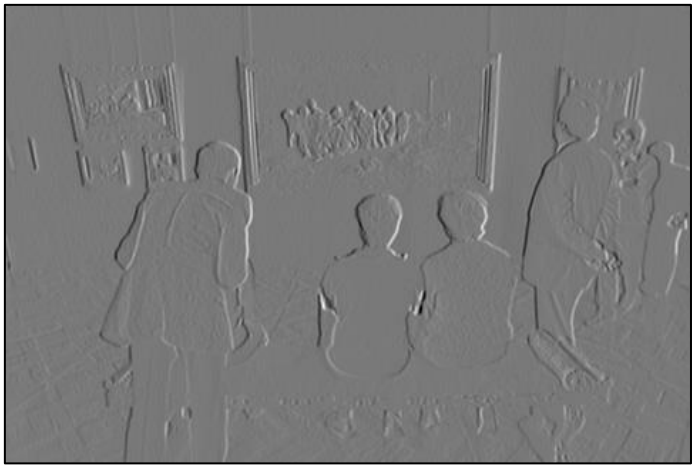


Fig.3 X – Gradient using G_x Sobel Kernel

Fig.4 Y – Gradient using G_y Sobel Kernel

Part B – Normalized Gradient Magnitude Map for "Dog.jpeg" and "Gallery.jpeg"

Fig. 5 Normalized Magnitude Map

Fig. 6 Normalized Magnitude Map

Part C - Threshold tuning

Fig.7 99% thresholding

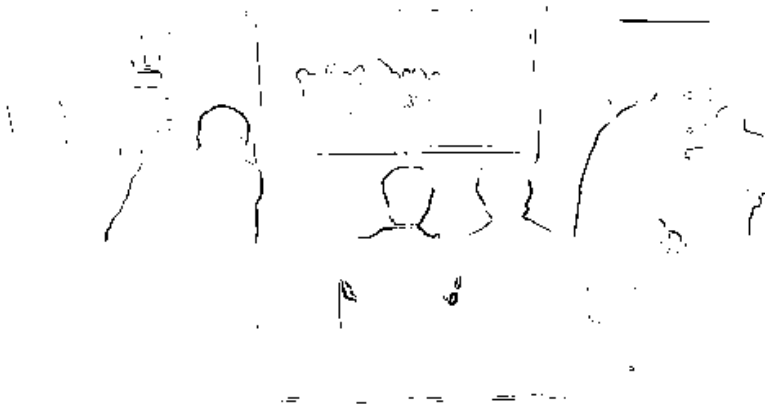


Fig. 8 95% thresholding

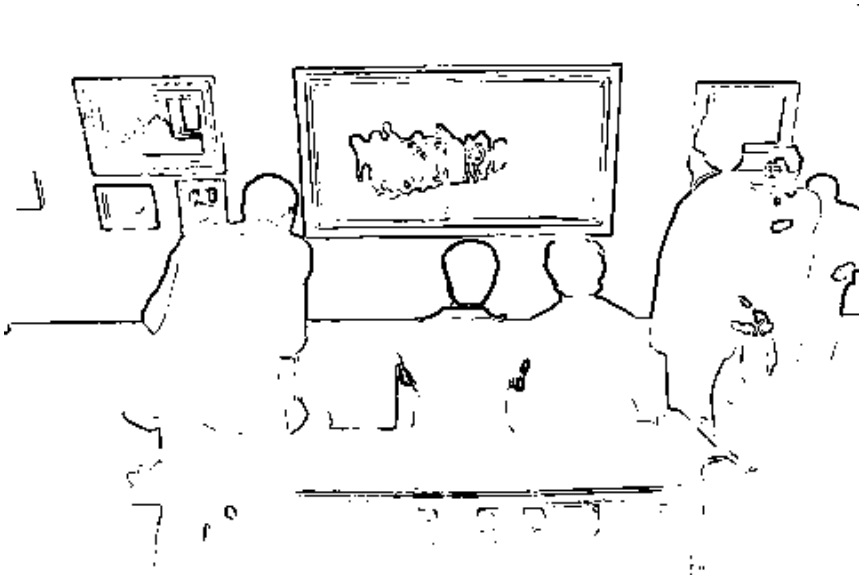


Fig. 9 90% thresholding

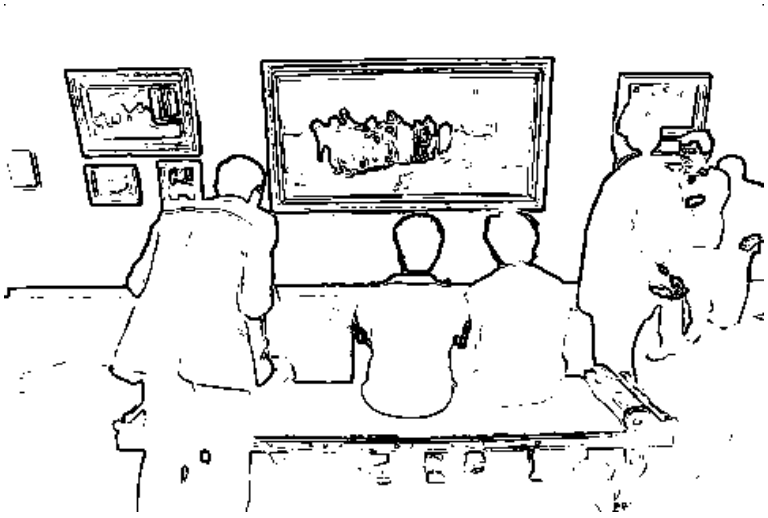


Fig. 10 85% thresholding

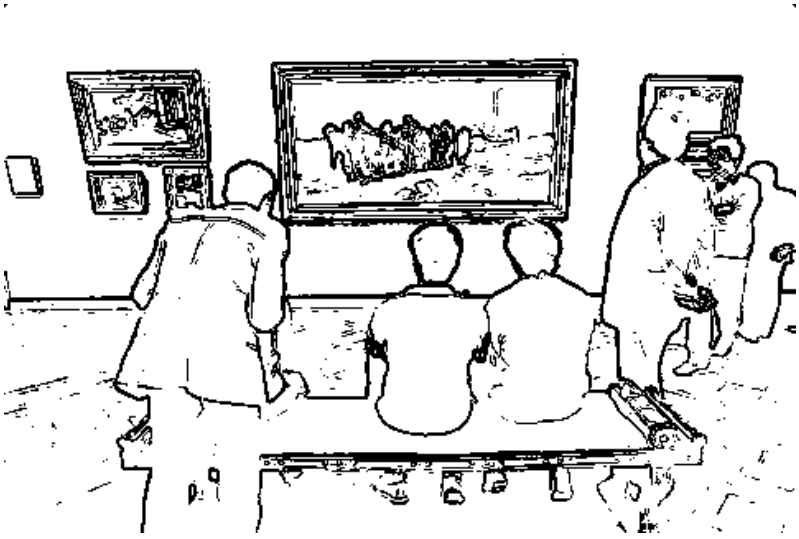


Fig. 11 99% thresholding

Fig. 12 95% thresholding

Fig. 13 90% Thresholding

Fig. 14 85% Thresholding

Analysis

In this question, we saw the x-gradient , y-gradient, gradient map, normalized gradient map and edge map with different thresholds of both “Dog” and “Gallery” input image. On comparing the edge maps with different thresholds, I found had few observations as follows:

- Sobel edge detection is susceptible to noise. When we decrease the threshold, the image captures edges which are essentially noise. To improve the performance of the detector, the input image requires a filtering before performing the edge detection.
- The edge in the “dog” as well “gallery” image is big and blurry. If we see the normalized edge map, we find the edge being thick and inconsistent. The detection method requires non-maximum suppression method which makes the edges thin and crisp.
- Threshold being calculated based on CDF selects the pixel value which result into an edge if found below the threshold. Thus, on increasing the threshold, reduces the edge in the final image and on decreasing the threshold, increases the edge of the final image.
- Edge map of the “dog” image can’t differentiate the grass clearly. This makes it difficult to guess the where the dogs are sitting. This is due to thick edges generated by the Sobel detector and noisy input image.

Best threshold value for both images – 90% as it captures the edges properly and free from the noise. Moreover, it conveys most of the information of the input image.

1.2. Canny Edge Detection

Canny edge detector is an update to the Sobel Edge detector in terms of processing the image to improve the quality of the edges in the image. Canny edge detection uses a multi-stage algorithm to detect a wide range of edge in the image. It is a technique that essentially extracts the useful structural information from different images and reduces the amount of data to be processed. It uses the calculus of variations – a technique that finds the function which optimizes a given functional. The optimal function in Canny's detector is described by the sum of four exponential terms, but it can be approximated by the first derivative of a Gaussian. [3][4]

The Canny edge detection algorithm is composed of 5 steps

- Noise reduction
- Gradient calculation
- Non-maximum suppression
- Double threshold
- Edge tracking by hysteresis.

Approach and Procedure

The Canny edge detector algorithm works using 5 steps. [4]

1. Noise Reduction
   This step ensures that the image is noise free. We saw in Sobel Edge Detector that the edges at lower thresholds generated noise which was captured using edges. Canny uses Gaussian blur to smoothen the image before performing the actual convolution. The different kernel size is are chosen depending on the expected blurring effect.
   The equation for a Gaussian filter kernel of size (2k+1) (2k+1) is given as

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \le i, j \le (2k+1)$$

2. Gradient Calculation
   This step detects the edges intensities and directions of the gradient by using Sobel Operators. When we smoothen the image, the sobel operators are convolved over the main image to generate derivatives or gradients in x-direction and y-direction. This is again followed by pixel normalization to bring it to uniform and same intensities.

Gradient

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Magnitude

$$|\nabla f| = \sqrt{g_y^2 + g_x^2}$$

Orientation

$$\theta = \tan^{-1}\left[\frac{g_y}{g_x}\right]$$

3. Non-Maximum Suppression
   This step ensures that the largest edge is captured. After the convolution with Sobel operators, the edge remained blurred. Non-maximum suppression method ensures that the final image consists of the edges which are most prominent and sharp. The principle behind the non-maximum suppression is the algorithm goes through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge directions.

4. Double Threshold
   This step helps in determining three different kind of pixels intensities. The three-pixel intensities stand for a strong, weak and non-relevant. The strong pixels have high intensities that's contributes to the final edge. Weak pixels lie between strong pixels and non-relevant pixels and do contribute to the edge. The non-relevant intensities do not contribute for the edge.
   a. High Threshold – Identifies strong pixels (higher than the high threshold)
   b. Low Threshold – Identifies the non-relevant pixels (lower than low threshold)
   c. Intensities between the thresholds are considered as weak and uses Hysteresis mechanism to identify between strong and non-relevant.

5. Edge Tracking by Hysteresis
   This step comes into consideration when we have intensities between the low and high thresholds. It consists of transforming weak pixels into strong once if and only if at least one of the pixels around the pixel being considered is a strong one.

ALGORITHM IMPLEMENTED

1. Read the input image using imread and initialize variables as Mat datatype
2. Convert the image into grayscale image using the function CvtColor.
3. Call the canny function to generate edges. Note: Canny function has Guassian blur function inbuilt. Also, provide lower and upper threshold values.
4. Write the image into a file.

Experimental Results

Fig. 15 Low Threshold – 0, High Threshold – 255

Fig. 16 Low Threshold – 85, High Threshold – 170

Fig. 17 Low Threshold – 100, High Threshold – 200

Fig. 18 Low Threshold – 10, High Threshold – 35

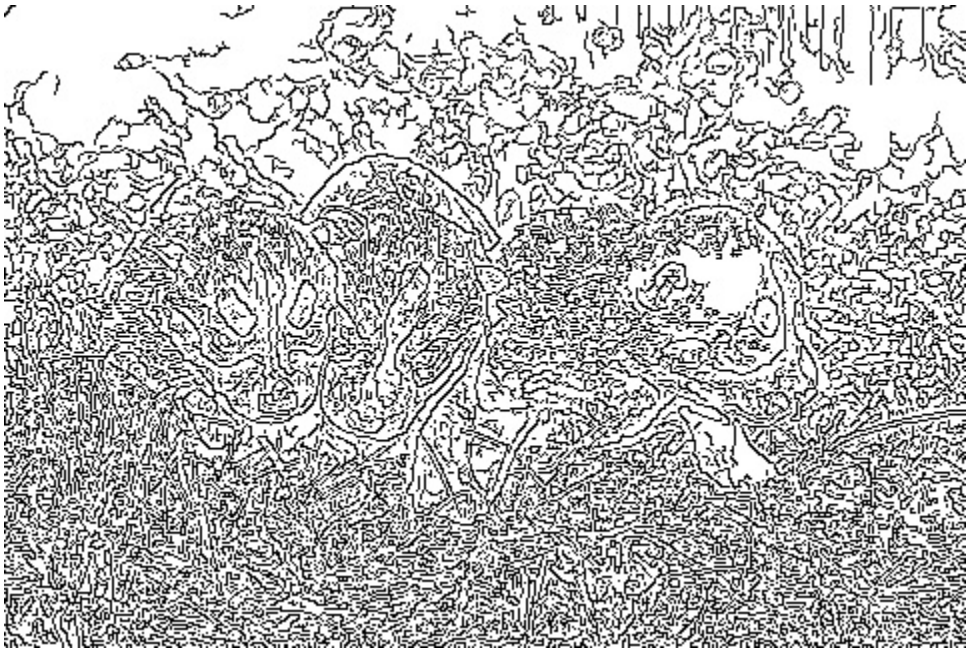


Fig. 19 Low Threshold – 30, High Threshold – 200

Fig. 20 Low Threshold – 160, High Threshold – 200

Fig. 21 Low Threshold – 90, High Threshold – 150

Fig. 22 Low Threshold – 234, High Threshold – 250

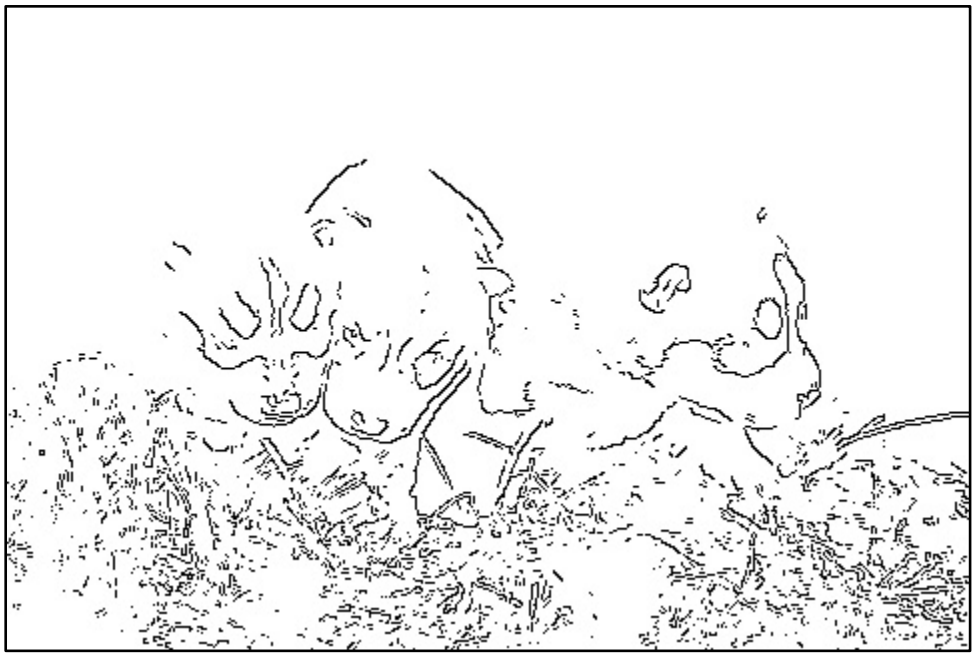


Fig. 23 Low Threshold – 10, High Threshold – 35

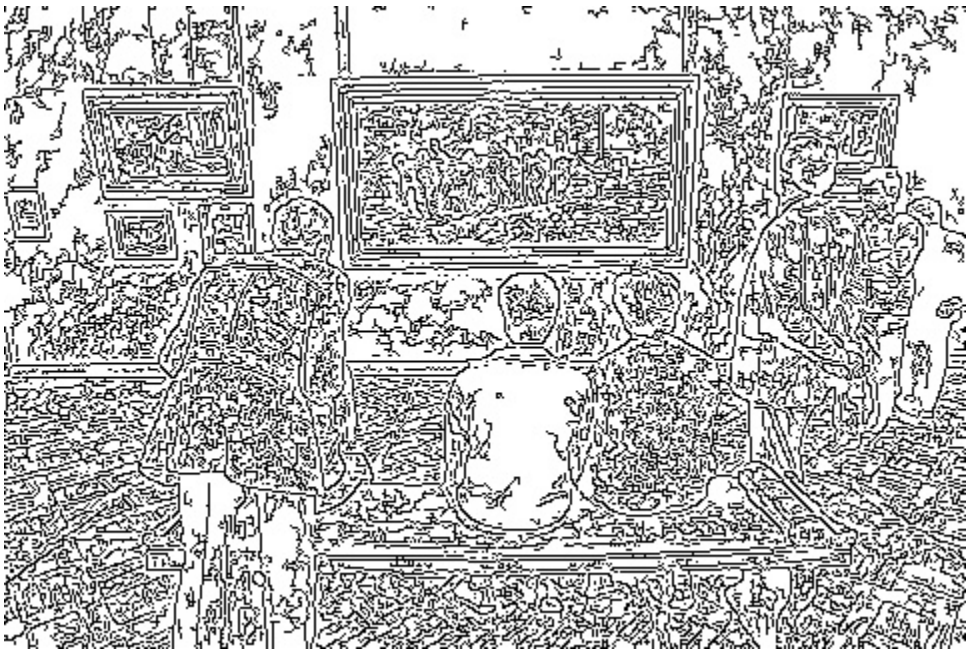


Fig. 24 Low Threshold – 0, High Threshold – 255

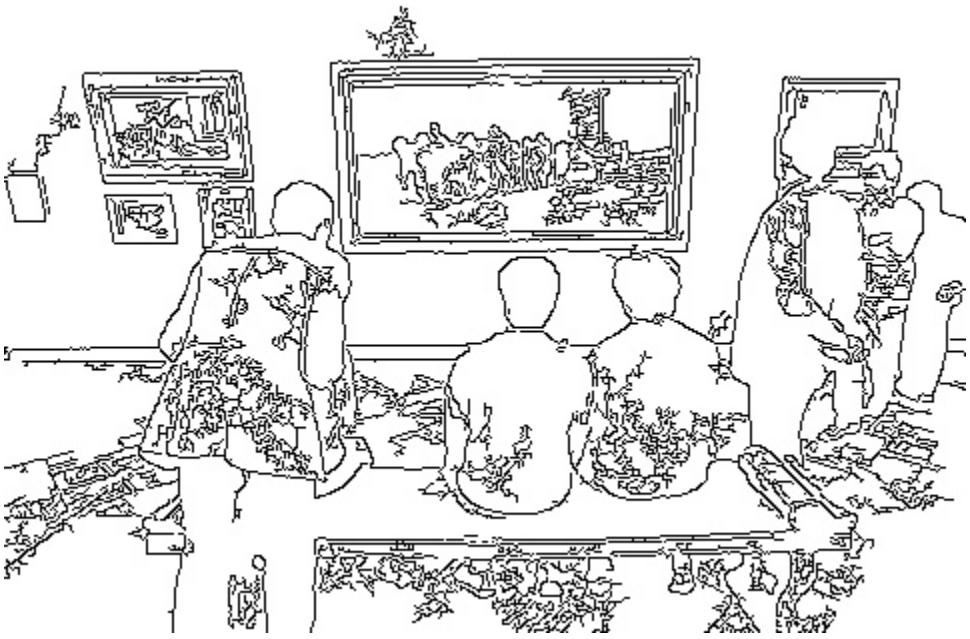


Fig. 25 Low Threshold – 30, High Threshold – 200

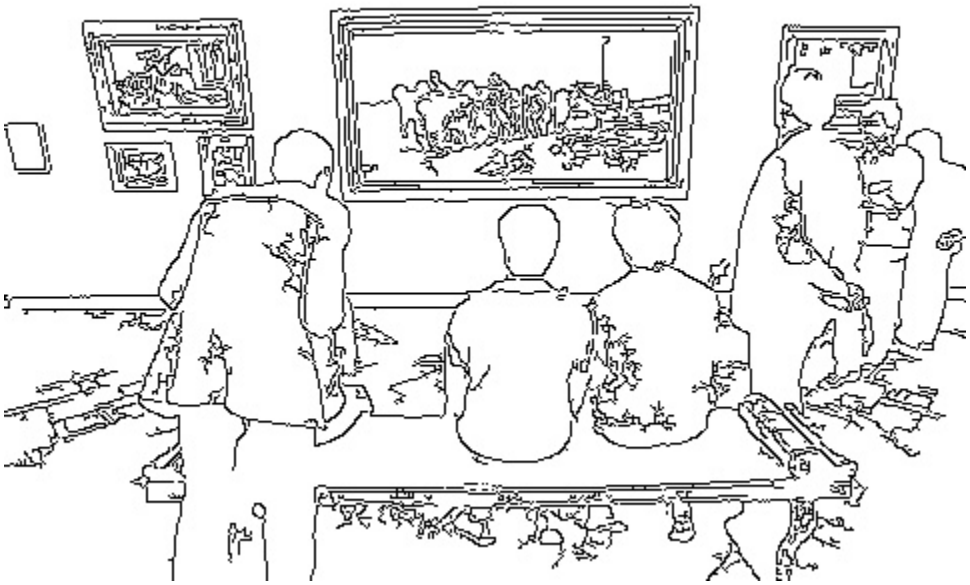
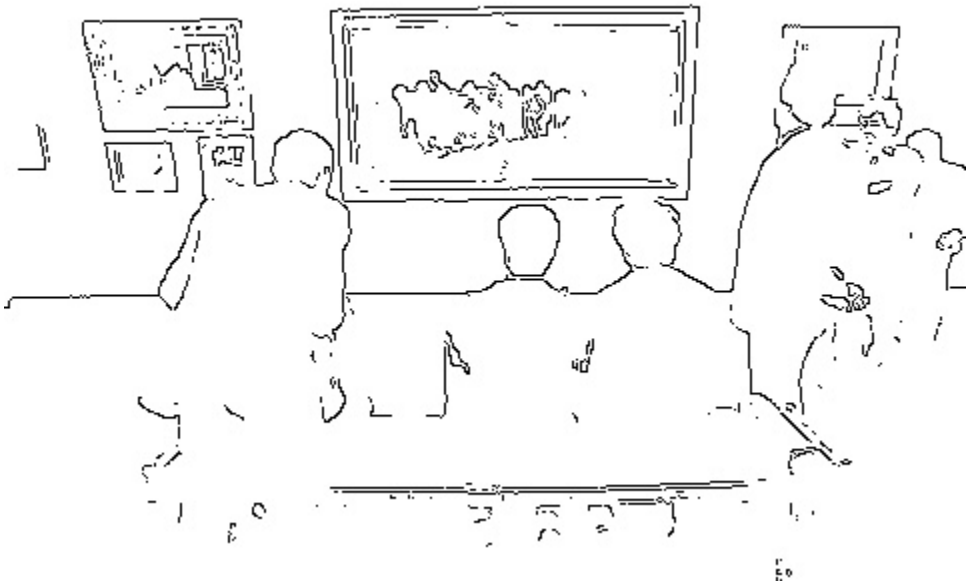


Fig. 26 Low Threshold – 230, High Threshold – 250



Analysis

Canny edge detection improved the edge map of the image drastically. We can observe crisp and clear edge boundaries as compared to the Sobel edge detection algorithm. This algorithm works gives good results because it works on the preprocessing of the image before detecting edges and parallelly uses post-processing such as Non-maximum suppression and double thresholding to improve the result or edges maps. However, Canny edge detector has some limitations. It fails to avoid detecting uninformative edges present inside an object.

Non-Maximum Suppression(NMS)

The difference in the edge map generated by Sobel and Canny is the intensity of the edges. When we compare the results of edge map generated by both the methods, we see the edge in more thin, crisp and clear. So, essentially to find clear and thin edge from the blurry and thick edges, the non-maximum suppression is introduced. In Canny edge detection, the algorithm records the value of gradient as well as the direction of the gradient at each pixel.

Non maximum suppression algorithm works on the edge map generated by the Sobel operators in both x & y direction. It goes through every pixel in the edge an finds the pixel with the maximum value present in that edge. It finds the gradient direction and looks over to the image pixels in both the positive and negative gradient direction. If the algorithm finds the edge pixel to be the largest one when compared to the neighboring pixels in the gradient direction, it makes it a prominent edge and preserves it. If the algorithm fails to find, it ignores the edge.

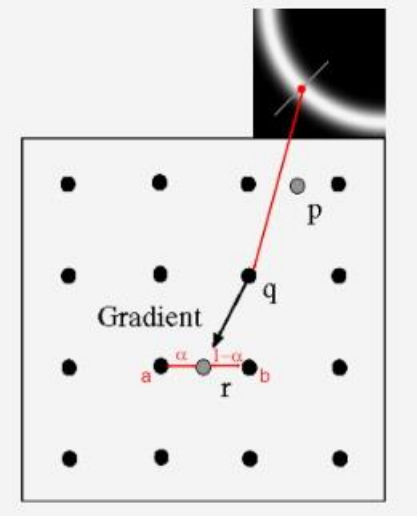


Fig. 27 Non-Maximum Suppression

In the above image, the gradient direction is shown with an arrow. The points p and r lie on the positive and negative direction of the edge pixel q. If q remains larger tan p and r in terms of pixel intensity, the algorithm preserves the pixel.

Non-maximum suppression can be achieved by interpolating the pixels for greater accuracy and is given by

$$r = \alpha b + (1 - \alpha)a$$

Double Thresholding (Hysteresis Thresholding)

In order to gain more flexibility over the thresholding, instead of single level thresholding, the Canny edge detector implements double thresholding and identifies three different types of pixels; strong, weak and non-relevant. These three-pixel intensities contribute to the final edge. The strong pixels are the pixels which are high and maintain the edge boundaries. Weak pixels are the pixels which are considered to contribute the edge boundary but not like the strong pixels. The remaining ones fall into the category of non-relevant pixels.

In double thresholding, the high threshold searches for the strong pixel intensities and if found, accepts it as an edge. The low threshold searches for weak pixel intensities to identify the non-relevant pixels. It rejects the non-pixel edge which falls below the low threshold. The pixel intensities falling between the two thresholds are re-considered to be categorized into strong pixels. If the pixel being considered or processed has at least one strong pixel nearby, it considered that pixel as strong pixel intensity which leads it to an edge.

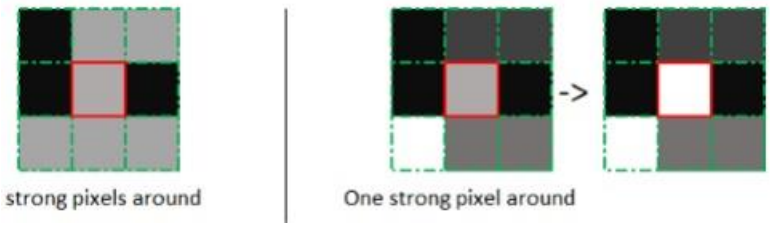


Fig. 28 Hysteresis Thresholding

Edge Map comparison :

On comparing the edge map of the “Dog” and “Gallery” images for different low and high thresholds, we see that keeping a low threshold values made the edge map too susceptible to noise and the algorithm tries to capture all the edges. For example, a value of 10 & 35 as our low and high thresholds, the image consists of extreme edges because all the edges above 35 are considered as strong pixel intensities and the algorithm keeps them as edge map. When we increase the thresholds, the noise reduces, and the edge map captures less edges. For example, a value of 230 and 250 as our low and high threshold, the image edges reduce. In the case when we keep the low and high threshold very close, the amount of edge detected depends on the value of the high thresholds. If the high threshold is less and the difference between low and high also remains less, more edges will still be detected and vice versa.

The edge map generated using Canny is better as compared to Sobel edge detector. This is because of the Non-maximum suppression. It makes the edges thin and crisp rather than the blurred and thick edges captured by the Sobel edge detector. Moreover, the Gaussian blur used in Canny also reduces the noise in the image and the edges remain smooth.

1.3. Structured Edge

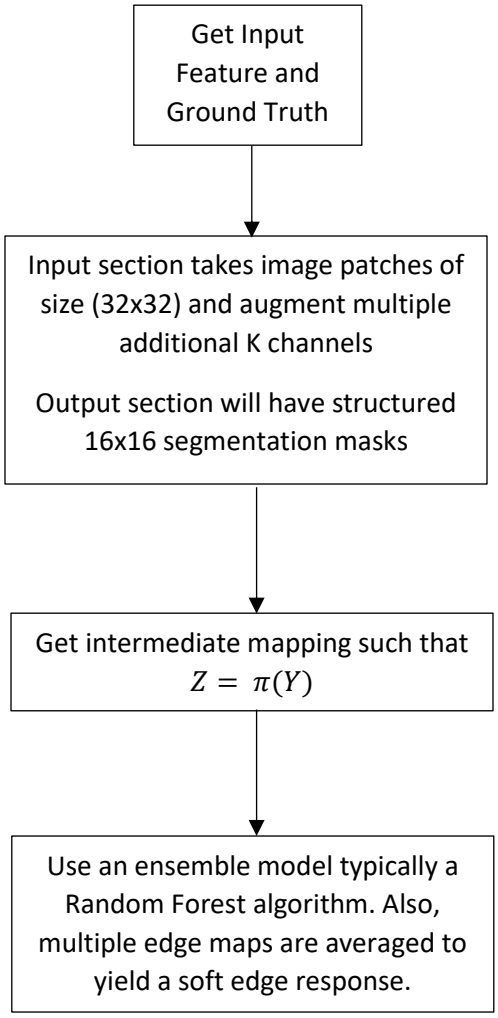
Structured Forest for Fast Edge Detection also known as Structured Edge is a data-driven approach to identify edges in an image. This approach is based on predicting local edge masks in a structured learning framework applied to random decision forests. This algorithm remains as a new, effective and advanced machine learning based technique which learns the feature or structure of the image during the model training phase and applies the learned model to unseen images to predict edges in that image. The result of this approach obtains real-time performance that is orders of magnitude faster than many competing state-of-the art approaches, while also achieving state of the art detection results on the BSDS500 segmentation dataset. [5]

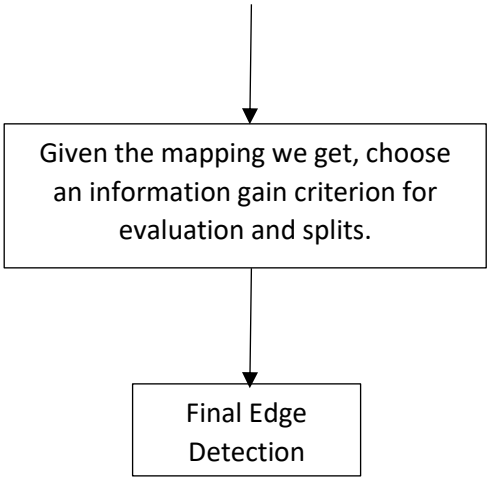
The Berkeley Segmentation Dataset 500 contains

- 500 images (200 train + 100 validation + 200 test)
- Each image was annotated by five subjects on average
- Served as evaluation of both “Segmentation” and “Contour”

Approach and Procedure

The flow chart of the algorithm is as follows:





In short, in structured edge detection, we use 32x32 image patch as data set to be used for training. The training of the image patches is done through Random Forest classifier. On the other side, the labels are 16x16 segmentation masks which later requires an intermediate mapping such that $Z = \pi(Y)$. After the intermediate mapping, the criterion for the splits depends on the information gain which we choose from several available choices. That's how we get a final trained model which predicts the labels for the classifier. [6]

Random Forest Classifier

A random forest classifier is falls under ensemble method for classification as it utilizes the fact that a random forest is made up of different decision trees and it combines the performance of all decision trees to generate a final overall decision model. Random Forest Classifier is utilized in the case of structural edge detector where it trains the input patches for edge and non-edge samples taken from the images. For splitting the trees, a performance metric such as information gain is used to split the trees to minimize the entropy.

The most important part in a random forest classifier is decision trees.

Decision trees is a simple upside-down machine learning model with its root at the top and the tree like structure splits itself based on a feature. The model selects a feature and chooses a threshold based on a greedy approach to avoid/reduce the out -sample error. When it finds maximum separation among classes/labels, it draws a decision boundary. The internal node represents a "test" on attribute , each branch represents the outcome of the test and each leaf node represents a class label. The important indicator of determining the performance is information gain and information entropy. The higher information entropy is, the higher uncertainty of the sample is. Decision trees/Random Forest are prone to overfitting which usually is linked with the depth of the tree. Pruning of the tree is a way to reduce the overfitting, which allows little misclassification to avoid overfitting. In random forest each

individual tree generates a label at its leaf. The class with the most votes becomes the model's prediction.

To reduce the overall variance in Random Forest, shuffling of features is done. Moreover, the dataset is divided into multiple subset using the principle of bootstrapping and a tree is trained on a subset of data. Random Forest is a way of averaging multiple deep decision trees, trained on different parts of the training set, with a goal of reducing variance. This comes as an expense of small increase in the bias and loss of interpretability.

The performance of such a robust algorithm can be evaluated by various metrics like precision, recall and F measure. When we compare this structured edge algorithm with the others, machine learning approach is far more advanced and gives better result than other method used.

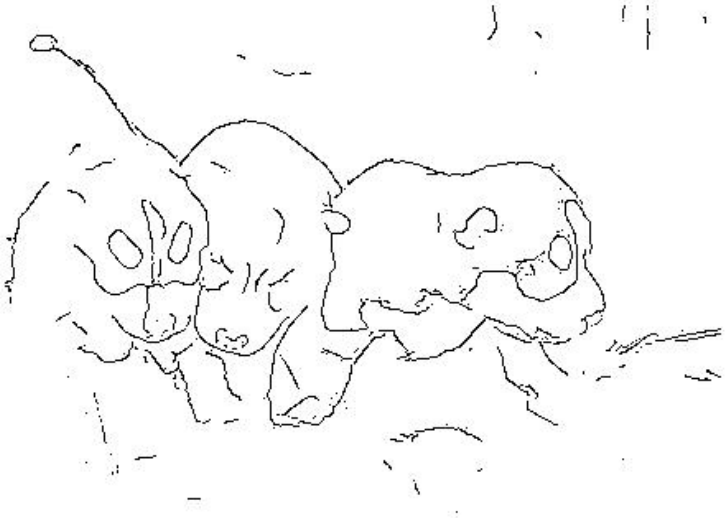
Experimental Results

The following are the outputs of Structured Edge Detector for different parameter values.

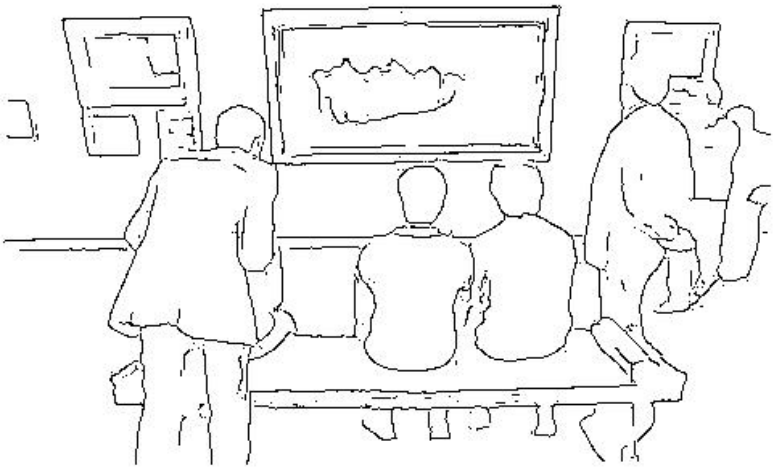
1.

```
%% set detection parameters (can set after training)
model.opts.multiscale=1;          % for top accuracy set multiscale=1
model.opts.sharpen=1;             % for top speed set sharpen=0
model.opts.nTreesEval=4;          % for top speed set nTreesEval=1
model.opts.nThreads=3;            % max number threads for evaluation
model.opts.nms=1;                 % set to true to enable nms
```

Binary edge map (with p > 0.2)



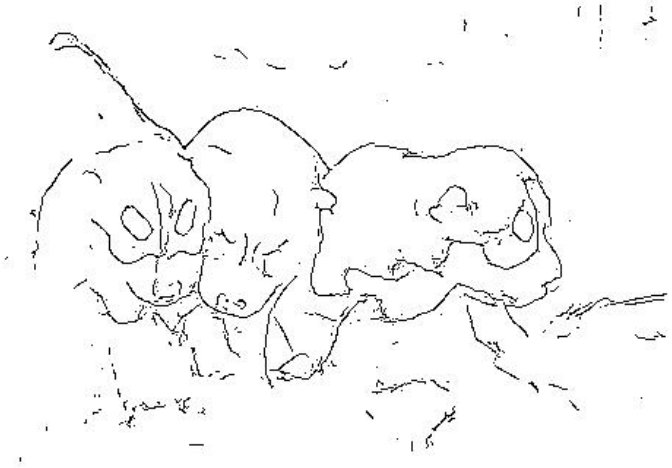
Binary edge map (with p > 0.2)



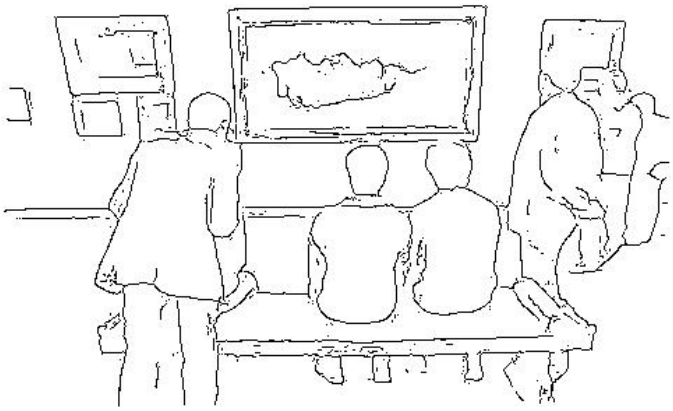
2.

```
%% set detection parameters (can set after training)
model.opts.multiscale=1;          % for top accuracy set multiscale=1
model.opts.sharpen=1;             % for top speed set sharpen=0
model.opts.nTreesEval=1;          % for top speed set nTreesEval=1
model.opts.nThreads=3;            % max number threads for evaluation
model.opts.nms=1;                 % set to true to enable nms
```

Binary edge map (with p > 0.2)



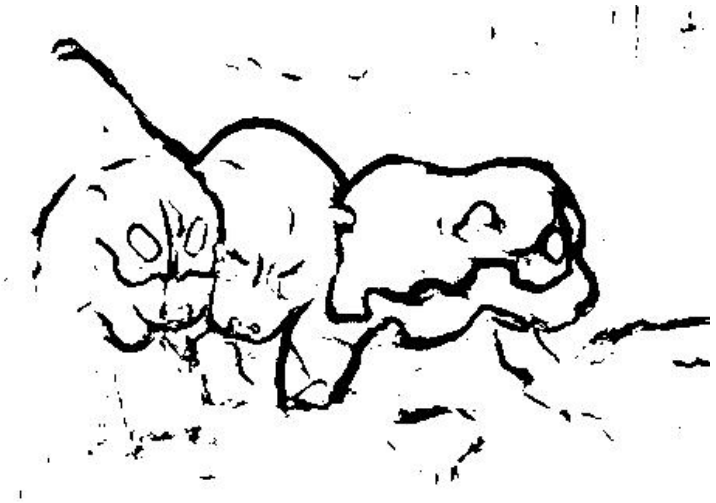
Binary edge map (with p > 0.2)



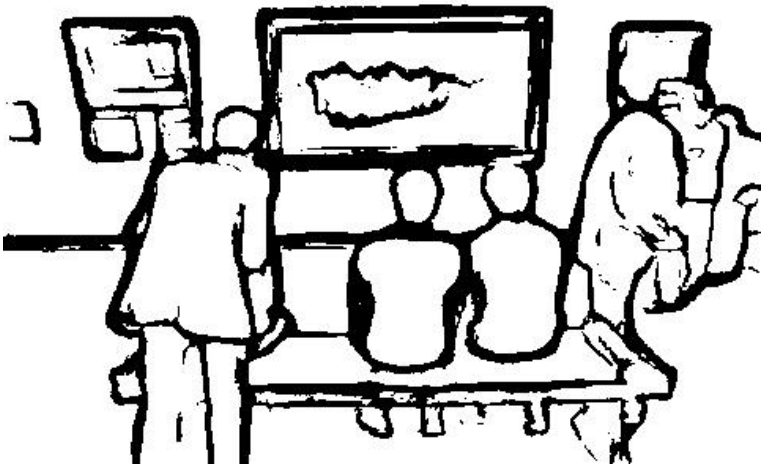
3.

```
%% set detection parameters (can set after training)
model.opts.multiscale=1;          % for top accuracy set multiscale=1
model.opts.sharpen=1;             % for top speed set sharpen=0
model.opts.nTreesEval=1;          % for top speed set nTreesEval=1
model.opts.nThreads=3;            % max number threads for evaluation
model.opts.nms=0;                 % set to true to enable nms
```

Binary edge map (with p > 0.2)



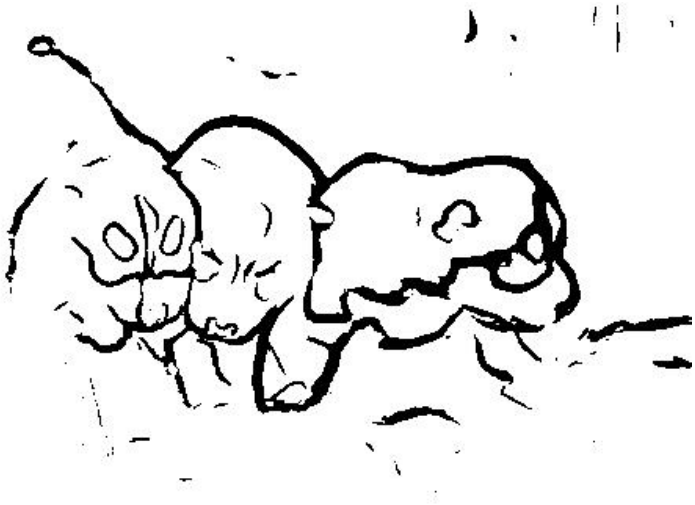
Binary edge map (with p > 0.2)



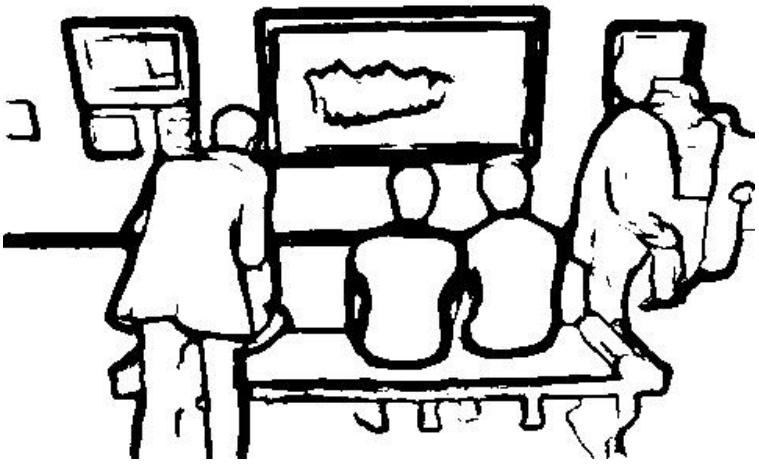
4.

```
%% set detection parameters (can set after training)
model.opts.multiscale=1;          % for top accuracy set multiscale=1
model.opts.sharpen=1;             % for top speed set sharpen=0
model.opts.nTreesEval=4;          % for top speed set nTreesEval=1
model.opts.nThreads=3;            % max number threads for evaluation
model.opts.nms=0;                 % set to true to enable nms
```

Binary edge map (with p > 0.2)



Binary edge map (with p > 0.2)



Analysis

The parameters chosen for this experiment while performing the structured edge detection is as follows.

Multiscale : This parameter is set to 1 for top level accuracy.

Sharpen : This parameter is used for setting up the top speed. This feature helps to generate a sharper and better edge maps as it is supports in aligning the masks over to the image data. It causes the sharpening of images when set to 1.

nTreesEval : This feature enables us to select the number of trees required to generate an edge in an image. The more this value is, the better the prediction is.

nThreads: This feature stands for number of threads required for evaluation. When we increase the no of threads, the process of processing a tree on different threads significantly reduces the output speed.

NMS : This parameter provides a selection of non-maximum suppression. When you enable It by making it 1, you get a crisp thin edge in the edge map. When you make it 0, you get thick and blurry edges. For best result, it should be at 1.

On comparing the performance of Canny edge detectors and Structured Edge detectors, I see that the output from Structured Edge is more refined and better. The SE detector remains successful in avoiding the high frequency noise edges such as the grass in the “dogs” image. In canny we see that it tries to generate a edges in the grass and fails to represent it. Also, the SE edge map also remains successful in detecting the edges in the object, making it a good and high performing edge detector.

1.4. Performance Evaluation

Performance Evaluation is a way to perform quantitative comparison between edge maps obtained from various edge detection techniques. The use of data driven techniques such as structural edge aims machine to generate contours of priority to human being. To satisfy that, human generated also called as ground truth is required to evaluate the quality of machine – generated edge map. Every pixel in a generated edge map belongs to either of the following four classes:

- True Positive : Edge pixels in the edge map coincide with edge pixels in the ground truth.
- True Negative : Non-edge pixels in the edge map coincide with non-edge pixels in the ground truth.
- False Positive : Edge pixels in the edge map corresponds to the non-edge pixels in the ground truth
- False Negative : Non-edge pixels in the edge map corresponds to the true edge pixels in the ground truth.

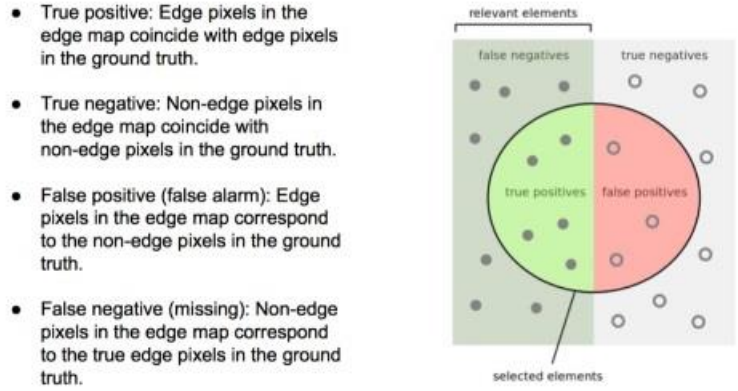


Fig. 29 Figure explaining metric for performance evaluation

We use F-measure to determine the performance of the machine-based edge detection system, which is a function of precision and recall.

$$Precision : P = \frac{\#True\ Positive}{\#True\ Positive + \#False\ Positive}$$

$$Recall : R = \frac{\#True\ Positive}{\#True\ Positive + \#False\ Negative}$$

A higher F measure implies a better edge detector.

Approach and Procedure

Experimental Results

Table for Sobel Edge Detector with different ground truth

	Precision	Recall	F – Measure
Dog 1	0.1557	0.5982	0.2643
Dog 2	0.1451	0.6248	
Dog 3	0.1513	0.6121	
Dog 4	0.1564	0.6816	
Dog 5	0.2258	0.6611	
Mean	0.1668	0.6356	

Table for Canny Edge Detector with different ground truth

	Precision	Recall	F – Measure
Dog 1	0.104	0.7435	0.1806
Dog 2	0.094	0.7501	
Dog 3	0.098	0.7389	
Dog 4	0.085	0.691	
Dog 5	0.133	0.7276	
Mean	0.103	0.728	

Table for Structured Edge Detector with different ground truth

	Precision	Recall	F – Measure
Dog 1	0.4959	0.6628	0.5575
Dog 2	0.4615	0.4355	
Dog 3	0.4908	0.4908	
Dog 4	0.4176	0.4176	
Dog 5	0.7417	0.7417	
Mean	0.5215	0.5989	

Table for Sobel Edge Detector with different ground truth

	Precision	Recall	F – Measure
Gallery 1	0.2256	0.7173	0.3572
Gallery 2	0.2054	0.7318	
Gallery 3	0.2249	0.7525	
Gallery 4	0.2121	0.7649	
Gallery 5	0.3080	0.7459	
Mean	0.2352	0.7425	

Table for Structured Edge Detector with different ground truth

	Precision	Recall	F – Measure
Gallery 1	0.6837	0.8305	0.7621
Gallery 2	0.6506	0.8430	
Gallery 3	0.6693	0.8145	
Gallery 4	0.6926	0.8115	
Gallery 5	0.8813	0.7764	
Mean	0.7155	0.8152	

Analysis

Performance of Edge Detectors :

1. Sobel Edge Detector

   Being the first of its kind in edge detection, the performance of Sobel edge detection is satisfactory. It is prone to local averaging error, noise, blurs and thick edges. Sobel edge works well when the texture of the image is not complex, and it's not recommended to use on images such as the grass image or any image which contains more details.

2. Canny Edge Detector
   Canny edge detector is an update over the Sobel Edge detector. The limitations of the Sobel edge detector is covered in the Canny edge detector. It provides crisp and smooth

edges. Moreover, it uses Gaussian Blur, Non-Maximum Suppression and Hysteresis thresholding to improve the overall performance of the edge detection.

3. Structured Edge
   Structured Edge is a data driven edge detector which uses random forest model to train patches of images. The performance of Structured Edge detector is remarkably well when compared to Sobel and Canny Edge detectors. But, edge detection in structured edge is pretty slow and takes time. Moreover, the structured edge depends on the different types of parameters and works well if tuned properly.

F – Measure :

The F- measure is image dependent. The F – measure for Gallery is high than Dog and thus can be said that It is easy for Gallery to get a higher F – score than Dog. The reason behind this comes from the fact that the dog image has high frequency components such as grass which leads to high noise. The edge in Dog image essentially gets too strong near the grass. But, in case of the Gallery image, the texture is smooth and does not contain much high frequency components.

2. Digital Half-toning

Halftoning is a technique that simulates continuous-toned pixel image into a two-toned pixel image by use of dots. It renders the image in a way that it can refer specifically to the image that is produced by this process. We see that grayscale tone imagery consists of a continuous tone where the pixel intensities vary from 0 to 255, halftoning processes the image that makes the image use only two tones. This method is essentially converting the image into two tones making the grayscale image into either white or black.

Half-toning was introduced to utilize the fact that human eye fails to recognize or differentiate two dots very close to each other. Thus, the high-density dots in the local cells can be rendered as a gray scale pixel.

The importance of using half-toning lies in the fact that every grayscale image contains 8 bits which makes it a 256-tone image and RGB on the other end has 24 bits which makes it a 16.7 million colors. But most printing devices are not able to reproduce different

shadows of gray or so many colors. So, the original image is transformed into an image which contains only white(no-ink) and black(one dot of ink).

The three half-toning techniques covered in this assignment is :

- Dithering
    - Fixed Thresholding
    - Random Thresholding
    - Dithering Matrix
- Error Diffusion
- Color halftoning with error diffusion
    - Separable Error Diffusion
    - MBVQ-based Error Diffusion

2.1. Dithering

This method focuses on using either fixed/constant thresholding , random thresholding and dithering matrix for transforming the grayscale image into binary image consisting of either black or white pixels. The fixed/constant thresholding methods to binarize the image checks for a threshold value. If the pixel value of the original input image is less than the fixed/constant threshold, the output image is a black dot. In case of random thresholding, the original image pixel when found less than a random value of threshold generated using rand() function, gives a black dot. In case of dithering matrix, a Bayer matrix consisting of Index matrix is convolved over the original image and the output value generated using the process. The Bayer matrix are generated recursively and vary in size. Common examples are 4x4, 8x8, 16x6, 32x32 etc.

Approach and Procedure

1. Fixed Thresholding

   This thresholding utilizes a 127-pixel intensity as the threshold to compare and transform the original image pixel intensities into binary pixel intensities. Each pixel value is compared with the value 127 and the output image (binarized) is assigned a black dot if the pixel value is less than the threshold and a white dot if the pixel value is greater than the threshold. The equation for this thresholding can be given as :

$$Output\ (i,j) = \begin{cases} 0 & 0 \le input(i,j) < threshold \\ 255 & threshold \le input(i,j) < 256 \end{cases}$$

   Algorithm Implemented :

   - Read the input image

- Initialize a threshold and compare every pixel of the input image with the threshold. If the input image pixel value is less than threshold, assign the output image 0.
- If input pixel greater than threshold, map 255.
- Write the file onto a .raw file

2. Random Thresholding

   This thresholding incorporates randomness in the pixel intensities by selecting the threshold randomly. It tries to break the monotones in the result from the fixed thresholding. The threshold is selected from a distribution and is used to compare with the original image intensity values. If greater that we map it to 255 otherwise we map it to zero.

   Algorithm Implemented :

   - Read the input image
   - For each pixel, generate a random number in the range 0-255, so called $rand(i,j)$
   - Compare the pixel value with $rand(i,j)$. If found greater, map it to 255, otherwise map it to 0.
   - Write the file onto a .raw file

   The equation can be defined as

$$Oputput(i,j) = \begin{cases} 0 & 0 \leq input(i,j) < rand(i,j) \\ 255 & rand(i,j) \leq input(i,j) < 256 \end{cases}$$

3. Dithering Matrix

   To avoid exposed or noisy image outputs obtained from the fixed and random thresholding, the dithering matrix is a subtle way to binarize the image. This is also called as ordered dithering which works by moving a Bayer matrix of given size over to the image. The original pixel intensities are compared with the pixel intensities in the threshold matrix which is essentially generated from the Bayer Filter matrix.

   The index matrix is a 2x2 size which calculates the likelihood of a pixel to be turned on. Depending upon the values, certain pixels are turned on and off when this filter is convolved over the original input image.

   Index Matrix is given by : $I_2(i,j) = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$

   And the Bayer matrix is calculated recursively using the formulae given below:

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

   The threshold matrix generated for an input gray image with normalized pixel values is given by :

$$threshold\ matrix\ \ T(x,y) = \frac{I(x,y) + 0.5}{N^2} * 255$$

The final thresholding of every grayscale image pixel is given below:

$$output(i,j) = \begin{cases} 0 & 0 \leq input(i,j) \leq T(i\ mod\ N, j\ mod\ N) \\ 255 & T(i\ mod\ N, j\ mod\ N) < input(i,j) < 256 \end{cases}$$

```
ALGORITHM IMPLEMENTED
1.  Read the inout image, height, width, BytePerPixel , index_value
using command line
2.  For fixed thresholding, consider a fixed threshold value eg. 127
and compare it with the input pixels. If found less than 127, map it
to 0, else map it to 255.
3.  For random thresholding, generate a random number using
rand()%256. Compare the value of the random number in the range of 0
to 255 with the input image. If found greater than random value,
map it to 255. Else map it to 0.
4.  For dithering, generate Bayer matrix using the formalae provided
in the lecture. Interate through the loop until count > 0.
Recursively add elements in the Bayer filter matrix based on index
provided.
This will generate three matrix I2, I8, I32.
5.  Compute the threshold matrix using the recursively generated
bayer matrix and perform thresholding to generate an output file.
5.  Write the individual output value onto a .raw file.
```

Experimental Results

The results of fixed thresholding, random thresholding and dithering is as follows :

Fig. 30 Fixed Thresholding

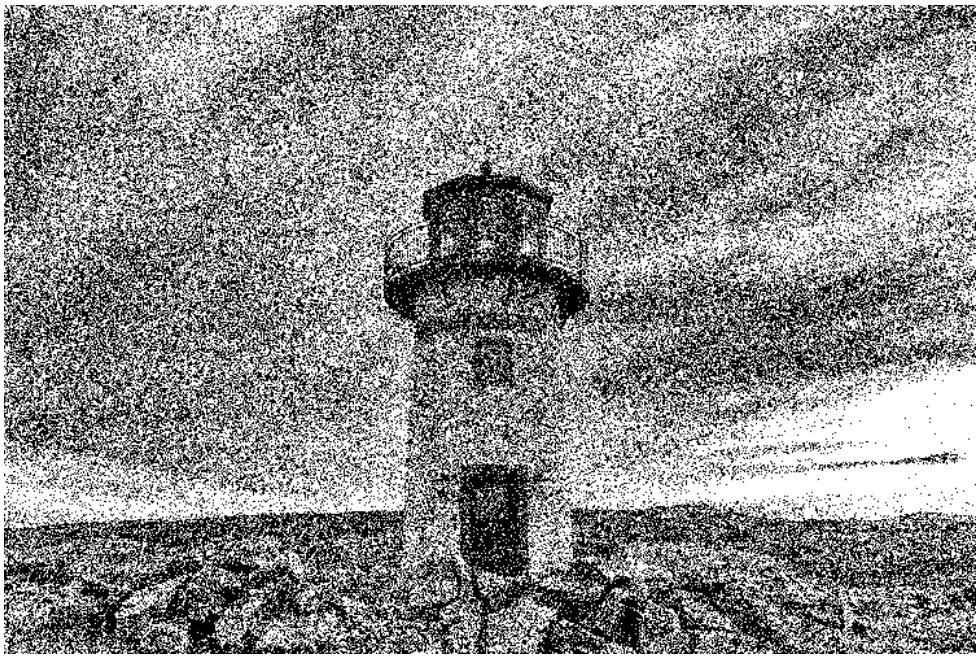


Fig. 31 Random Thresholding

Fig. 32 I2 Dithering Matrix

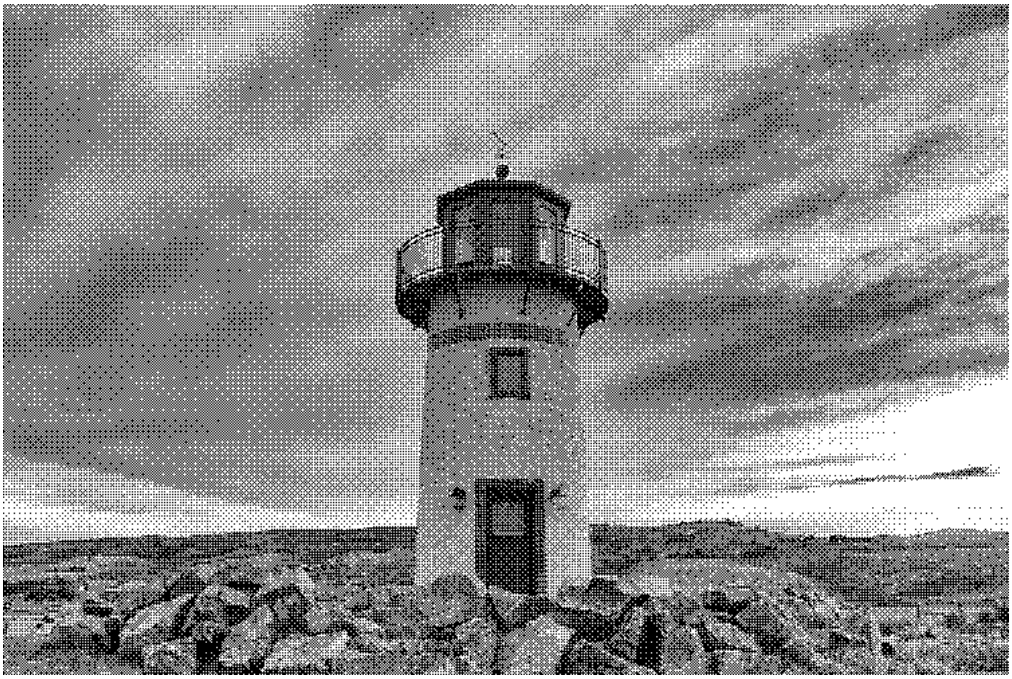


Fig. 33 I8 Dithering Matrix

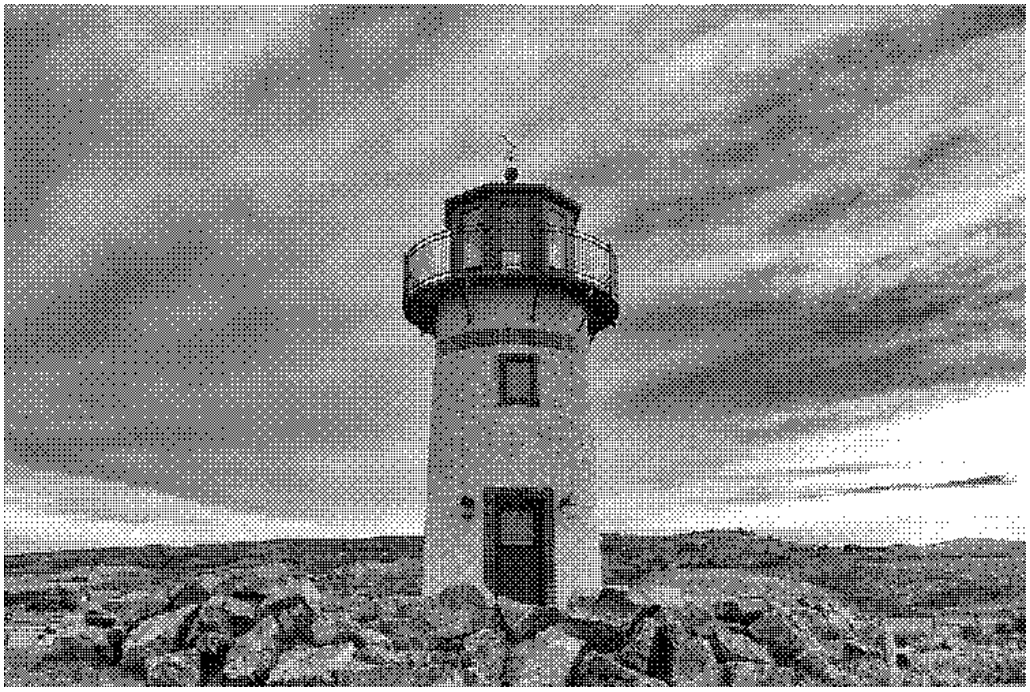


Fig. 34 I32 Dithering Matrix

Discussion

In this part of the question, we work on the half-tone images. The various half-toning methods implemented are the images obtained from fixed threshold, random threshold and dithering matrix. The following conclusions are my observations on the output as follows:

- On comparing the output of all the three methods, dithering matrix generates better half-toned images than the fixed and random.
- Random thresholding generates the worst half-toned image out of the three. The image is a distorted image which results from a randomized threshold generated from any standard distribution.
- The fixed thresholding method exposes the image due to fact that it gives only two dots i.e. either a black or white one.
- On comparing the different Bayer matrix-based dithering, we see that the result of I8 threshold matrix is better than the I2 threshold matrix. However, it gets difficult to decide any difference between I8 and I32. This can be implied that I8 should remain an ideal choice to perform dithering as it saves computation and memory cost.
- On zooming the outputs from the dithering procedure, we see that numerous amounts of dots scattered over the image, makes it distorted.

2.2. Error Diffusion

Error diffusion is another well known half-toning algorithm which focuses on the diffusion of the error when calculated at a particular pixel to its neighboring pixels. The diffusion of error to its neighboring pixel is achieved through three different diffusion matrices. The main application of error diffusion method is to convert continuous-toned image to a binary, dual tone image. The ability of diffusing the error to a new pixel which is not processed, produces better local intensities, thus leading to a better enhanced binary image. Moreover, the filter coefficients of the error diffusion matrix sums to one making the local value of output image almost equal to the average value of input image.

The three different error diffusion method :

- Floyd – Steinberg error diffusion method with 3x3 kernel
- Jarvis, Judice and Nunke(JJN) error diffusion method with a 5x5 kernel
- Error diffusion by Stucki using a 5x5 kernel

The different error diffusion method listed above also uses two different kind of scanning procedure. It's called serpentine scanning and razor scanning.

Approach and Procedure

The error diffusion algorithm consists of three steps :

1. Thresholding
   Each pixel undergoes fixed thresholding with a constant value of around 127. This image is basically a binarized form of the original image.

$$b(i,j) = \begin{cases} 255 & if\ f'(i,j) > T \\ 0 & otherwise \end{cases}$$

   Here, $b(i,j)$ is the binarized image, $f'(i,j)$ is the original image, T stands for a threshold and i & j are the pixel values of the image.

2. Error Calculation
   This step calculates the error between the original image pixel and the binarized image pixel value. The error in the error diffusion indicates the change the pixel needs to convert the multi-level image into a binary form.

$$e(i,j) = f'(i,j) - b(i,j)$$

   Here, $e(i,j)$ is the error, $f'(i,j)$ is the original image, $b(i,j)$ is the binarized image and i & j are the pixel values of the image.

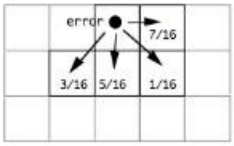
3. Error Diffusion
   The error is diffused to future pixels by using several filters and error diffusion matrix called as Floyd Steinberg, Jarvis, Judice and Ninke(JJN) and Stucki.

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

Here,

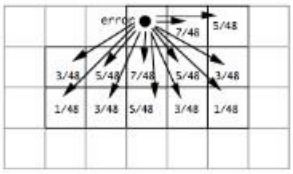
- **Floyd-Steinberg**

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



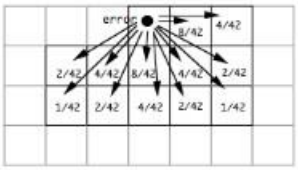
- **JJN**

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



- **Stucki**

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



Serpentine Scanning vs Razer Scanning method

The different scanning methods makes this algorithm different from other half-toning methods. The serpentine scanning scans the image from left to tight in even rows and from right to left on odd rows. Whereas the razer scanning method goes from left to right in all the rows.

ALGORITHM IMPLEMENTED

1. Read the input image, height, width, BytePerPixel using command line.
2. Perform boundary extension using zero padding on the input image.
3. Calculate the error while performing FS error diffusion. This is done in nested loop. Convolve the FS matrix over the zero padded image and diffuse the error itself in nested loop to its neighbouring pixels.
4. Calculate the error while performing JJN error diffusion. This is done in nested loop. Convolve the JJN matrix over the zero padded image and diffuse the error itself in nested loop to its neighbouring pixels.
5. Calculate the error while performing S error diffusion. This is done in nested loop. Convolve the S matrix over the zero padded image and diffused the error itself in nested loop to its neighbouring pixels.
6. Write the output images onto .raw file

Experimental Results

The results of error diffusion using three matrices is as follows :

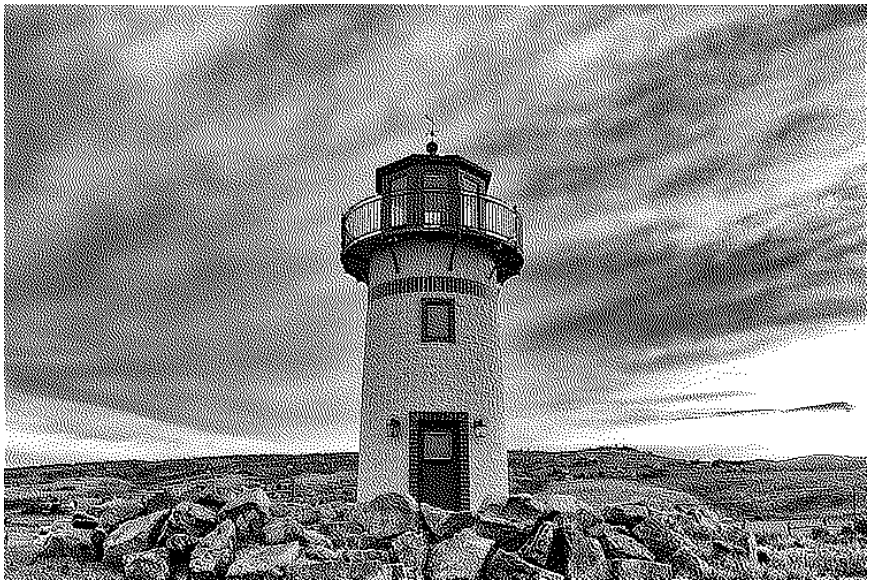


Fig. 35 Jarvis, Judice and Ninke (JJN) error diffusion output

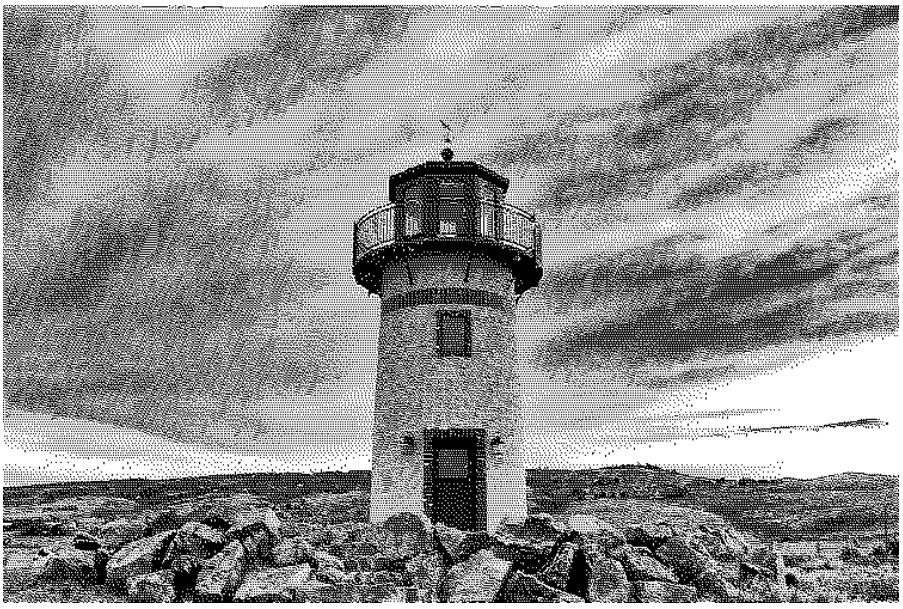


Fig. 36 Floyd – Steinberg's error diffusion output

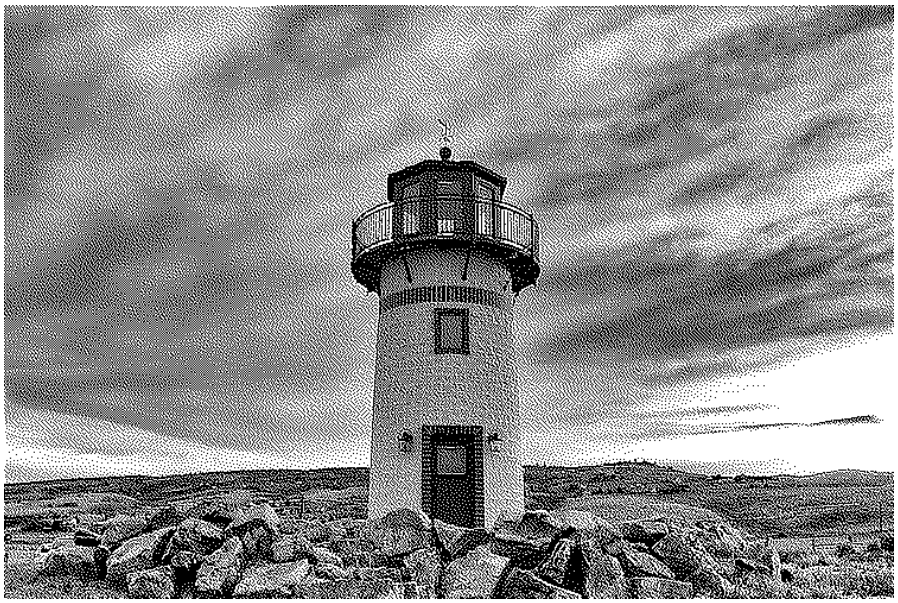


Fig. 37 Stucki error diffusion output

Discussion

In this part of the question we compare the error diffusion methods using different error diffusion matrices and also compare the results with the dithering method performed in the earlier part of the question.

On comparing the half-toning output generated by the three different error diffusion methods, we see that the Floyd – Steinberg's error diffusion method utilizes a kernel of 3x3, thus diffusing the error to just one block to its neighbor. This method is fast and efficient as it saves time in diffusing error. On the other hand, the JJN or Stucki method utilizes a 5x5 filter, making it diffuse the error more towards its neighbors. The output of JJN and Stucki remains almost similar and better when compared to the output from Floyd – Steinberg.

When compared with the half-toning methods such as dithering, the output from the error diffusion is better, crisp and soother than the dithering method. Moreover, the performance of the dithering half-toning method is slower than the performance of the error diffusion methods. Out of all the half-toning methods used in this assignment, the performance of JJN or Stucki is the best and recommended among other methods.

There are other several method also used for half-toning such as Sierra, Atkinson, Burkes etc. which are designed to perform well than the methods discussed here. These methods are fast, more efficient and generate an image which resembles to a continuous-toned image. All the above listed method works on different principles and thus also vary in their output depending upon the type of image. Also, Atkinson is a extended version of error diffusion.

2.3. Color Halftoning with Error Diffusion

Like halftoning in grayscale images, color halftoning is also possible which generates dots of 8 different colors. This is also an effective way to represent the colored images more efficiently, thus preserving the maximum information from the image. This generates the image which is useful in printing colored images. This part focuses on two different color halftoning algorithms or methods. These are Separable error diffusion and MBVQ based error diffusion.

We know that a colored image can be represented by a 24-bit pixel value. The no of combination in a way a color can be distributed goes in millions. In color halftoning, we reduce the combinations to just 8 colors. The 8 color are: Blue, Magenta, Green, Cyan, White, Black, Yellow, Green.

The separable error diffusion uses Floyd Steinberg error diffusion with serpentine scanning for color halftoning. It converts the RGB to CMY color space and applies the error diffusion matrix to quantize each channel separately. This is similar to the procedure used in error diffusion method for halftoning grayscale images, just that the three different channels have to separated, processed parallelly and combine them into an RGB form. This method requires the color space to be changed to CMY.

MBVQ which stands for Minimal Brightness Variation Quadruples overcomes the shortcoming of the separable error diffusion method. The method works by separating the RGB color space into minimum brightness variation quadrants. With this we can render every input color using one of the six complementary quadruples: RGBK, WCGC, RGMY, RGBM, or CMGB. We can render any color using the 4 colors pertaining to a quadrant than 8 colors. This saves resources with different color requiring different quadrants.

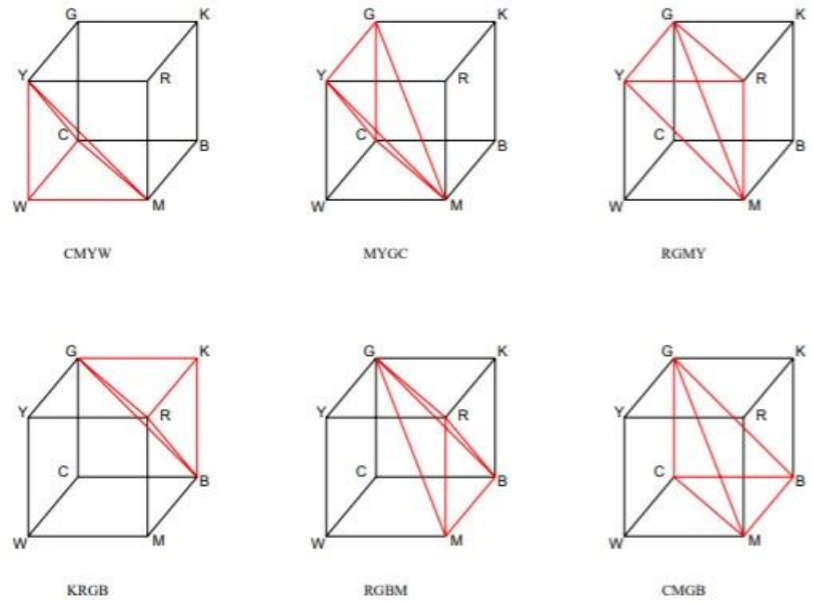


Fig. 38 Six Tetrahedral Volumes

Approach and Procedure

Separable Error Diffusion

This method uses two steps:

1. RGB to CMY
   We have the RGB channel to CMY channel because for printers, this is done using CMY space. The conversion is done through a simple formula given as

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

2. Perform error diffusion
   The three channels generated by the RGB to CMY color space undergoes FS error diffusion method using serpentine scanning. All three are individually treated and then combined into a CMY image. Later this is converted back to RGB using the formulae given above.

```
ALGORITHM IMPLEMENTED
1.  Read the inout image, height, width, BytePerPixel using command
line
2.  Call the function seperateolor() which seperates the individual
RGB channel and converts into CMY space by subtracting 255.
3.  Diffusion of error using the FS serpentine method on each
individual channels.
4.  Merge the individual CMY channels into one CMY image and then
convert it into RGB space.
5.  Write the output onto .raw file.
```

MBVQ based Error Diffusion

This method utilizes 3 steps,

1. Determine MBVQ
   MBVQ is basically a set of 4 points in one of the six regions in the cube. Every RGB pixel lies in one of the regions and inside each region we have minimum brightness change. The six different regions the MBVQ can take is “CMYW”, “MYGC”, “RGMY”, “KRGB”, “RGBM”, “CMGB”.

2. Find nearest vertex
   This step searches for the nearest pixel point in that region were the RGB pixel is. The vertex is essentially the color at the vertex which is found to be near to RGB pixel.

3. Calculate Error
   The error is essentially the error in terms of pixel which we get from the difference in the RGB point and the color at the vertex. The error found is diffused to the neighboring pixels in the next step.

4. Error Diffusion
   This step uses Floyd- Steinberg kernel to diffuse the error to the next pixel values. This step uses serpentine scanning method for efficient scanning of the image to get an efficient and high performing halftoned image.

```
ALGORITHM IMPLEMENTED
1.  Read the inout image, height, width, BytePerPixel using command
line
2.  Calculate the MBVQ from a function, which returns the quadrant
of the cube based on the pixel value of RGB
3.  Based on the MBVQ, calculate the closest vertex based on the
function provided. The function value returns a color which is
closest to the original pixel value
4.  After getting the color of the pixel value, determine the pixel
value based on another function colorRepresentation().
5.  Use the FS diffusion method to diffuse the error onto the
neighbouring pixels after calculating the error based on the
difference between old pixel and new pixel.
6.  Write the output file onto .raw format
```

Analysis

The two methods discussed above efficiently performs halftoning of an RGB. It maps the entire 3 channel color combination into either 8 color dots, or 4 color dots based on the algorithm used. Separable error diffusion works on finding closest vertex out of the 8 points in the cube representing a color dot and MBVQ error diffusion works by finding the closest vertex out of the 4 points in a quadrant.

Shortcoming of Separable Error Diffusion :

- Due to fact that this method separates the three channels and performs error diffusion in each channel separately, the output image generated after merging the channels again generates color artifacts and poor color rendering.
- This method works on perform error diffusion on RGB channels respectively, thus exploiting the inter color correlation.
- The output image is noisy and sharp than the original input image.

We overcome this artifact by using MBVQ based error diffusion method.

Experimental Results

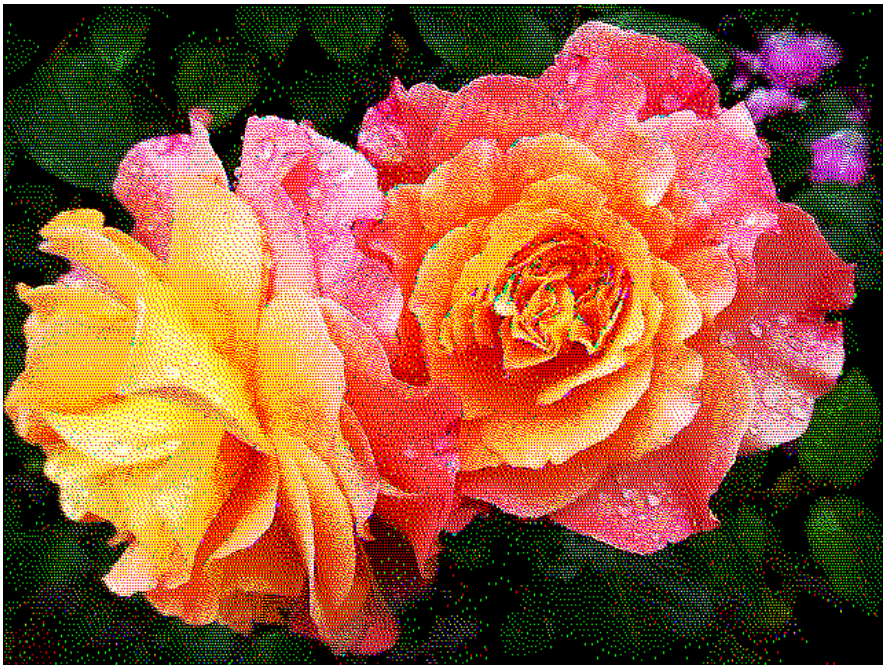


Fig. 38 Color Halftoning using Error Diffusion Method

3. References

[1] https://en.wikipedia.org/wiki/Edge_detection

[2] https://medium.com/datadriveninvestor/understanding-edge-detection-sobel-operator-2aada303b900

[3] https://en.wikipedia.org/wiki/Canny_edge_detector

[4] https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

[5] EE569 Discussion slides covering Structured Edge.

[6] http://graphics.cs.cmu.edu/courses/16-824/2016_spring/slides/seg_1.pdf

[7] https://www.imaging.org/site/PDFS/Papers/1998/PICS-0-43/666.pdf

[8] https://en.wikipedia.org/wiki/Halftone