EE569 Digital Image Processing

**HOMEWORK#2 Report**

**Issued: 2/8/2021 Due: 11:59PM, 2/23/2021**

**Name: Siyu Li ID:2455870216**

**E-mail: lisiyu@usc.edu**

**Problem 1: Edge Detection (50 %)**

**(a) Sobel Edge Detector (10%)**

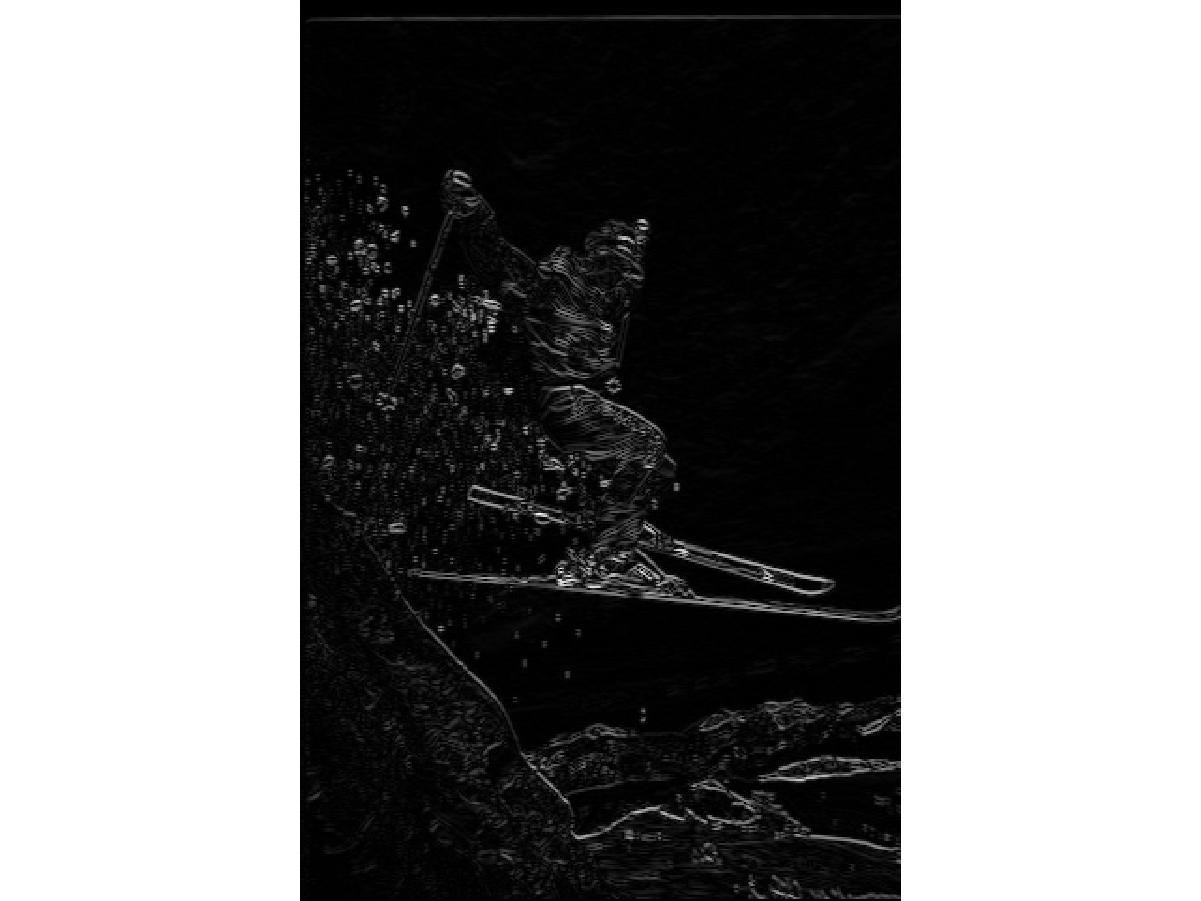
**I. Approach and Procedures**

Sobel edge detector is a very applicable method on computer for its low computational complexity. Its principle is to detect or calculate the change between neighbor pixels which is gradient. Here it is implemented by calculating the gradient map from X direction and Y direction respectively. To deal with negative value on the X and Y direction gradient map, we should take their absolute values before doing normalization.

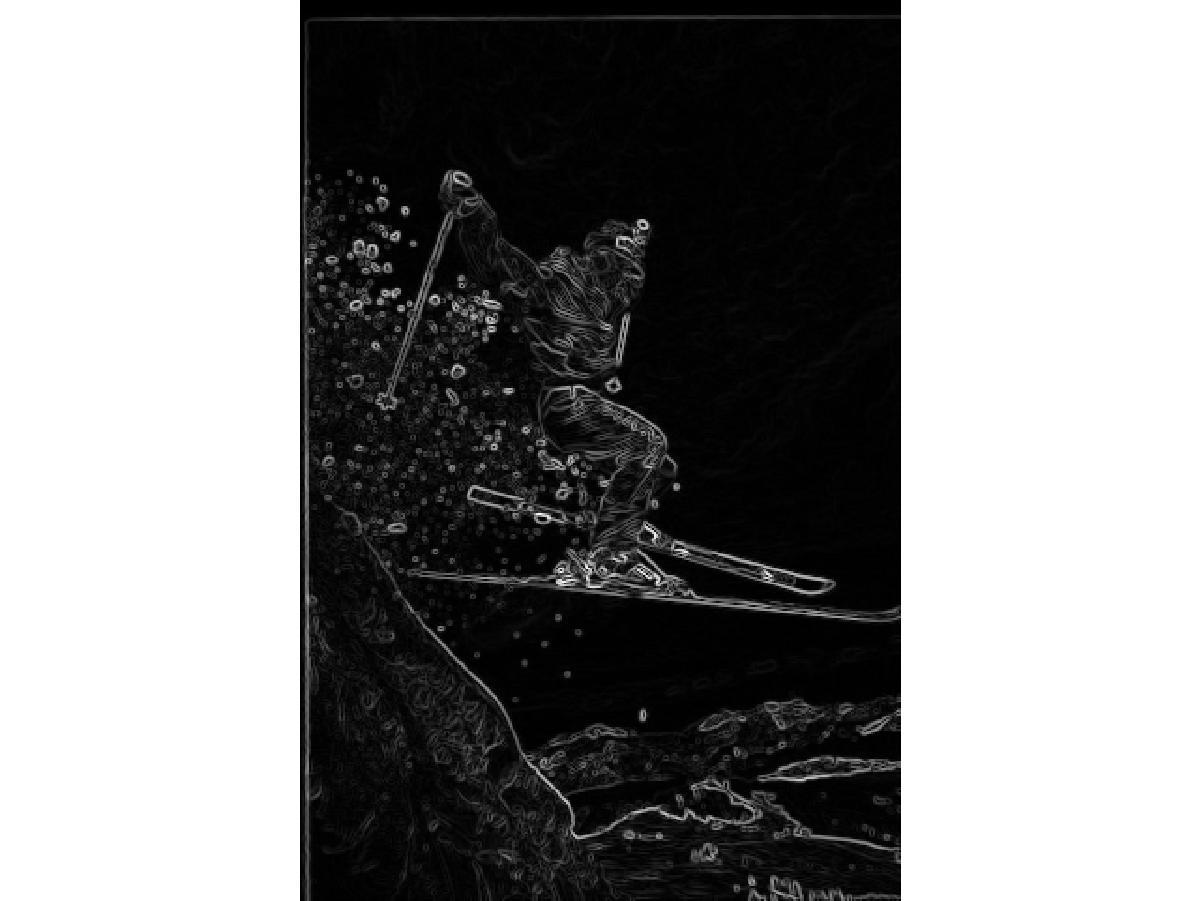
For example, -4 and 4 original gradient values mean absolute the same rate of pixel intensity change with different direction only. But if they are not taken absolute value and then normalized, the final normalized gradient value for -4 and 4 in original gradient map will be different, which can not reflect how steep the pixel intensity values change. And to calculate the final magnitude gradient map which takes both X direction gradient map and Y direction gradient map into account, we use to get the final Magnitude map. Here Xgradient and Ygradient map are maps which have already taken absolute value but have not been normalized. Then normalize the final Magnitude map to 0-255.

At last, to get the final edge map, we should apply a threshold function to the normalized final Magnitude map to get a binary image which contains only two-pixel values 0 and 255. The threshold function works by setting all the value below the threshold 0 and the rest 255. The threshold here is decided by percentage of arrangement. By using 90% threshold for example, we first arrange all pixels based on intensity value, and set the first 90% pixel 0 and the rest 255.

**II. Experimental Results**

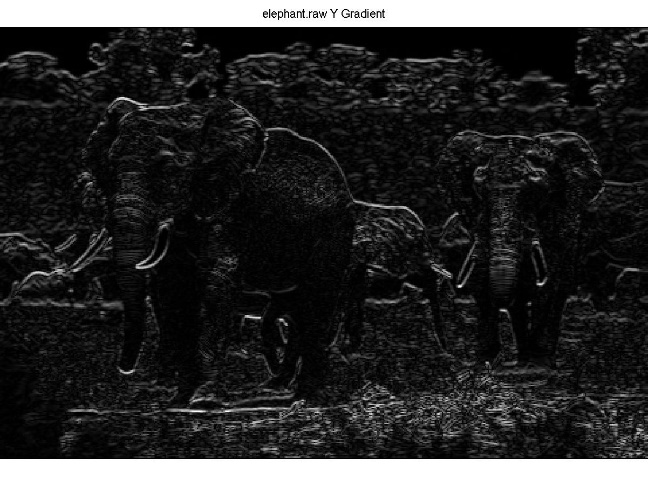
****

**(a) Gradient on X direction (b) Gradient on Y direction**

****

**(c) Final Gradient Magnitude Map**

**Fig 1.1 Result of Sobel Edge Detector on ski\_person.raw**

****

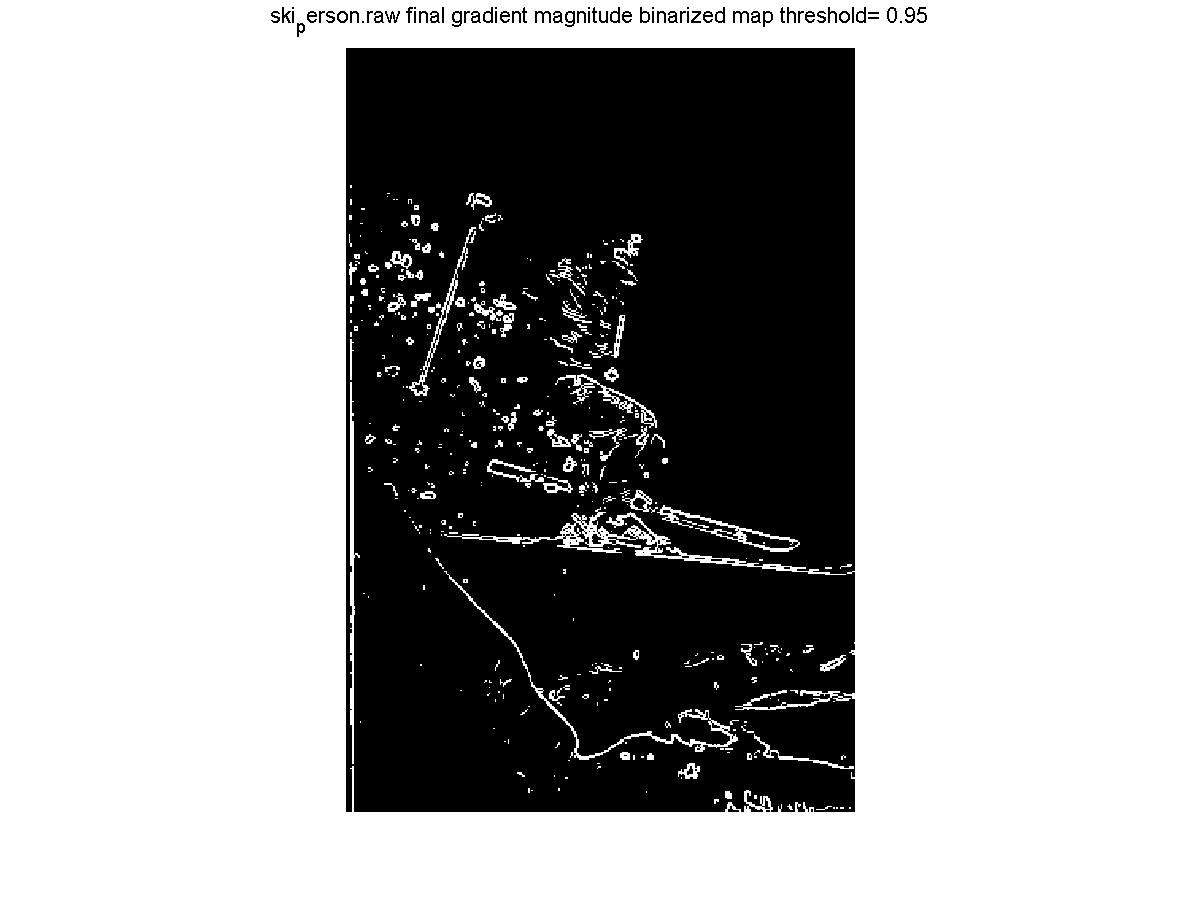
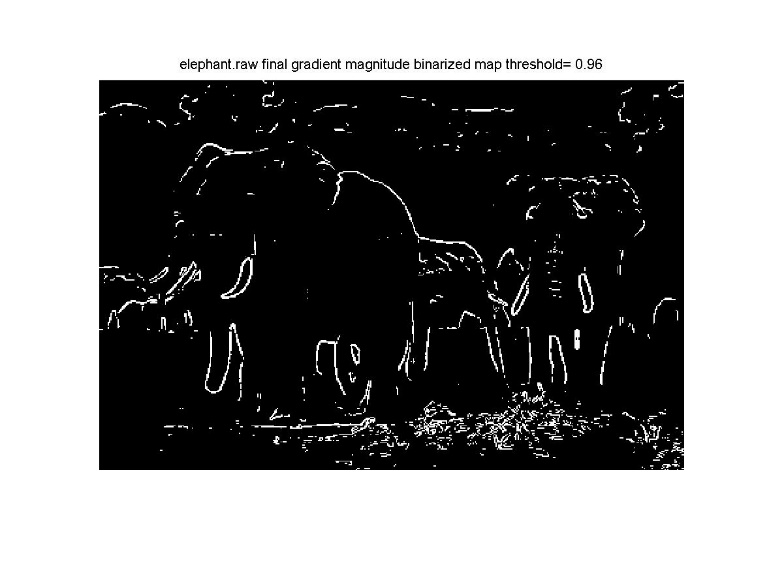
**(a) Gradient on X direction (b) Gradient on Y direction**

****

**(c) Final Gradient Magnitude Map**

**Fig 1.2 Result of Sobel Edge Detector on elephant.raw**

In order to get the best final binary edge map for the two pictures, we chose threshold=0.96 on result of elephant and threshold=0.95 on result of ski\_person.

****

**(a) Final Edge Map of elephant.raw threshold=0.96 (b) Final Edge Map of ski\_person.raw threshold=0.95**

**Fig 1.3 Final Edge Map**

Other binary final edge map result with different thresholds is also available in the original code, but due to the lack of space we do not post them here.

**(b) Canny Edge Detector (10%)**

**I. Approach and Procedures**

Canny edge detector is a more advanced method using maximum suppression to delete or suppress unnecessary edges and improve performance of detection. This part is implemented by edge function by traversing different parameters in Matlab according to homework requirements.

**(1):** Explain Non-maximum suppression in Canny edge detector in your own words:

We usually assume that edge is composed of very few pixels where the pixel value change abruptly, which means edge should be composed of only very thin curves or lines. However, the edge calculated by most of detection kernels is always very wide because of the built-in blurring property of kernels. Non-maximum suppression is a method which can put non-maximum pixel value to zero to make edges thinner. In canny detection process, only the largest point in the gradient direction can be remained and other points will be set 0. The gradient direction is calculated by .

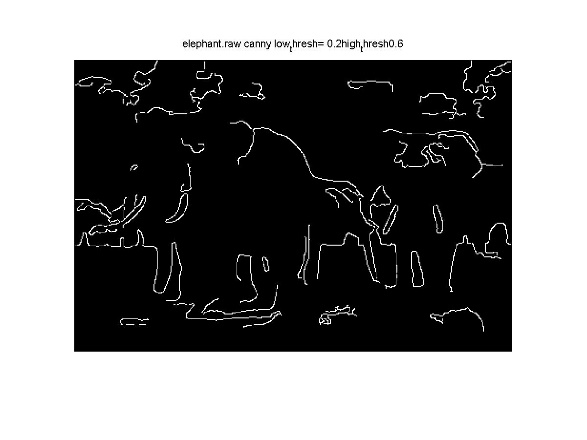
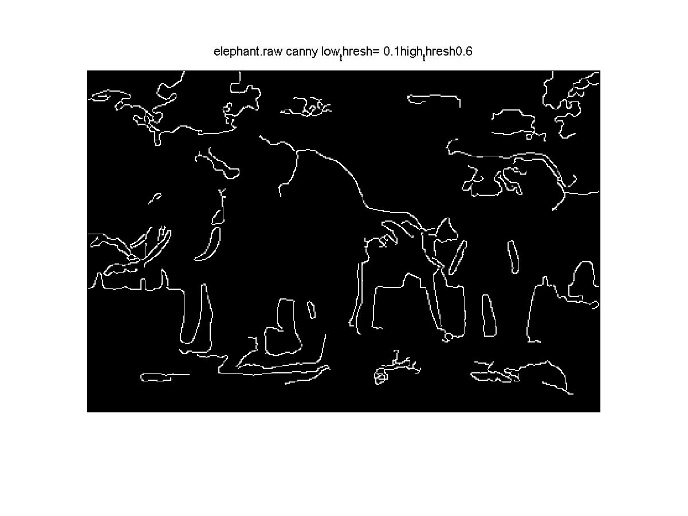
**(2):** How are high and low threshold values used in Canny edge detector?

After the process above, the points with pixel value above high threshold are considered as strong edges while points below low threshold are not considered as edges. And the rest points are considered as weak edges which play the role of connecting strong edges. Strong edges will be set to value 255 on final edge map and weak edges will still exist if their neighbors are strong edges otherwise they will also be set to 0 as is the same with pixels below the low threshold.

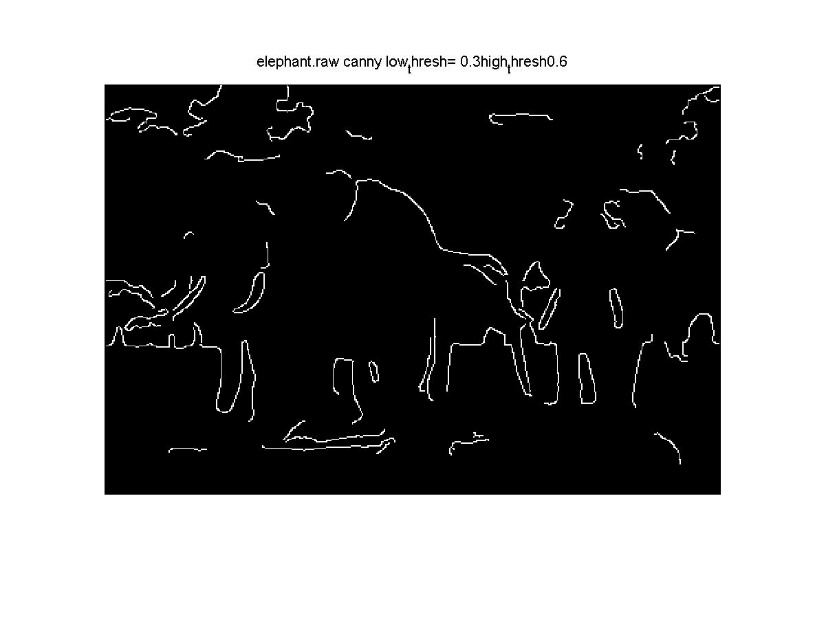
**II. Experimental Results**

**(3):** Generate edge maps by trying different low and high thresholds and discuss your results.

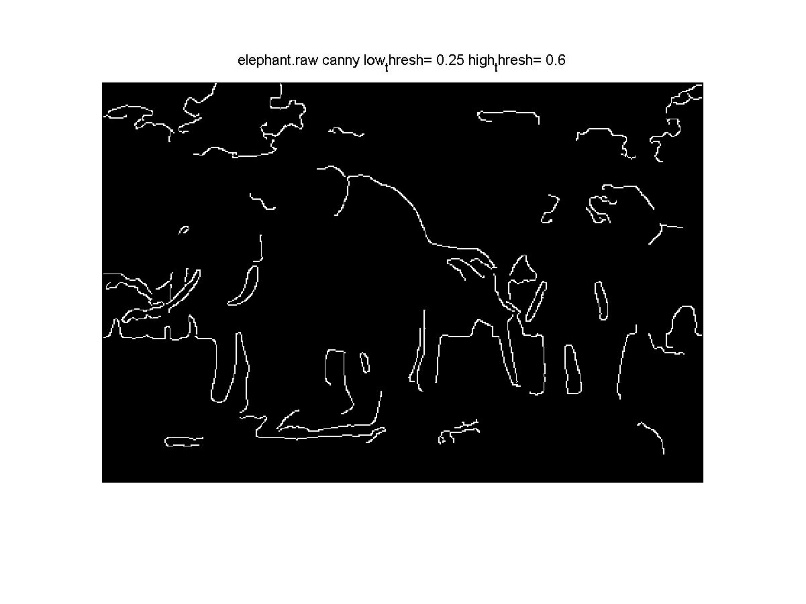
High threshold is applied to determine the boundary of major objects. If the high threshold is too high, the final edge map will lose lots of contour of major object. And when it is too low, the noise and texture information will be mistakenly regarded as strong edge but they are not contour or edge at all. Low threshold here contributes to the continuity of edges, it connects relatively scatter strong edges. If it is set too low, it will introduce lots of noise and texture and on the other hand if it is set too high, there will be lots of discontinuities on the edge map which have a negative influence on the description of major object by showing the complete edge.



(a) low threshold=0.1 high threshold=0.6 (b) low threshold=0,2 high threshold=0.6



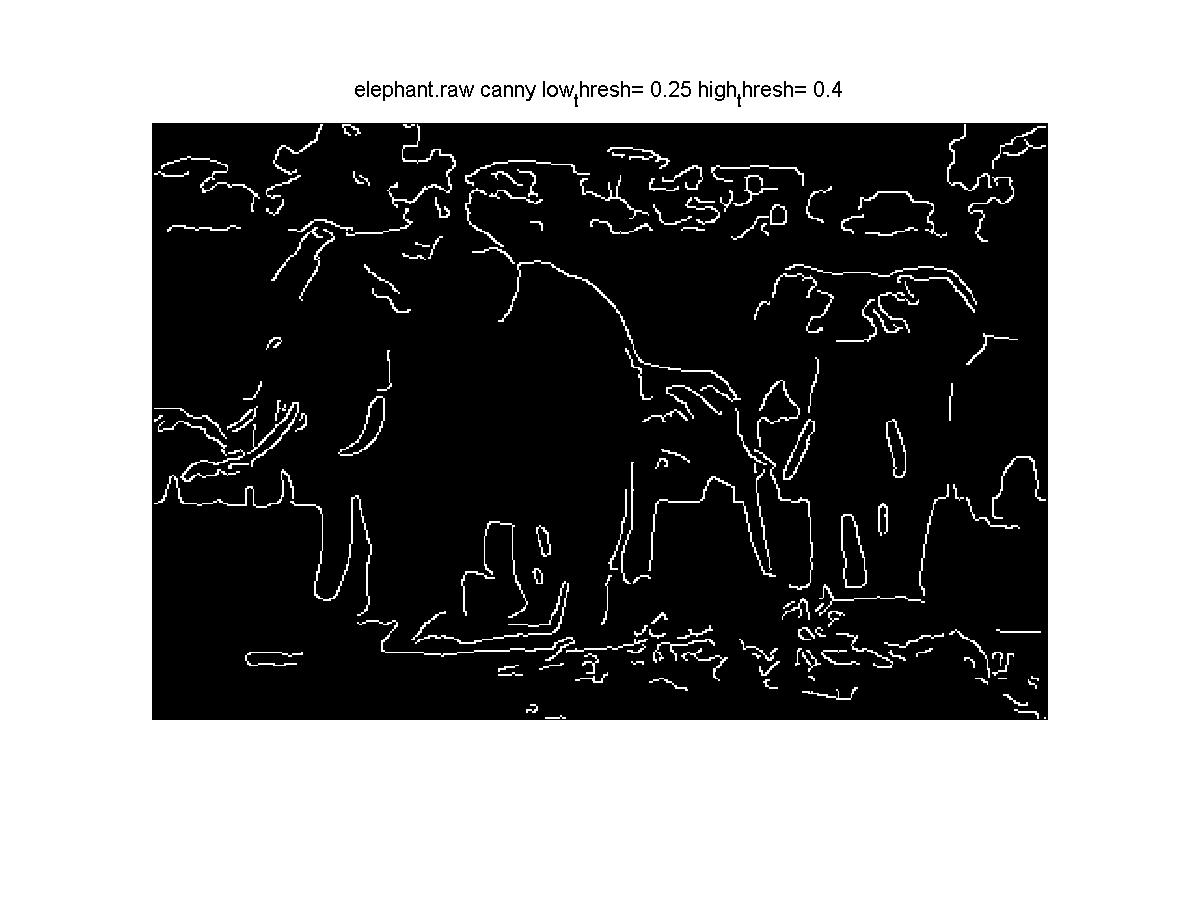
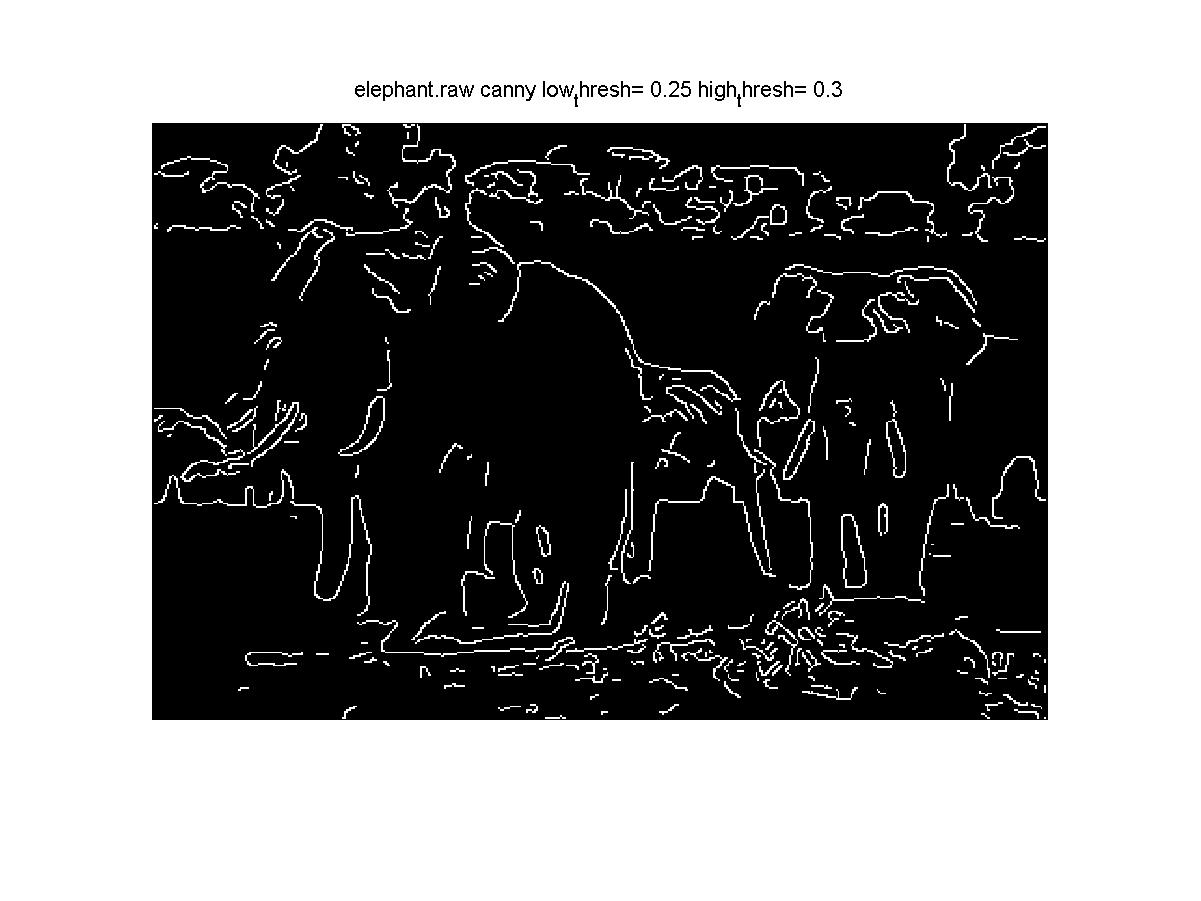
(c) low threshold=0.3 high threshold=0.6 (d) low threshold=0.15 high threshold=0.6

****

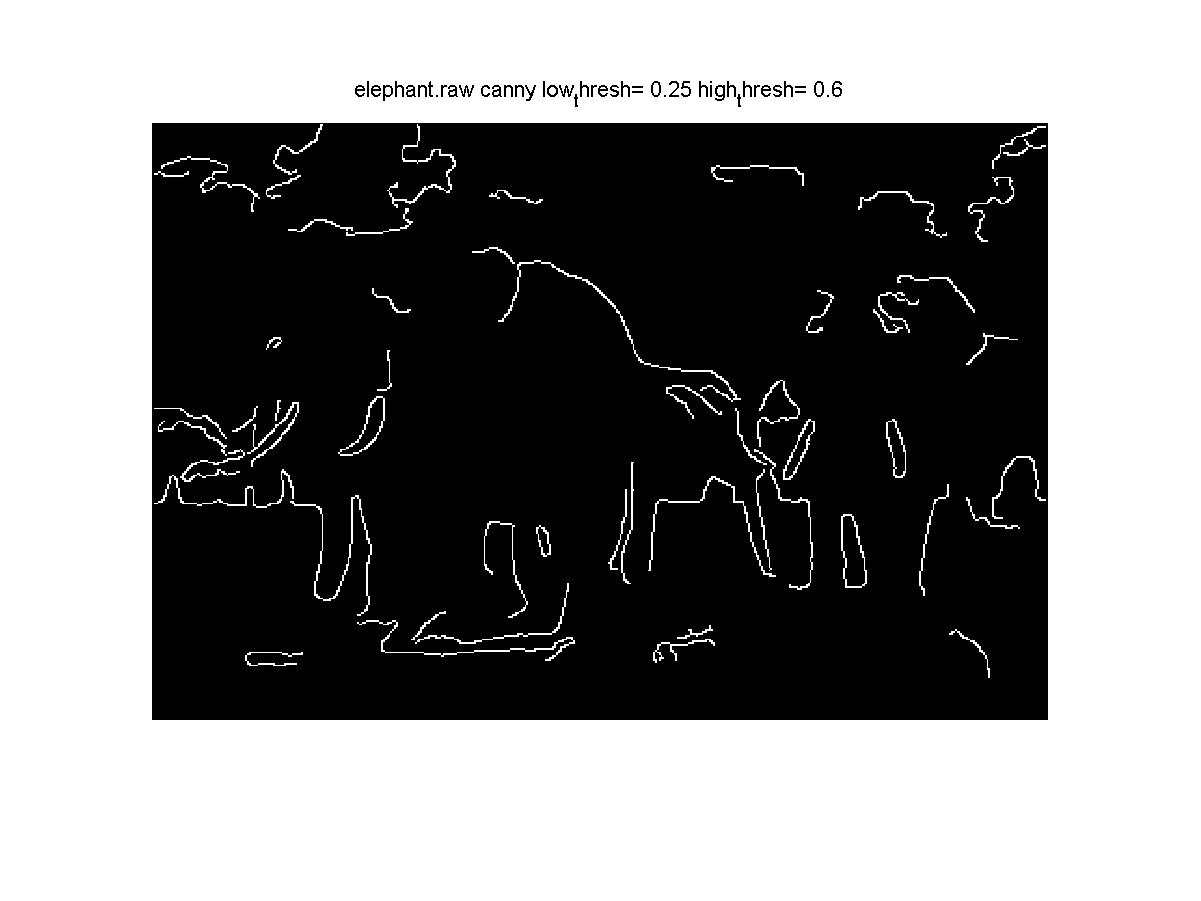
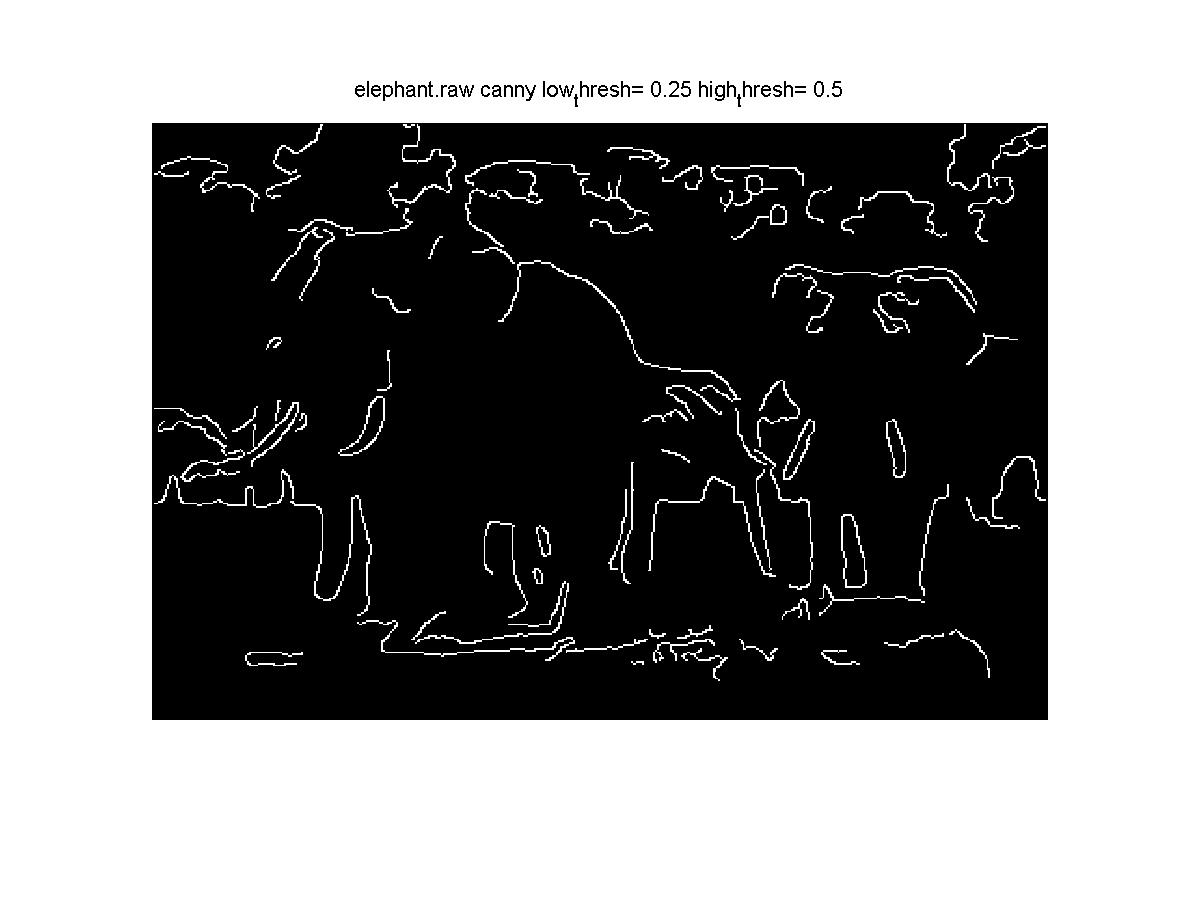
(e) low threshold=0.25 high threshold=0.6 (f) low threshold=0.35 high threshold=0.6

**Fig 1.4** **Canny Algorithm on elephant.raw with different low thresholds and high threshold=0.6**

As we can observe from Fig 1.4, with high threshold fixed, the larger the low threshold is, there will be less irrelevant texture and noise but meanwhile there will also be less continuity on the final edge map.



(a) low threshold=0.25 high threshold=0.3 (b) low threshold=0.25 high threshold=0.4

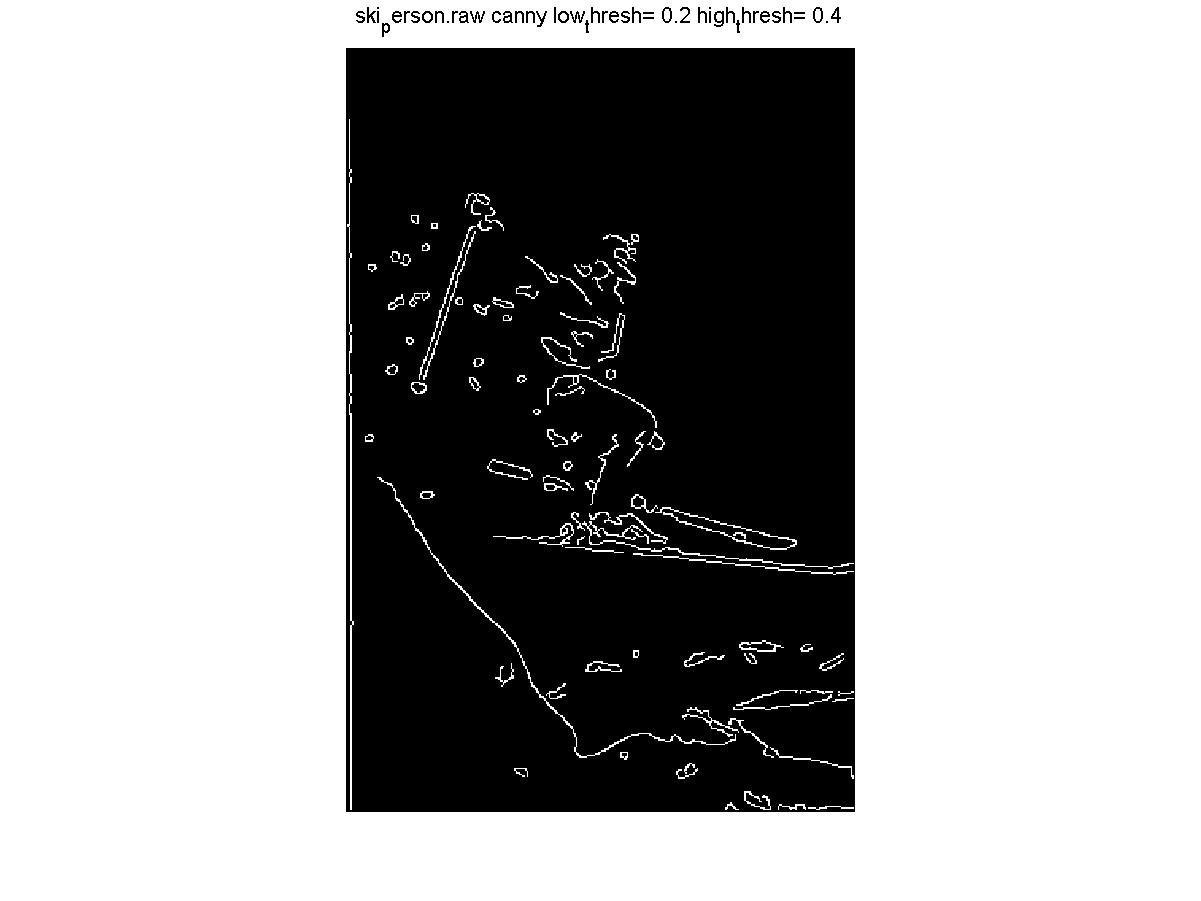
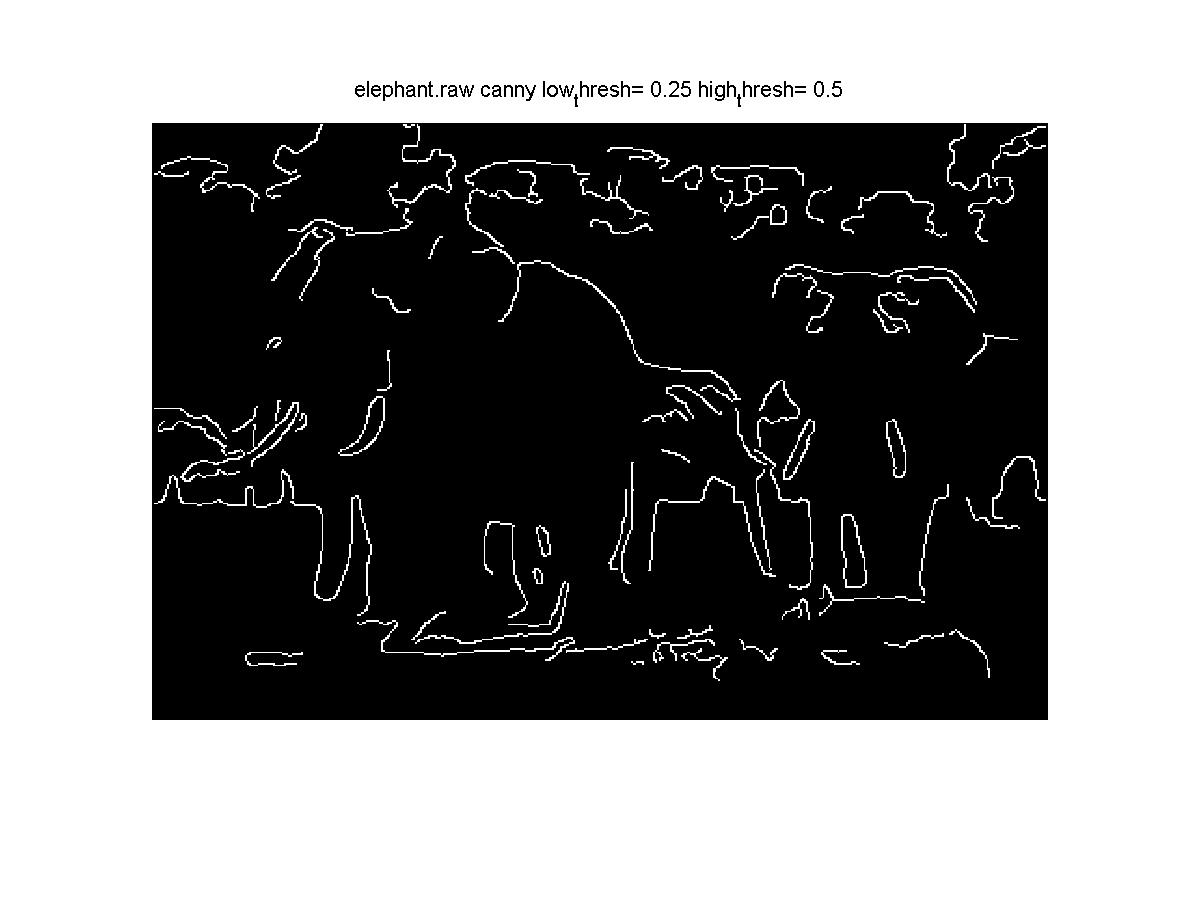


(c) low threshold=0.25 high threshold=0.5 (d) low threshold=0.25 high threshold=0.6

**Fig 1.5 Canny Algorithm on elephant.raw with low thresholds=0.25 and different high threshold**

As we can observe, with low threshold fixed, the remained edge is more and more brief with the high threshold increasing. In Fig 1.5(a), there is a lot of noise and texture which is not edge of major object detected with low high threshold. And in Fig 1.5(d), there is some lack of contour of major object which makes the edge information incomplete because of too high high-threshold value.

The effect of low and threshold on final edge map has been illustrated above in detail with Fig 1.4 and Fig 1.5. Due to the limitation of space, we will **only show the best result for ski\_person and elephant.** Other result can be acquired by my source code if needed.



(a) optimal edge map of elephant.raw (b) optimal edge map of ski\_person.raw

**Fig 1.6 Optimal Result of Final Edge Map of elephant.raw an ski\_person.raw**

The final optimal parameter on elephant.raw is low threshold=0.25 and high threshold=0.5 while the optimal for ski\_person.raw is low threshold=0.2 and high threshold=0.4. With the optimal parameter setting, we can detect the edge of major object more properly. But this optimal result is selected by my observation, which is very subjective. To get more precise evaluation of how well canny algorithm can detect edges, we need to build a probability map by adding final edge map with different parameters together and normalizing it and to compare it with ground truth set to calculate Precision, Recall, F-score value in problem 1(c).

**(c) Structured Edge (15%)**

**I. Approach and Procedures**

**(1):** Structural edge is a machine learning based algorithm for edge detection. In brief, there are two major steps which are training and test. A random forest which contains some decision trees is trained by reduced feature set from training set. After we get the random forest model, every pixel in the test set is judged by the model whether it is edge. Finally, we will form a probability map from the judgement process above. The details are as follows.

To train a decision tree, the image is separated as several patches with each patch calculating 3 LUV channels, 2 magnitude channels and 8 orientation channels in total. For reducing the cost of training process, dimensional reduction is applied by blurring the different image channels by triangle filter with different size and then doing downsampling on filtered maps. After the dimensional reduction process, the pixels remained are the features needed to be trained. For the labels Y in the training set, it is edge maps with certain size so in machine learning it is unstructured data which can not be calculated directly. So, encoding Y to Z is implemented to do feature reduction of Y such as PCA. With preprocessing and feature reduction, decision tree can be trained effectively with the criteria of information gain base on Shannon entropy. To combine the output from decision trees, we just average the outcome of decision trees.

Label Y

Training Image

Encode Y to Z

Generating Augmented channels

Blurring Each Channel with Different Filter Size

Feature Reduction Based on PCA

Downsample Blurred Image

Clustering

Train Information Tree by Standard Information Gain Criteria

**Flow Chart 1.1 The Training Process of SE detection**

Test Image

Downsample

Random Forest

Generate the Result

Output

**Flow Chart 1.2 The Test Process of SE detection**

**(2):** The Random Forest should be classified because the decision tree is very likely to run into overfitting issues because the feature at the beginning in a tree has very big weight. Random Forest contains several independent trees and the outcomes of different trees are balanced. So, it can deliver a more robust generalization capability.

**(3):** The chosen parameters are as follows:

**multiscale=1;** This parameter can make edge detection more accurately because it averages several different results. So, I set it 1 to activate it.

**sharpen=2;** This parameter is used to sharpened the edge even if we already have non-maximum suppression method. By trying different parameters, we can find sharpen=2 can give a better result

**nTreesEval=4;** This is the parameter to decide the number of trees in the random forest. The larger it is, the more generalization capability it will have. To balance the cost and performance, I set it 4 here.

**nThreads=4;** It defines the maximum number of threads for evaluation, so I just use the default value.

**nms=1;** nms here is used to activate non-maximum suppression process which can alleviate the blurring effect and redundancy of edge. To have better performance, I active it here by set it 1.

**II. Experimental Results**



**(a)** elephant.raw Structural Edge Result **(b)** elephant.raw Probability Map of Canny



**(c)** elephant.raw Binarized Structural Edge Result (Threshold=0.15)

**Fig 1.7 elephant.raw Structural Edge Map Result and Comparison with Canny Result**



**(a)** ski\_person.rawStructuralEdge Result **(b)** ski\_person.raw ProbabilityMap of Canny



**(c)** ski\_person.raw Binarized Structural Edge Result (Threshold=0.15)

**Fig 1.8 ski\_person.raw Structural Edge Map Result and Comparison with Canny Result**

**This step is carried out with the help of online open-source code edge toolbox.** In Fig 1.7 and Fig 1.8 we only post Structural Edge (SE) result with the best parameters of random forest above. In the result of both elephant.raw and ski\_person.raw, the structural edge method yields better result because they have more accurate edges and less irrelevant texture information than canny method. We can observe lots of edge discontinuities on canny method result. And in SE method result we can find the edge is very obvious while the texture exists but not very obvious. After thresholding in binarized SE result in both Fig 1.7 and Fig 1.8, we can only observe the accurately described edge only and can hardly see the texture. The SE method has better result overall.

**(d) Performance Evaluation (15%)**

**I. Approach and Procedures**

This step is also carried out by online source code edge toolbox. We first separate each layer of ground truth file to calculate F score, Precision and Recall of sobel, canny, SE method separately and then input the entire ground truth file to calculate the overall mean F score, Precision and Recall. Then we draw the graph to reflect the relationship between F score and threshold in each method for overall ground truth. Here the inputs which are used to compare with the ground truth are respectively normalized Sobel gradient magnitude map, canny probability map generate by my code canny\_proba\_map.m and the output of edgeDemo.m which is the probability map of structural edge. In this step, **I generate the canny probability map** by add lots of canny results with different low and high thresholds which are in the relatively optimal range together and do normalization to the final map. The canny probability map generated in this way can reflect the most averaged and stable performance of canny method overall. **Here I do not delete the thresholding parts in EdgesEvalImg because I have built a probability myself which is not a binary input image which don’t need thresholding and I am just interested in observing how the F score of canny probability map changes with different thresholds.**

**II. Experimental Results**

**(1):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ground truth |  | sobel |  |  | cANNY |  |  | sTRUCTURAL EDGE |  |
|  | P | R | F | P | R | F | P | R | F |
| 1 | 0.3935 | 0.40375 | 0.39848 | 0.62397 | 0.61547 | 0.60757 | 0.75841 | 0.25828 | 0.28039 |
| 2 | 0.42349 | 0.42426 | 0.27687 | 0.58198 | 0.68913 | 0.61865 | 0.75132 | 0.30545 | 0.30917 |
| 3 | 0.55573 | 0.4017 | 0.32188 | 0.71337 | 0.54555 | 0.60637 | 0.78765 | 0.22786 | 0.26512 |
| 4 | 0.54048 | 0.43075 | 0.28657 | 0.50249 | 0.62557 | 0.54636 | 0.37934 | 0.27537 | 0.26078 |
| 5 | 0.59802 | 0.42803 | 0.30855 | 0.5762 | 0.61392 | 0.58262 | 0.72609 | 0.26668 | 0.27957 |
| OVERALL | 0.76973 | 0.41566 | 0.38143 | 0.82679 | 0.61045 | 0.69416 | 0.83485 | 0.26245 | 0.31292 |

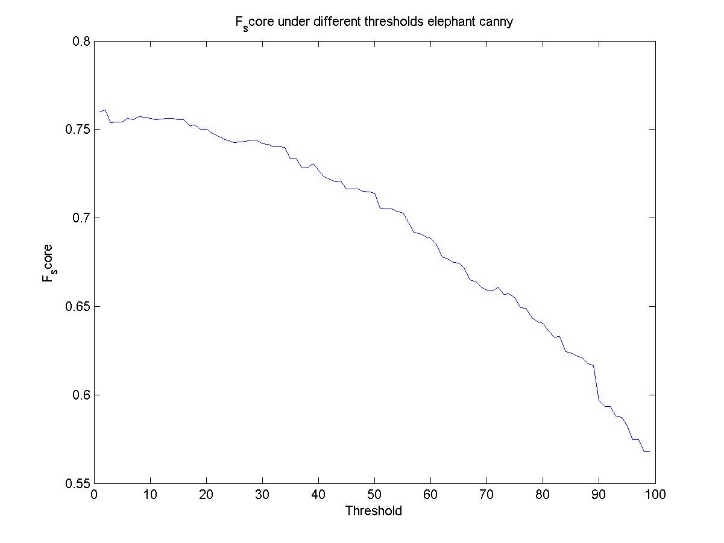
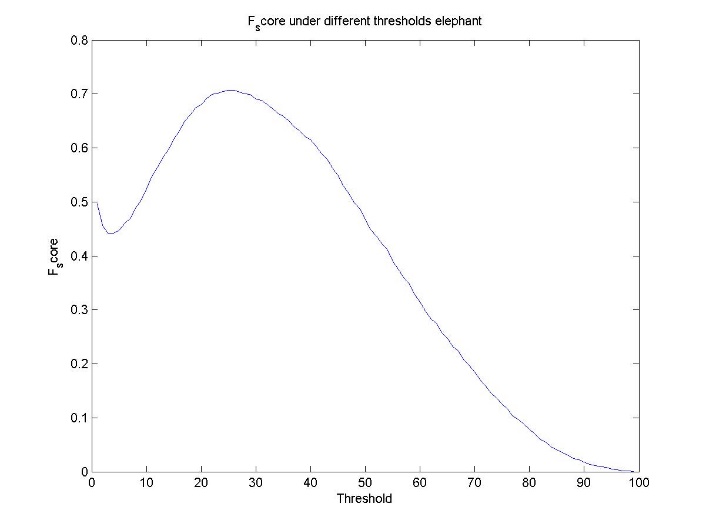
**Table 1.1: Precision, Recall and F-Measure for elephant.raw**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ground truth |  | sobel |  |  | canny |  |  | structural edge |  |
|  | P | R | F | P | R | F | P | R | F |
| 1 | 0.56956 | 0.42884 | 0.30019 | 0.59903 | 0.63497 | 0.60712 | 0.61384 | 0.41376 | 0.38087 |
| 2 | 0.55878 | 0.41627 | 0.30368 | 0.56984 | 0.55646 | 0.55444 | 0.58325 | 0.36811 | 0.34794 |
| 3 | 0.71572 | 0.40047 | 0.33734 | 0.75607 | 0.55134 | 0.62811 | 0.65038 | 0.3284 | 0.34791 |
| 4 | 0.53331 | 0.4253 | 0.28811 | 0.54871 | 0.62188 | 0.57416 | 0.57873 | 0.4143 | 0.3688 |
| 5 | 0.5725 | 0.43465 | 0.30225 | 0.60009 | 0.64743 | 0.61343 | 0.61364 | 0.4204 | 0.38398 |
| OVERALL | 0.74694 | 0.41921 | 0.3622 | 0.77499 | 0.59704 | 0.66448 | 0.65715 | 0.38322 | 0.39152 |

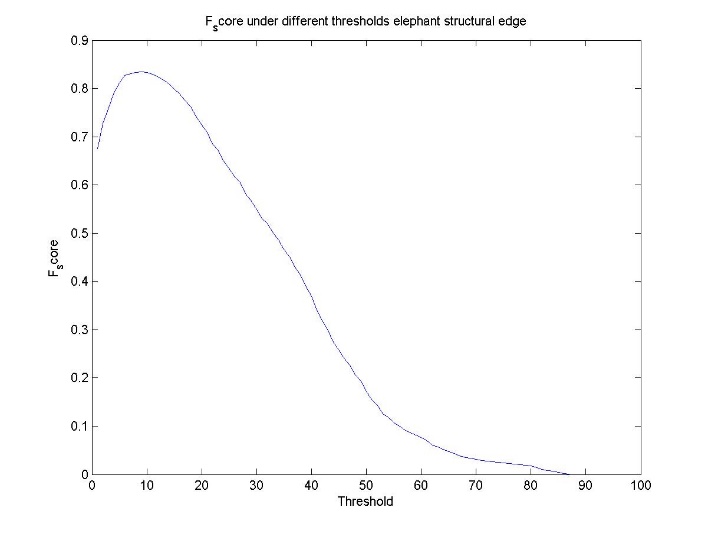
**Table 1.2: Precision, Recall and F-Measure for ski\_person.raw**

Table 1.1 and Table 1.2 show the performance of three detectors on image elephant.raw and ski\_person.raw. The canny method’s performance is much better than that of the Sobel method. It is easier for both Sobel and canny method to detect edge in elephant.raw than ski\_person.raw. **However, because the F score here is the mean of F score under different thresholds and in Fig 1.9 it shows that in structural edge method, the F score changes greatly with the threshold changes** and it falls drastically when threshold is greater than 0.4. **Because higher threshold for SE method will eliminate most of the edge so the performance will deteriorate very fast. Thus, mean F score of different thresholds can hardly reflect the real performance of SE method correctly here because it introduced lots of low performance F scores generated with improper threshold.** We should use the graph of F score and different threshold to find the peak F score to evaluate the performance correctly. **And we can observe from Fig 1.9 that in the F score and threshold graph of elephant.raw, based on the peak F score in the graph, SE method performs better than both canny and Sobel method.**

**(2):**

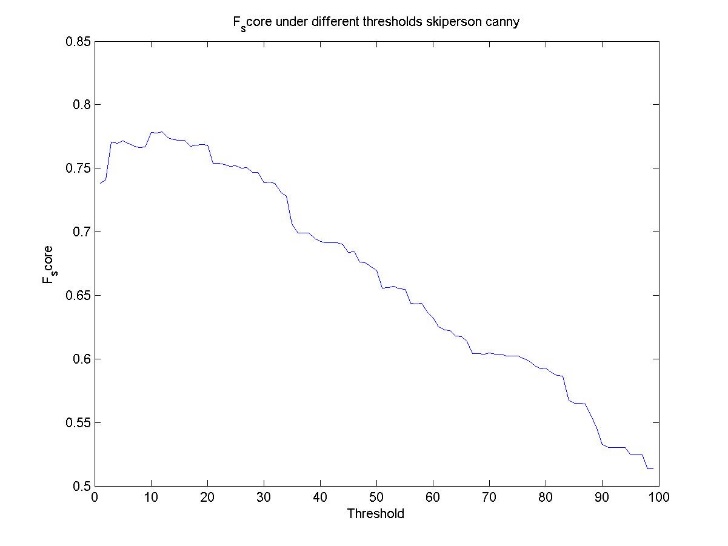
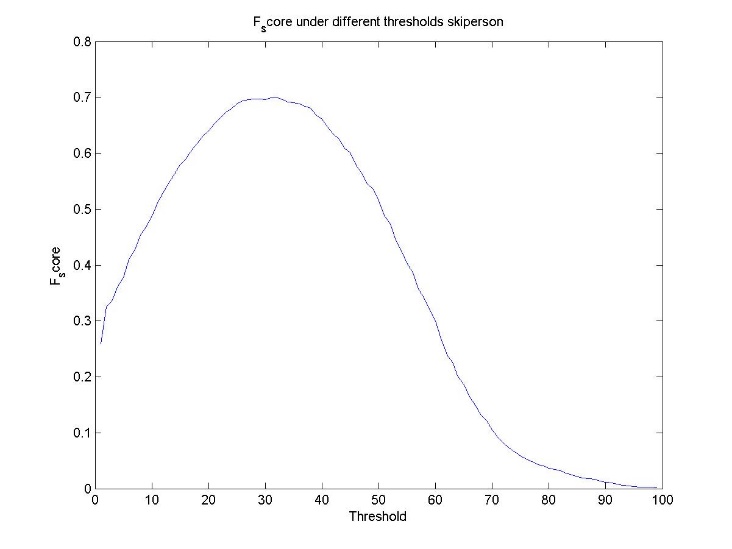


(a) Sobel F score (b) Canny F score

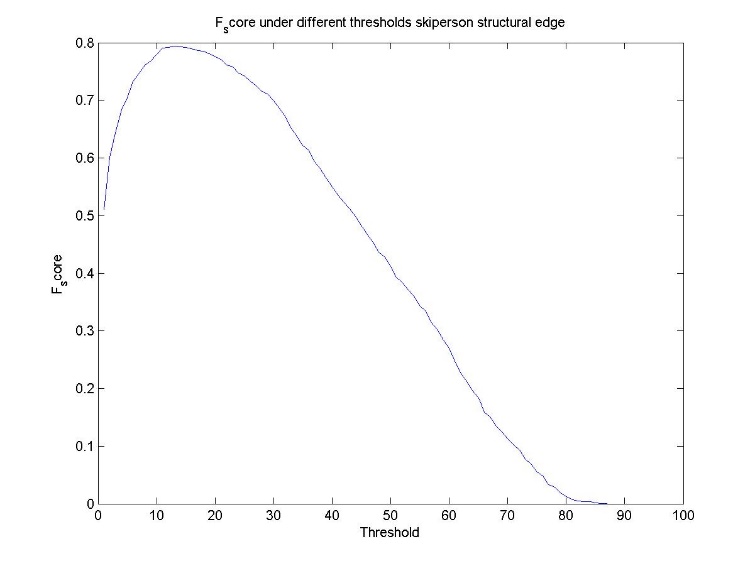


(c): Structural Edge F score

**Fig 1.9 F score and thresholds under different methods elephant.raw**



(a) Sobel F score (b) Canny F score



(c): Structural Edge F score

**Fig 1.10 F score and thresholds under different methods ski\_person.raw**

Fig 1.9 and Fig 1.10 reflect how F score changes by threshold in the three methods. As we discussed previously, we should use peak value in Fig 1.9 and Fig 1.10 rather than mean value based on threshold to evaluate the performance of each methods on elephant.raw and ski\_person.raw. **For results of both the raw image, the peak F score is highest in structural edge method and is the lowest in Sobel method, which means structural edge performs the best and Sobel the worst. And canny is in the middle of Sobel and structural edge method.** From other perspective, structural edge method is very sensitive to setting of thresholds while canny is very robust when threshold changes. This can account for the abnormal phenomenon in Table 1.1 and Table 1.2 where SE method have very clear edges and less irrelevant texture but only to get lower F score value than that of canny. The probability map of structural edge method does not have much redundant texture or details and is mainly composed of useful edges, so as a result the F score deteriorates very fast when the threshold is greater than a certain critical point. Because real useful edges are being removed gradually when threshold is too big.

**(3):** Based on the comparison of Fig 1.9, Fig 1.10, Table 1.1 and Table 1.2, **we can conclude that the image elephant.raw result is more likely to get a higher F score on all three detection methods.** From an intuitive perspective, the pattern in the texture and edge in ski\_person is very similar to each other which makes the detection difficult. Meanwhile in elephant.raw the texture and edges have more difference from each other making it a higher tendency to get a high F score.

**(4):** Precision can measure how many pixels which are labeled as edges are really edges, while recall can test how many edge pixels are labeled as pixels. If we only consider precision and only label one single point which is really edge as an edge pixel, the precision will easily reach 100 percent. On the contrary, if only recall is considered and we follow similar steps, recall could also be 100 percent. It obvious that both of the situations above are meaningless for evaluation. Thus, F-Score can strike a good balance between recall and precision. Only in the situation where both recall and precision are high can we acquire a high F score. You can never get a large F score in the abnormal situation above. Let 𝑃 + 𝑅 equals constant C first, so 𝐹 = 2\*𝑃\*(𝐶−𝑃)/𝐶 . And when we take partial derivative of F to P and R respectively and set them both 0, we can get at the maximum point of F score, 𝑅 = 𝑃 = 0.5\*𝐶. Thus, it is proved that F score reaches the maximum when precision is equal to recall

**Problem 2: Digital Half-toning (30%)**

**(a) Dithering (15%)**

**I. Approach and Procedures**

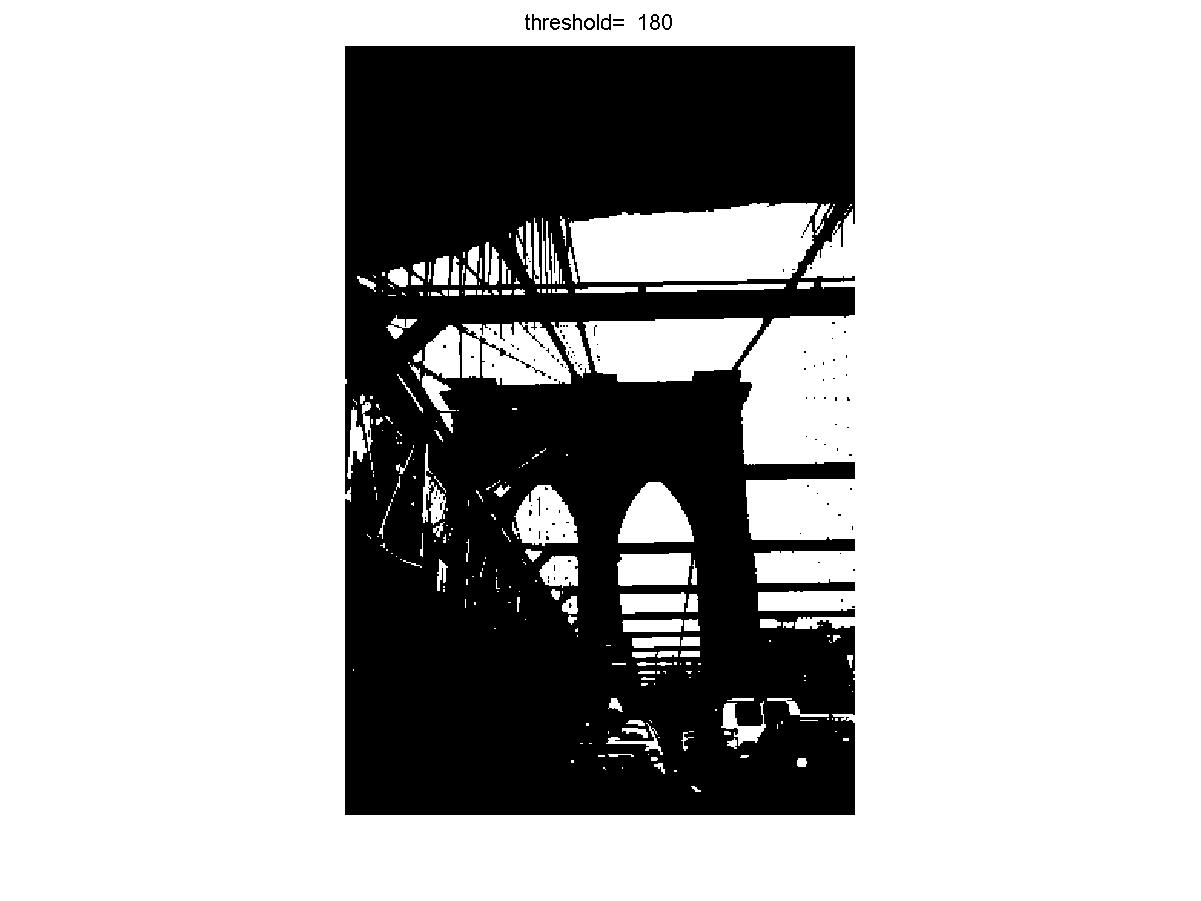
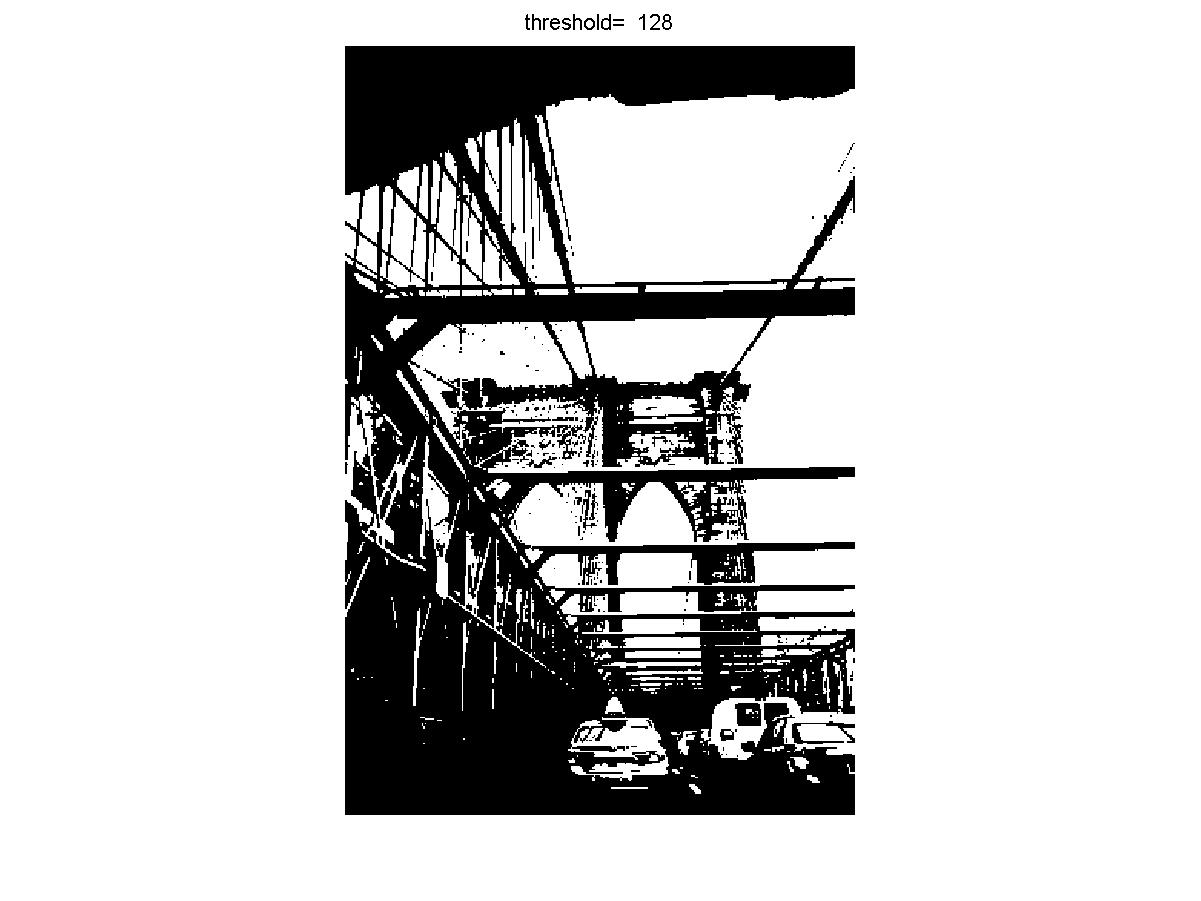
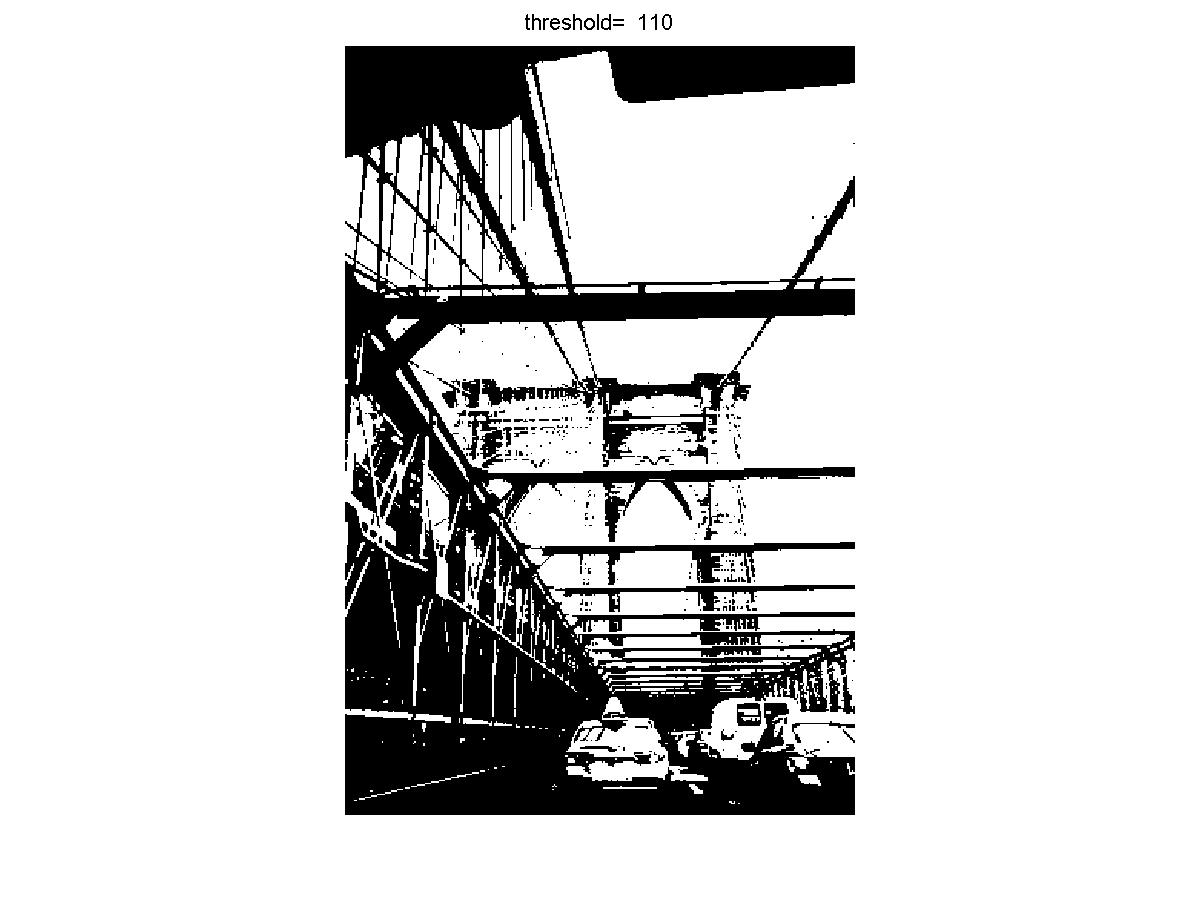
Dithering is a half-toning implementation method that can transform multi-level intensity quantized image to binary image which only has the value of 0 and 1. The goal of dithering is to transform the original image to a binary image so that the printer can print it. Meanwhile, the method should keep the image look as close to the original one as possible.

**1.** Fixed Thresholding means we set an empirical threshold value first. And we traverse the whole image to compare each pixel to the threshold. The pixel larger than the threshold will be set to 1 which is full white and the pixel smaller than or equal to the threshold will be set to 0 which means full black. But with this method, there will be a huge loss of texture or detail information of objects in the image even if we set a proper threshold. So more advanced method needs to be proposed.

**2.** Random Thresholding is applied by setting a random threshold when traversing each pixel and compare the pixel with its corresponding threshold. When the pixel is larger than the threshold it will be set to 1 which is full white and when the pixel is smaller than or equal to the threshold it will be set to 0 which means full black. This method alleviates the huge loss of detail or texture information and very obvious fixed style. However, the result still suffers from serious noise and information lacking issues and the result just looks messed up.

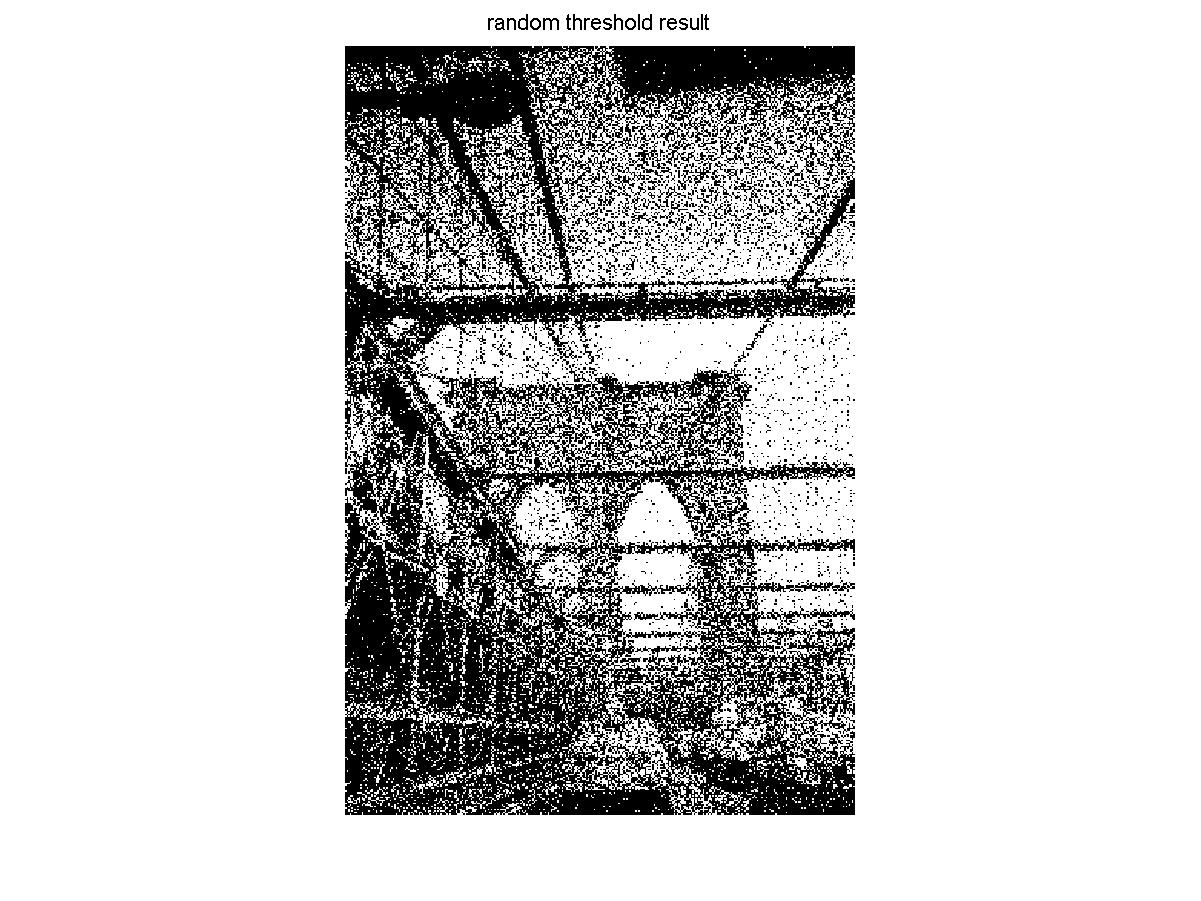
**3.** Dithering matrix is a method which compare threshold matrix with a certain fixed pattern with corresponding pixels to do binarization. This can turn out more detail and eliminate the noise better than the methods above because there are lots of different thresholds in the matrix, which can depict the brightness of the matrix region by changing the density of bright pixels. For example, a region of 2 by 2 contains 4 pixels with 127 intensity, but you can only print 0 or 255 on a printer. Here by dithering matrix with thresholds 255, 192,126, 63 in it, we can get 4 pixels with 2 full bright and another 2 full dark. Due to the limitation of the resolution of human eyes, we can observe a very similar intensity as if the 4 points printed are all with 127 intensity. Here is what makes it superior to the two methods above. However, the fixed artificial pattern in the matrix exhibits a grid-like visual effect which needs to be improved.

**II. Experimental Results**



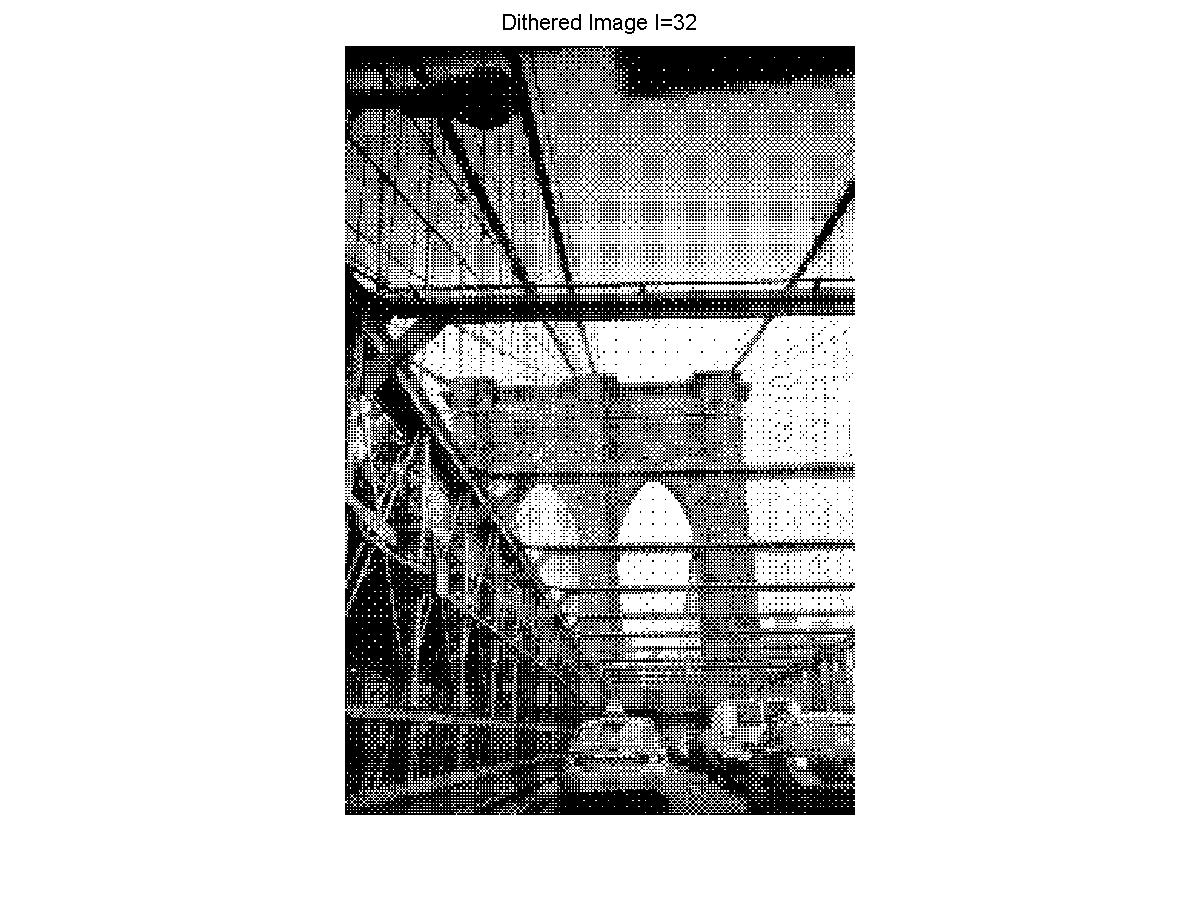
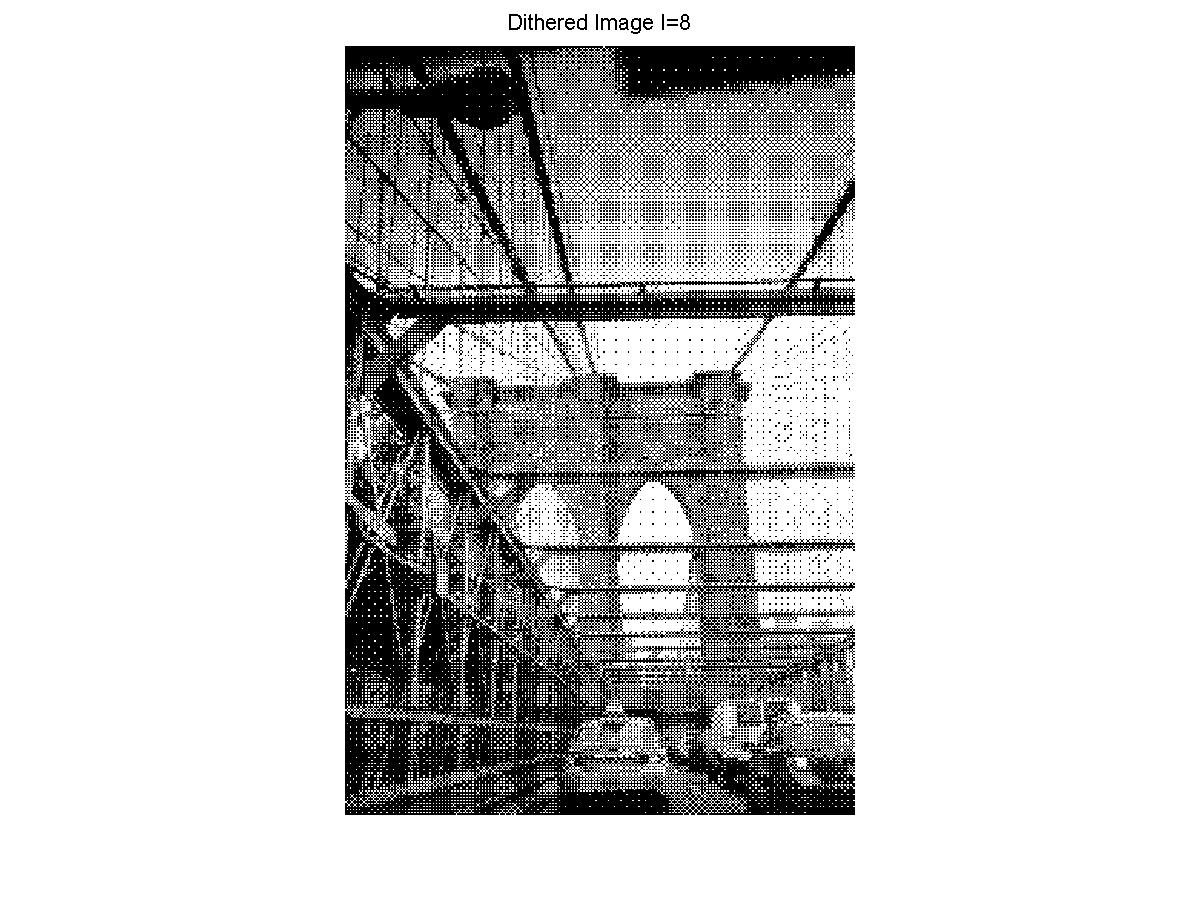
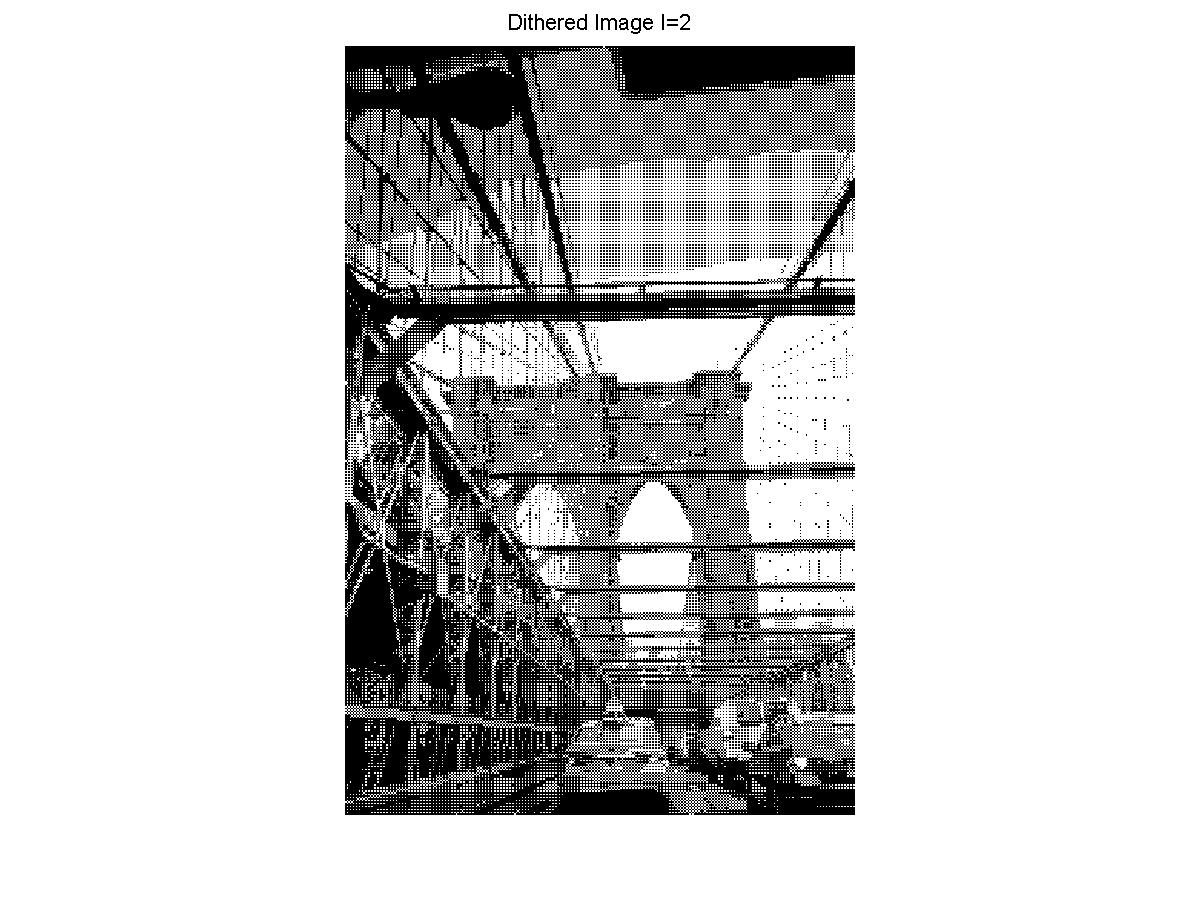
**Fig 2.1 Results of Fixed Threshold Method with Different Thresholds**

As we can observe in fixed threshold method, there is a huge loss of texture and detailed information even under relative thresholds like 90 or 110.



**Fig 2.2 Random Threshold Method Result**

In random threshold method, there is a noticeable improvement in texture and detailed information. Some details of the road and bridge can be distinguished in Fig 2.2 rather than just full black in fixed threshold method in Fig 2.1. But the serious noise corrupts the entire image which makes it still unable to see the small details and very different from the original image.



**Fig 2.3 Results of Dithering Matrix Method with I=2, 8 and 32**

As we can see from Fig 2.3, we can notice that results of dithering matrix method have more details than the method above. However, it shows very obvious artificial fixed grid-like pattern in the image which makes it not so similar to the original one. With larger matrix I, this symptom can be alleviated. For example, I=32 result has less grid-like feel than when I=2 and 8, but it still exists so error diffusion method is proposed and we will illustrate it later.

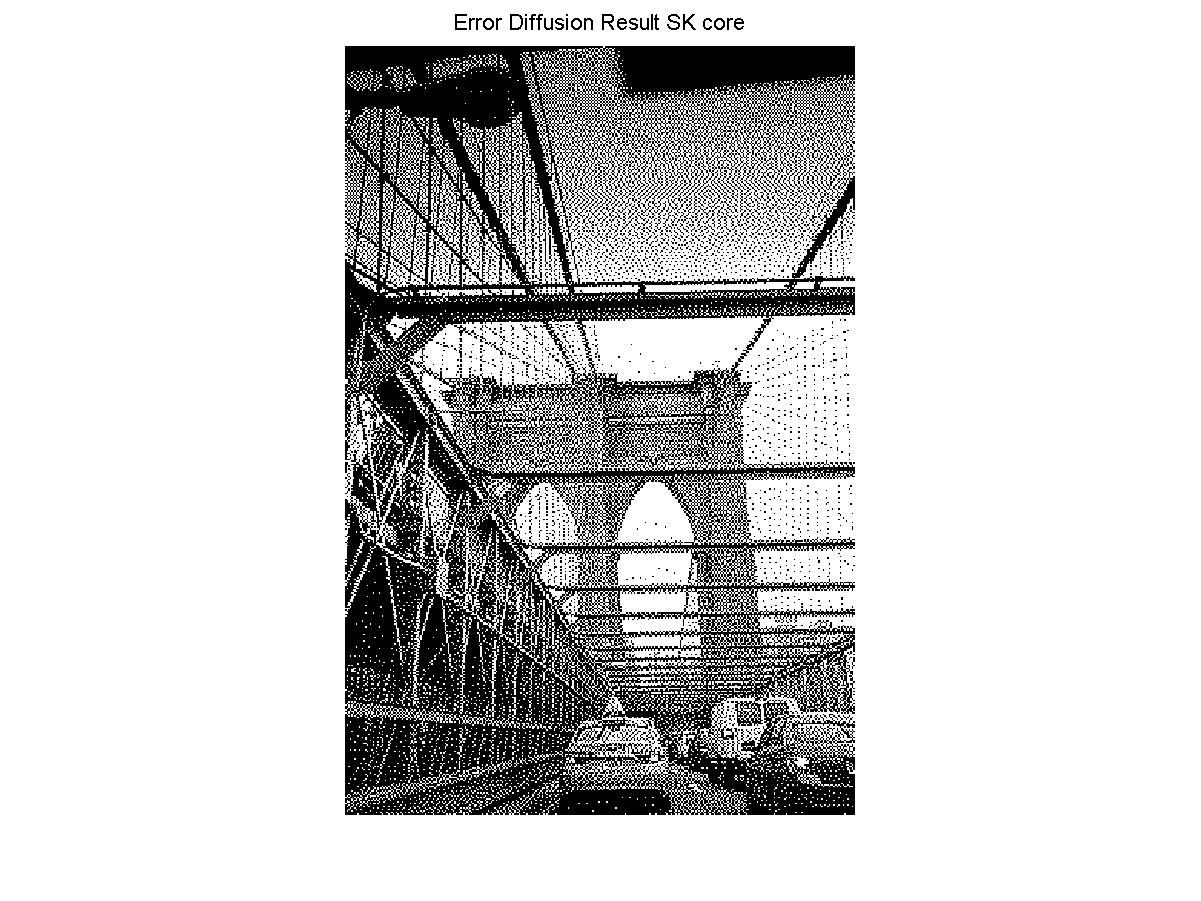
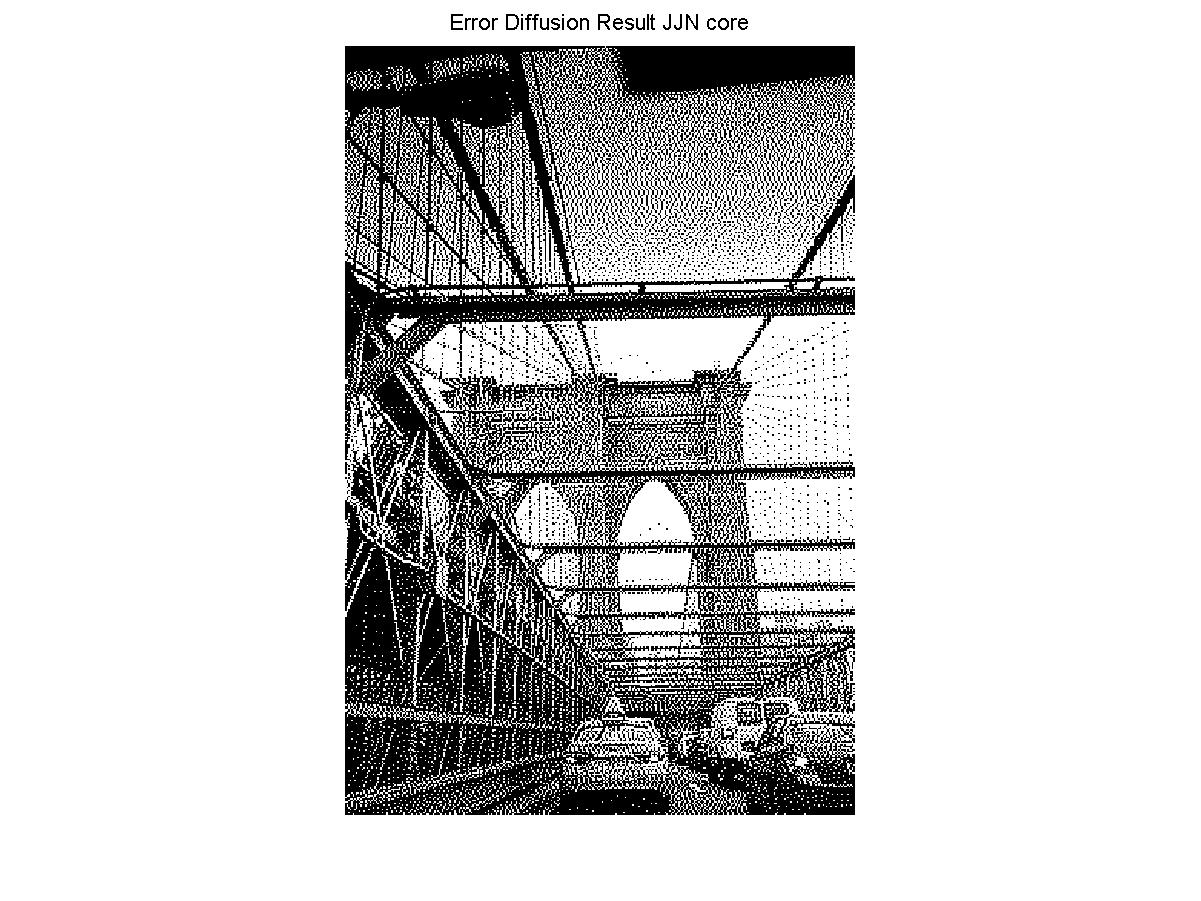
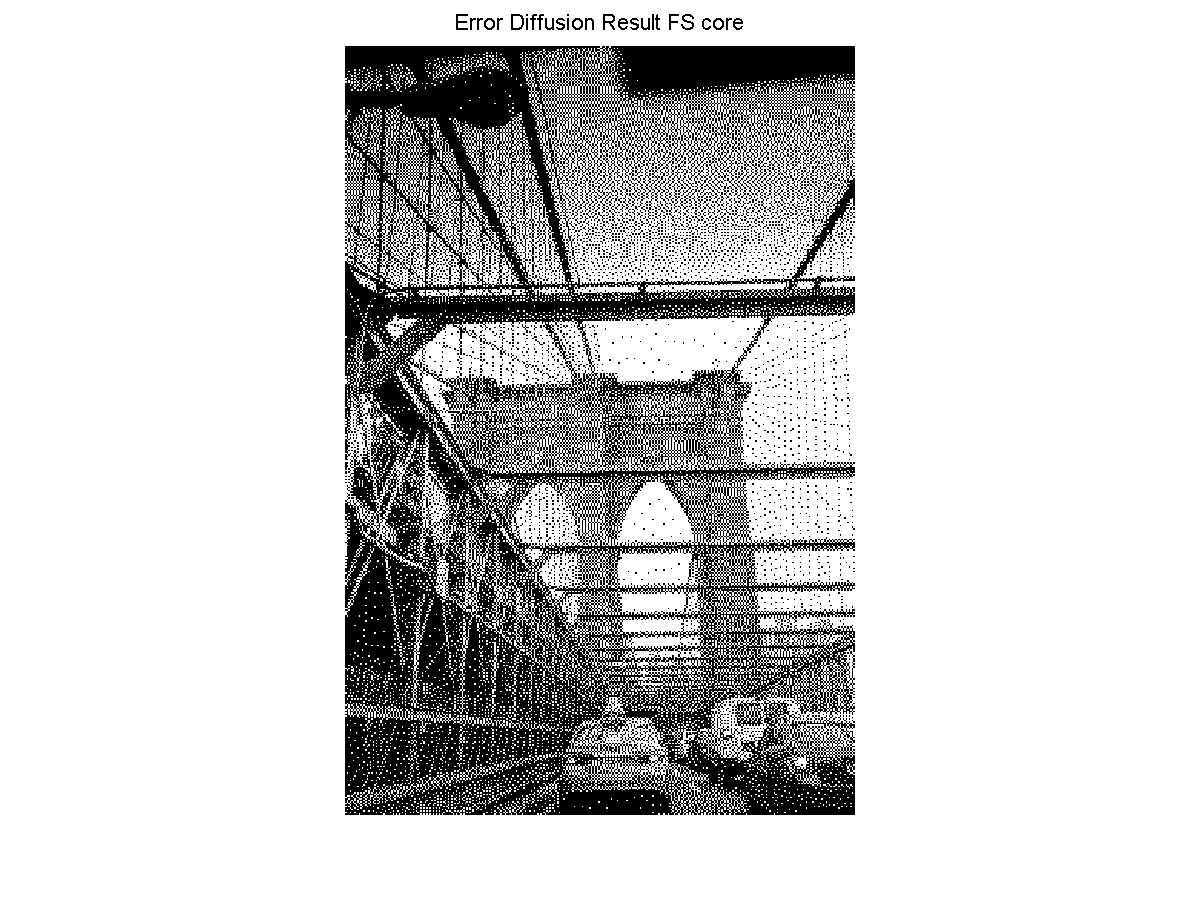
**(b) Error Diffusion (15%)**

**I. Approach and Procedures**

Error is a new kind of halftoning method which considers the error of neighbor pixels and applies it to adjust the threshold of next pixel dynamically. This can avoid fixed artificial pattern effectively and make the image more similar to the original one.

With the error diffusion matrix, it begins with top left pixel and then to the right. When we change row, go direct straight down from the last pixel in the row above and go from rightmost to leftmost. Every time the row changes, we should inverse the traversing direction so it will look like a zig-zag pattern overall. And for every single pixel, its binarized value based on corresponding threshold (usually 0.5) will be compared to the real value to get an error term and this error term will be diffused to its neighbor by a certain scaler for each neighbor pixel. Floyd-Steinberg Error Diffusion, JJN Error Diffusion and Stucki Error Diffusion are absolutely the same on principles but just different in the diffusion matrix which determines the way error diffuses to the neighbor. For each pixel, it has already accumulated error value from some of its neighbor pixels processed before prior to processing, which is just the way current pixel binary result is affected by neighbor pixels errors.

**II. Experimental Results**

****

**Fig 2.4 Results of Error Diffusion Method with Different Matrix (Floyd-Steinberg, JJN, Stucki)**

Owning to the error diffusion mechanism which can adjust the threshold of a pixel dynamically based on error feedback from neighbor pixels, the results of all three matrices are better than previous one as shown in Fig 2.4. It nearly completely addressed the issue of gird-like visual feel and we can observe more detail from the image. Also, there is also less noise.

From the details in three images in Fig 2.4, we can see that there is some slight difference among the three matrix results. Floyd-Steinberg’s method exhibits a smoother edge than the other two methods, but it also eliminates some details and its edge does not stand out very much. For JJN’s method, the edge is very sharp, which makes the major object stand out and easy to distinguish. But there is lots of spotty feel and edge discontinuities. Stucki’s method’s performance is between that of Floyd-Steinberg’s and JJN’s. It has both relatively sharp edges and smoothness of the picture overall. **As a result, personally I prefer the Stucki matrix-based method.**

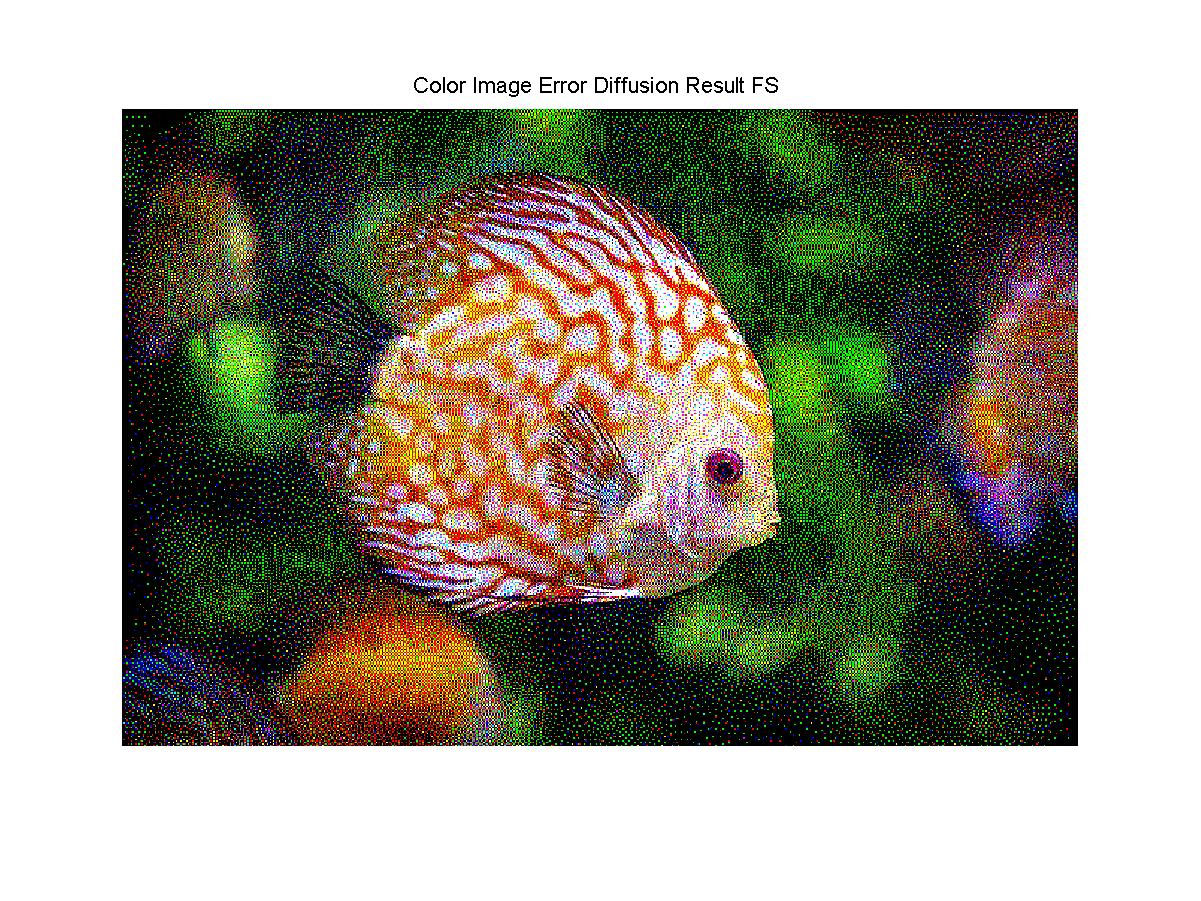
**Problem 3: Color Half-toning with Error Diffusion (20%)**

**(1) Separable Error Diffusion**

**I. Approach and Procedures**

To do the halftoning of color image by error diffusion method, the simplest one is to apply error diffusion method to three color channels respectively which is separable error diffusion here. When implementing this method, we first separate them into three channels and process each channel with error diffusion method using Floyd-Steinberg matrix as if they are gray scale image. Then we combine the processed three channels and show the final result. This method is simple but it does not take the correlation of the three channels into account.

**II. Experimental Results**



**Fig 3.1 Separable Error Diffusion Result of Color Image**

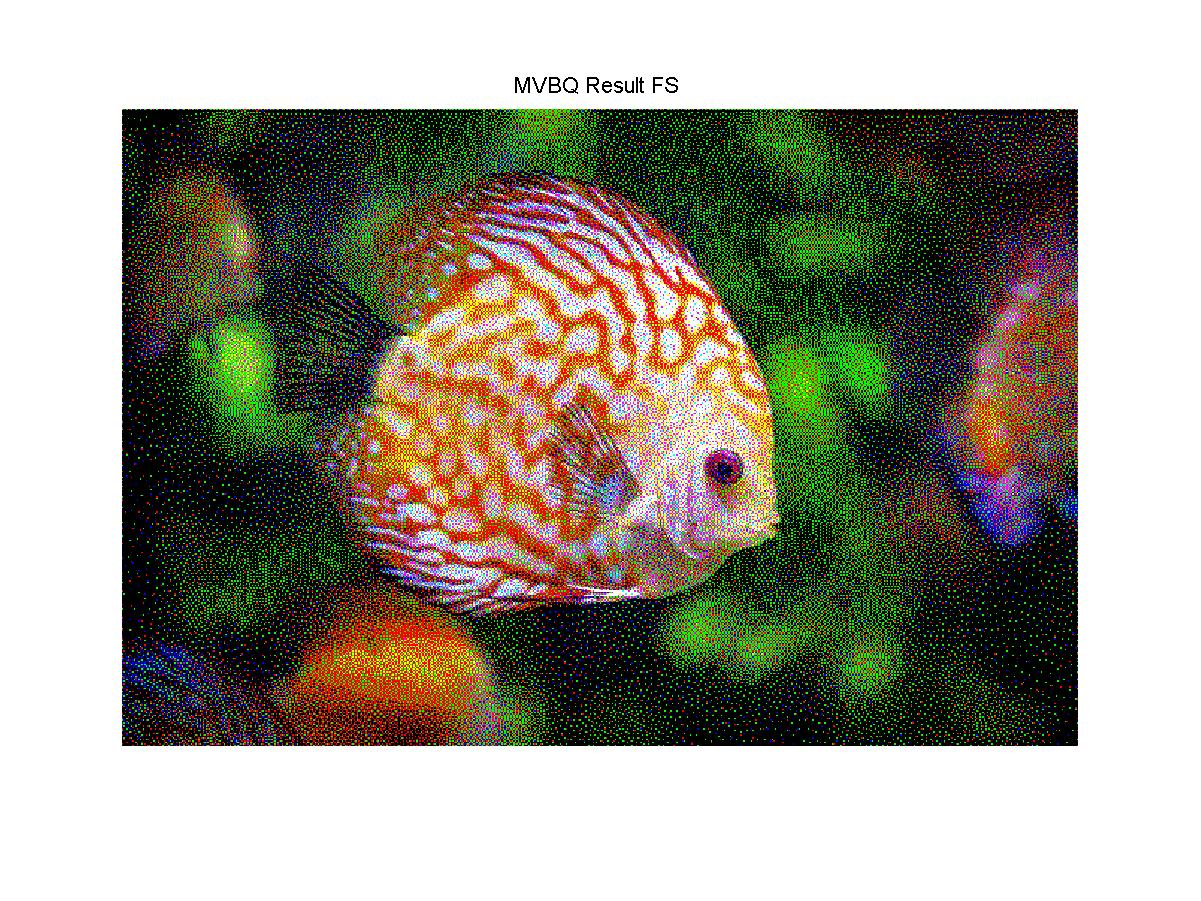
Because of the separation of the process in three channels, we can observe a lot of undesired noise dots in Fig 3.1. In the region where it should be very dark there are lots of yellow points which makes the picture messed up and noisy. So more advanced method which take the correlation between pixels in three channels into account should be proposed.

**(2) MBVQ-based Error diffusion**

**I. Approach and Procedures**

To refine the shortcoming of no correlation of three channels when processing, MBVQ based error diffusion is applied in this step. The assumption which MBVQ is based on is that human eyes are more sensitive to the intensity of color rather than the tone of color. To apply the correlation between three channels, this method first decides which color tetrahedra a certain color pixel is in and the different tetrahedras are partitioned by the brightness. Then the pixel is projected to the nearest vertex of the tetrahedra it belongs to make the color tone change more slightly. This approximation can reduce three colors by only one color. And just because the projection process considers all three colors, we can eventually make three channels correlated in the final calculation. In the coding process, I used the open-source code function getNearestVertex.m.

**II. Experimental Results**



**Fig 3.2 MBVQ Based Error Diffusion Method Result**

As we can observe from Fig 3.2, with the relevance of three channels considered, the image is noticeably smoother and similar to the real. There are less undesired dots in a certain color region. For examples, there are less white and black dots on the fish body’s orange part and less yellow and blue dots on the body’s white part. And the edge really stands out which makes the major objects more salient rather than just messed up in the background when compared to Fig 3.1.