Term Project :
Implement and Improve
Multiscale Error Diffusion Technique (MED)
for Digital Halftoning

EE 569 Term Project

December 4, 2009

I. Introduction

Halftoning is one of the oldest applications of image processing, since it its essential for the printing process. Because many image rendering technologies only have binary output, we have to convert a gray-level image into a binary image and try to keep the image detail as much as we can. There are many methods to perform digital halftoning. They can be grouped in three major categories: (1) Direct Binarization, (2) Dithering, and (3) Error Diffusion.

The Direct Binarization method tries to minimize the squared error, so the threshold of this method is 0.5 for normalized gray level from 0 to 1. However, this fixed-level quantization method produces the worst result, because the output binary image usually has some continuous black or white regions. It will lose lots of detail of original gray image, although it has the minimum squared error. We usually use the following equation to implement the fixed-level halftoning.

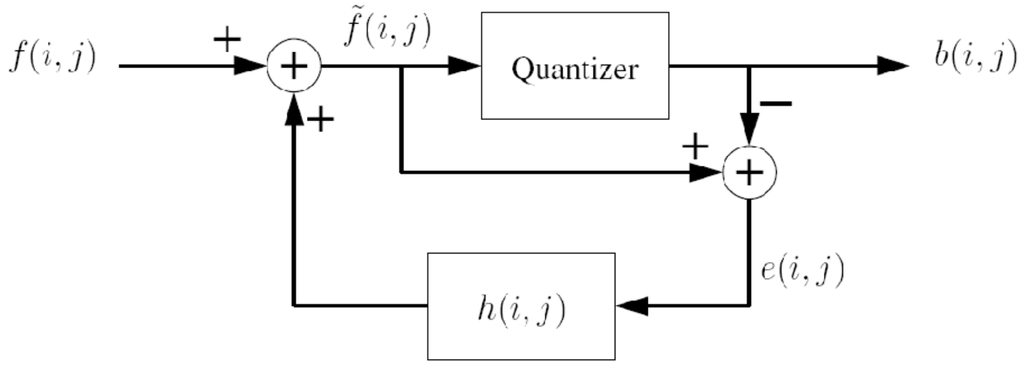
$$G(i,j)=\begin{cases}0 & if \;\; 0\le F(i,j)<T \\ 1 & if \;\; T\le F(i,j)<1\end{cases}$$

The Dithering method means the addition of some kind of noise prior the quantization. The classical dithering method is ordered dithering method. We use some specific ordered masks to determine whether the pixel should be turned on or not. Following equations are the basis of the Dithering Matrix Pattern method.

$$I_2(i,j)=\begin{bmatrix}1 & 2\\ 3 & 0\end{bmatrix},\;\; I_{2n}(x,y)=\begin{bmatrix}4*I_n(x,y)+1 & 4*I_n(x,y)+2\\ 4*I_n(x,y)+3 & 4*I_n(x,y)\end{bmatrix}$$

$$T(x,y)=\frac{I(x,y)+0.5}{N^2},\;\; G(i,j)=\begin{cases}1 \;\; if \;\; F(i,j)>T(i \bmod N, j \bmod N)\\ 0 \qquad\qquad\qquad\qquad\qquad\qquad else\end{cases}$$

The Error Diffusion method is based on the simple principle that once a pixel has been quantized, thus introducing some error, this error should affect the quantization of the neighboring pixels. The steps of Error Diffusion method is that when we threshold the input pixel, we can store the difference between the output pixel value and the input pixel value, which is the error, and then diffuse the error into the neighboring pixels. Following is the flow chart and diffusion filters of Floyd-Steinberg's Error Diffusion.



Serpentine Scanning :

$$\text{Left to right} = \frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix} \qquad \text{Right to left} = \frac{1}{16}\begin{bmatrix} 0 & 0 & 0 \\ 7 & 0 & 0 \\ 1 & 5 & 3 \end{bmatrix}$$

In this project, I will implement a different technique for digital halftoning than above methods. This digital halftoning technique is based on Multiscale Error Diffusion (MED) algorithm. This method uses an image quadtree to represent the difference image between the input gray-level image and the output halftone image. Besides, it also uses “maximum intensity guidance” method for assigning dots and diffusing the quantization error.

II. Problem to Solve

“Generate digital halftoning binary images from some real world gray-level images by using Multiscale Error Diffusion algorithm and compare the results with other digital halftoning methods. If possible, improve the original Multiscale Error Diffusion algorithm”

III. Motivation

In homework 2, we have implemented the fixed-level quantization method, the ordered dithering method and Floyd-Steinberg's error diffusion method to deal with the digital halftoning problem. Compared with these three methods, the error diffusion method can produce the best result. However, is the Floyd-Steinberg's error diffusion method the best way for digital halftoning? In the lecture, professor said that there is another method for digital halftoning, which is multscale error diffusion method, can provide a better result for halftoning but it costs more memory space and computation time than other techniques. After I surveyed the related papers, I found that this method has some drawbacks as professor said, but I tried my best to implement and improve this method in this project.

IV. Selected Algorithm

After surveying papers which are related to digital halftoning, I selected the original Multiscale Error Diffusion Algorithm to achieve my goal. Following is the basic idea of Multiscale Error Diffusion Algorithm. Using a two-step procedure, we focus on the sub-images that have the largest intensity, and thus the greatest need for dots on the output image B. At each iteration of the algorithm, we introduce a white dot (value = 1) at some specific location of the output image B. The location is chosen in a greedy way by traversing the error tree E_k, top-to-bottom. We then diffuse the error to the neighbors of that pixel in the E_r array with a non-causal filter, ensuring that there is no error leakage and also that the total decrease in the E_r array is equal to 1. Finally, we update the error-image pyramid with the new error values, thus decreasing the values of each level in total by 1. This procedure is applied iteratively until the intensity of the root of the error tree is less than 0.5, which implies that the global error is bounded in absolute value by 0.5.

There are three major concepts in Multiscale Error Diffusion algorithm, which are (1) Image Quadtree Representation, (2) Maximum Intensive Guidance and (3) Multiscale Error Diffusion.

- **Image Quadtree Representation :**

Let X, B, and E be arrays of dimension $K \times L$, corresponding to the input image, output binary image, and the difference (or error) image, as defined by $E = X - B$. In this project, we consider the case of square images, i.e., $K = L = N = 2^r$. Based on this representation, we can get a set of image arrays X_k with $0 \le k < r$ and where $r = \log_2 N$. Thus, $r+1$ is the total number of different of the largest size of dimension $N \times N$, X_{k-1} denotes that of size $\frac{N}{2} \times \frac{N}{2}$, and so on. The elements of the coarser resolution arrays are defined by following equation.

$$X_k(i_k, j_k) = \sum_{i=0}^{1}\sum_{j=0}^{1} X_{k+1}(2i_k + i, 2j_k + j),$$
$$where\ i_k, j_k = 0,\dots,2^k - 1; k = r-1,\dots,0.$$

The coarsest resolution X_0 is simply a one-element array, whose value is the total intensity of the whole input image. This kind of image arrays is called an image pyramid or image quadtree.

To deal with a good halftoning, we require that the input and output pyramids are as close as possible on all levels. We want to minimize $E_k = X_k - B_k$ for $0 \le k < r$.

- **Maximum Intensity Guidance in an Image Pyramid :**

Start from the root of error image E_0, which consists of one element $E_0(0,0)$. Consider the four sub-images of the E_0, which are $E_1(i_1, j_1)$, where $i_1 = 0,1$ and $j_1 = 0,1$. We find the highest local intensity of the four sub-images. Next, consider its four sub-images at level 2, i.e., $E_2(i_2, j_2)$, and find the highest intensity value of the specific four sub-images. Continue this procedure until the finest resolution $E_r(i_r, j_r)$ is reached. At the end of this procedure, we have chosen and kept the location $(\hat{i}_r, \hat{j}_r)$ of one pixel of the original image and the value $E_r(\hat{i}_r, \hat{j}_r)$ at the corresponding location of the error-image pyramid.

- **Multiscale Error Diffusion in an Image Quadtree :**

In this step, we quantize the specific pixel in the constructed error-image quadtree. Given the pixel chosen by "Maximum Intensity Guidance" in previous step, we quantize this pixel by setting $B_r(\hat{i}_r, \hat{j}_r) = 1$, i.e., we assign a white dot at the corresponding location of the output array. The quantization error is given by

$$e_q = E_r(\hat{i}_r, \hat{j}_r) - 1$$

In original algorithm, they set $E_r(\hat{i}_r, \hat{j}_r) = e_q$ and diffuse the error to $(\hat{i}_r, \hat{j}_r)$'s neighbors in the error E_r by using a non-causal diffusion filter. Following is the original implement 3×3 diffusion filters.

$$H_{center} = \frac{1}{12}\begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad H_{corner} = \frac{1}{5}\begin{bmatrix} 0 & 0 & 0 \\ 0 & -5 & 2 \\ 0 & 2 & 1 \end{bmatrix} \quad H_{side} = \frac{1}{8}\begin{bmatrix} 0 & 0 & 0 \\ 2 & -8 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

However, this error diffusion method is not flexible and not robust. As you can see, if we want to implement a 3×3 filter MED algorithm, we need to hardcode nine diffusion filters for the boundary pixels. In addition, if we want to implement larger size filter for MED algorithm (5×5 or 7×7), the filter number will increase greatly and hard to maintain in the program. Because the original implement of this step is not flexible for different size of diffusion filters, I improved this inconvenience by changing the center element of filters into zero, and calculating the coefficient of diffusion filters dynamically based on its location. By using this method, we don't need lots of different filters for boundary pixels. We only need one global filter for one specific size diffusion filter version. Although the algorithm works with any choice of filter, I chose the 2-D Gaussian filter for the different size diffusion filters. Following is the equation of 2-D Gaussian distribution.

$$p(x_1, x_2) = \frac{1}{2\pi\sigma^2}\cdot\exp(-\frac{x_1^2 + x_2^2}{2\sigma^2})$$

However, the center element is different from Gaussian filter, which is zero for implement convenience. Following are the filters that I used in my program. The coefficients are calculated dynamically based on its location.

$$H_{1\times1} = [1],\ H_{3\times3} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 2 & 1 \end{bmatrix},$$

$$H_{5\times5} = \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 0 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix},\ H_{7\times7} = \begin{bmatrix} 1 & 1 & 2 & 2 & 2 & 1 & 1 \\ 1 & 2 & 2 & 4 & 2 & 2 & 1 \\ 2 & 2 & 4 & 8 & 4 & 2 & 2 \\ 2 & 4 & 8 & 0 & 8 & 4 & 2 \\ 2 & 2 & 4 & 8 & 4 & 2 & 2 \\ 1 & 2 & 2 & 4 & 2 & 2 & 1 \\ 1 & 1 & 2 & 2 & 2 & 1 & 1 \end{bmatrix}$$

My implementation concept is basically the same with original algorithm, but more flexible. Besides, it follows that the total decrease in the error image E_r is equal to 1, i.e., there is no error leakage, which is just like original algorithm.

V. Results

To evaluate the performance of Multiscale Error Diffusion algorithm, I selected some classical images such as *grayscale.raw*, *lena.raw*, and *baboon.raw*, to go through all kind of halftoning techniques, including fixed-level halftoning, dithtering method, Floyd-Steinberg's error diffusion, and multiscale error diffusion algorithm. In addition, I also put some real world gray-level images, which is *textbook.raw*, as input to test the different halftoning techniques, and I calculated the MSE and PSNR for above methods. Please see **Fig 1 ~ Fig 10**

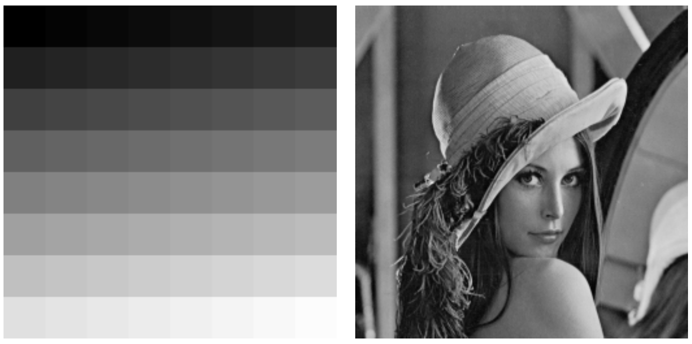


Fig 1 : Original *grayscale.raw* image (256×256) and *lena.raw* image (256×256)

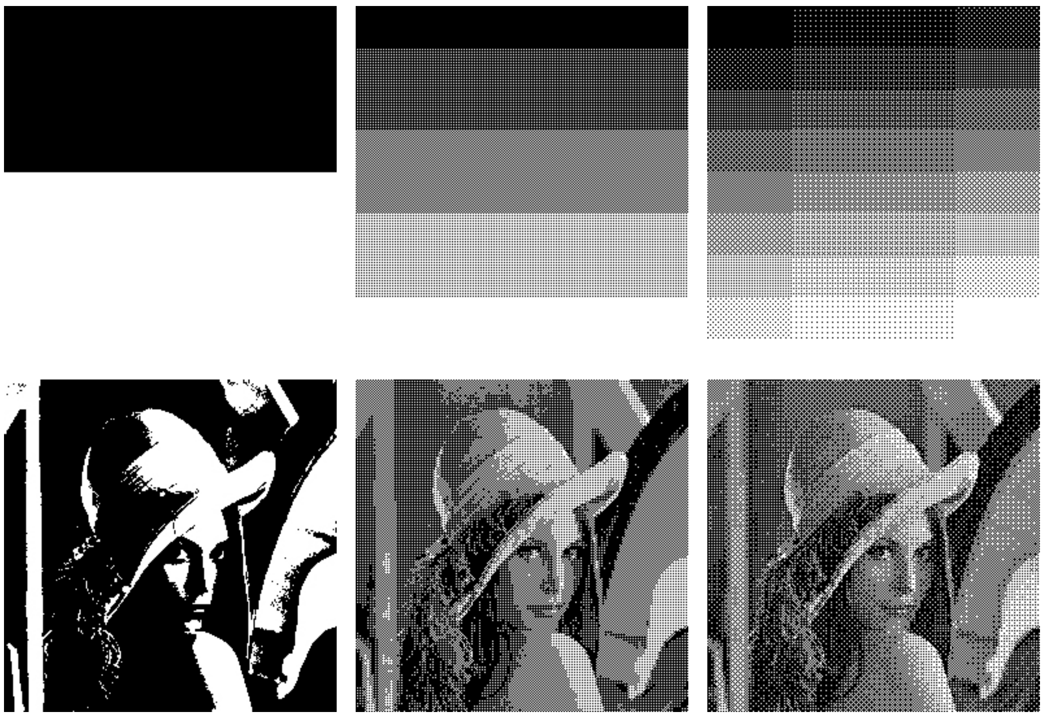


Fig 2 : *Fixed-level* method, *Dithering_I2* method, and *Dithering_I4* method

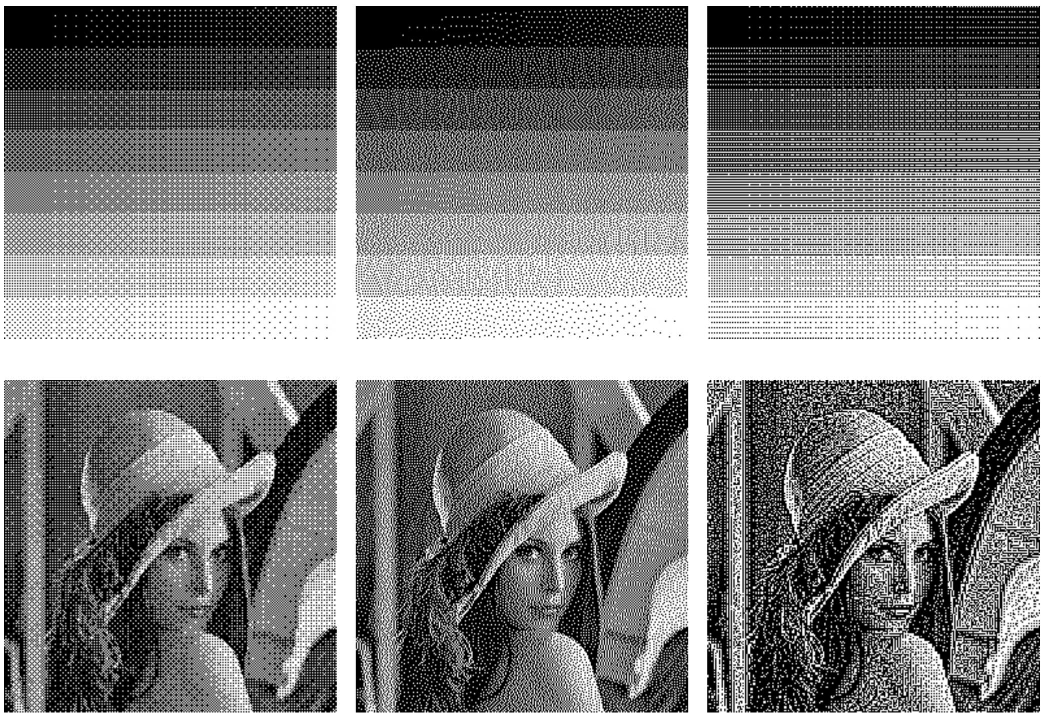


Fig 3 : *Dithering_I8* method, *F-S Error Diffusion* method, and *MED_mask1* method

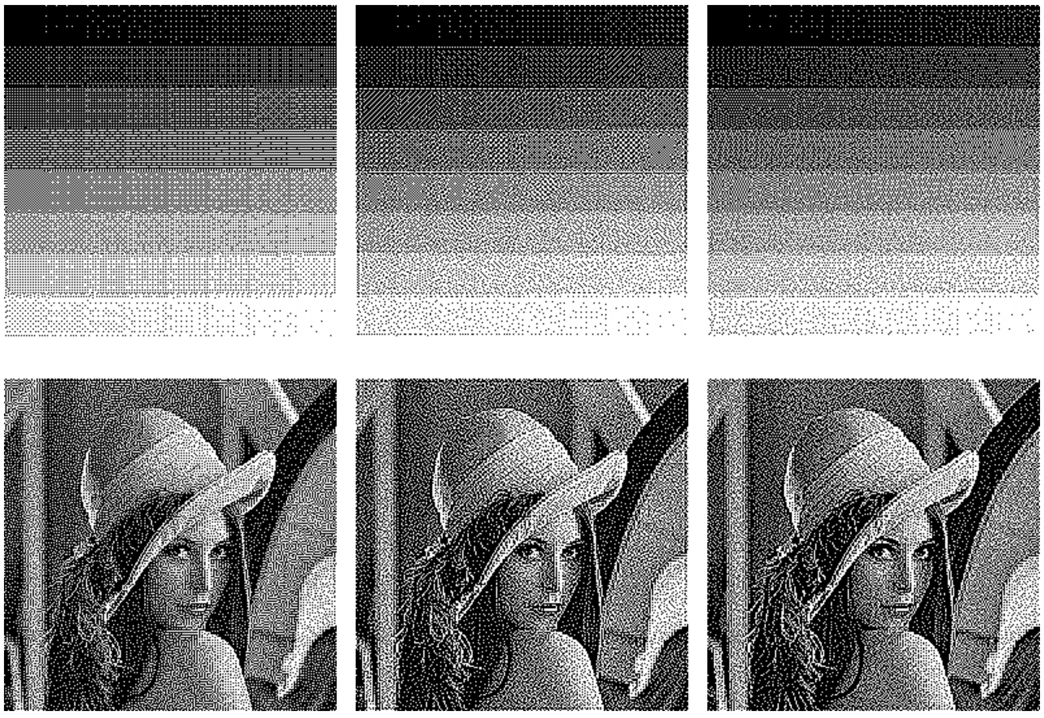


Fig 4 : *MED_mask3* method, *MED_mask5* method, and *MED_mask7* method

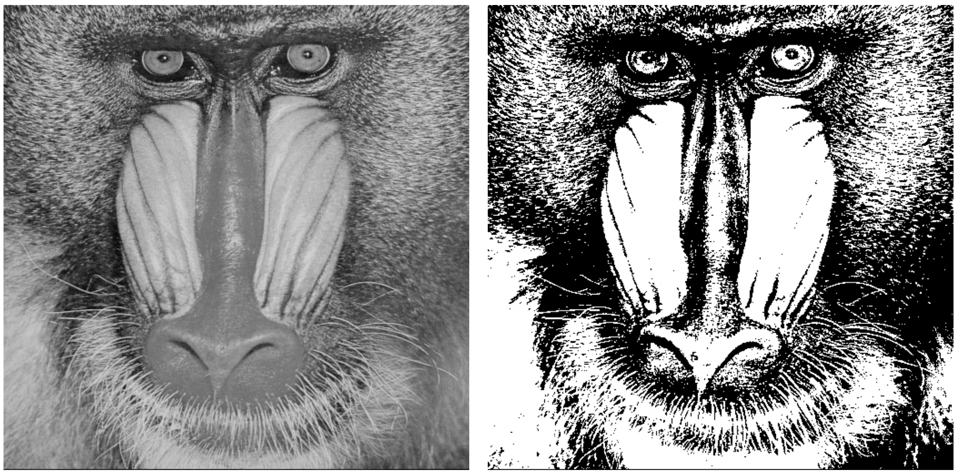


Fig 5 : Original *baboon.raw* image (512×512) and *Fixed-level* method

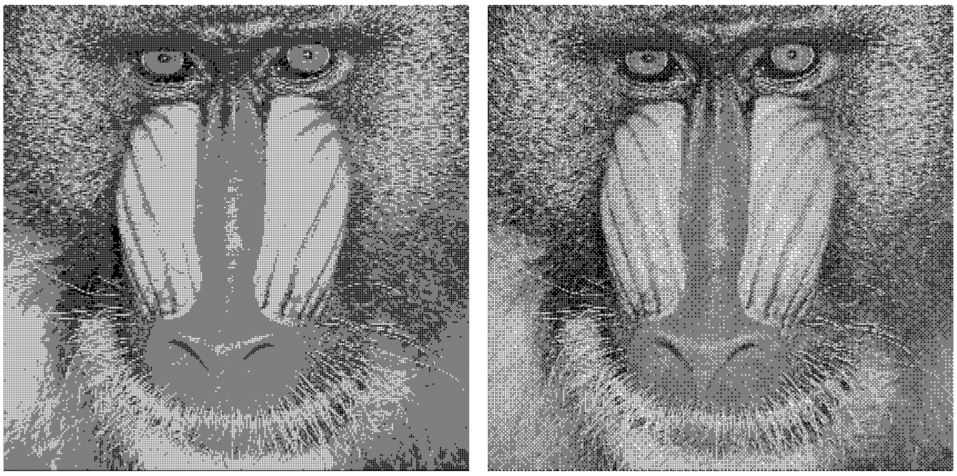


Fig 6 : *Dithering_I2* method and *Dithering_I8* method

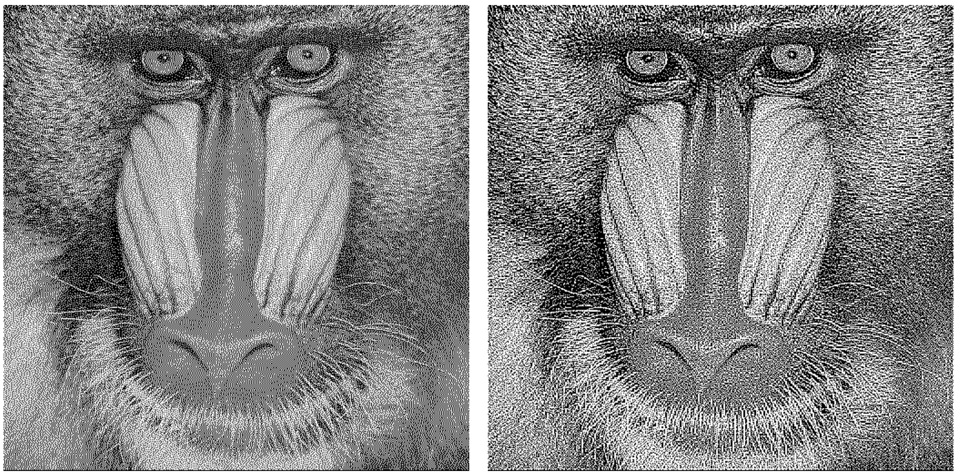


Fig 7 : *F-S Error Diffusion* method and *MED_mask7* method

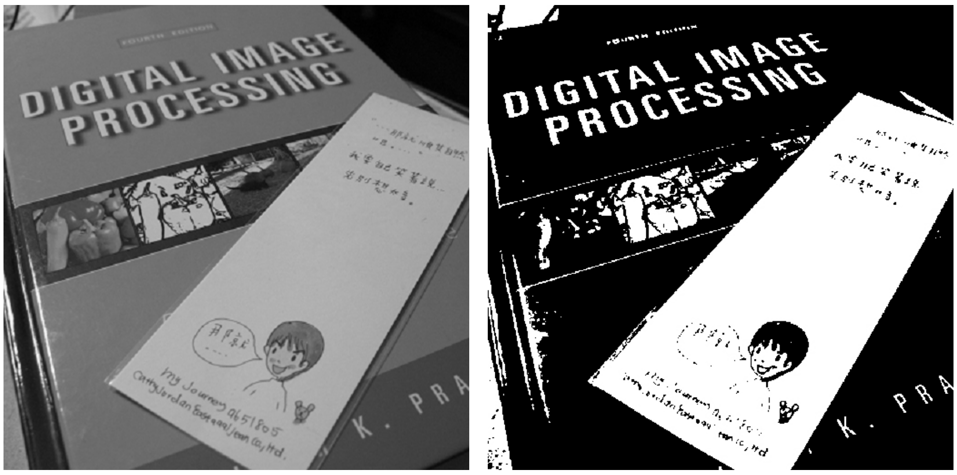


Fig 8 : Original *textbook.raw* image (512×512) and *Fixed-level* method

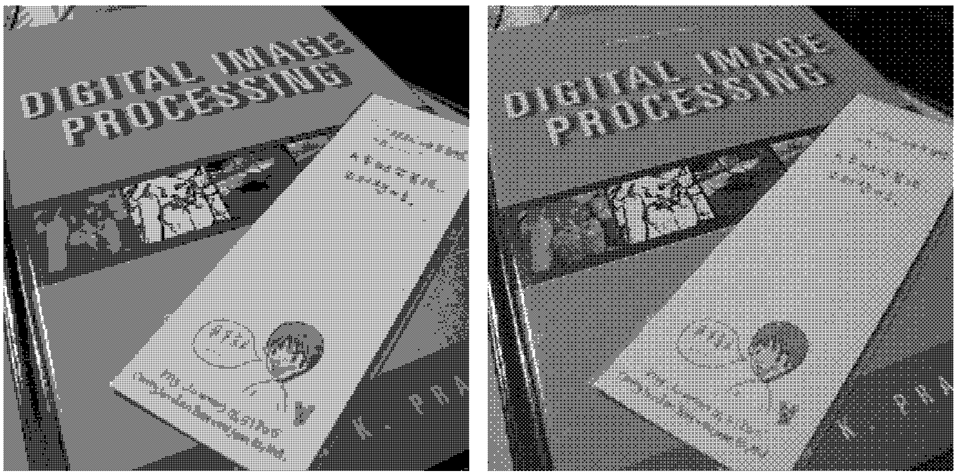


Fig 9 : *Dithering_I2* method and *Dithering_I8* method

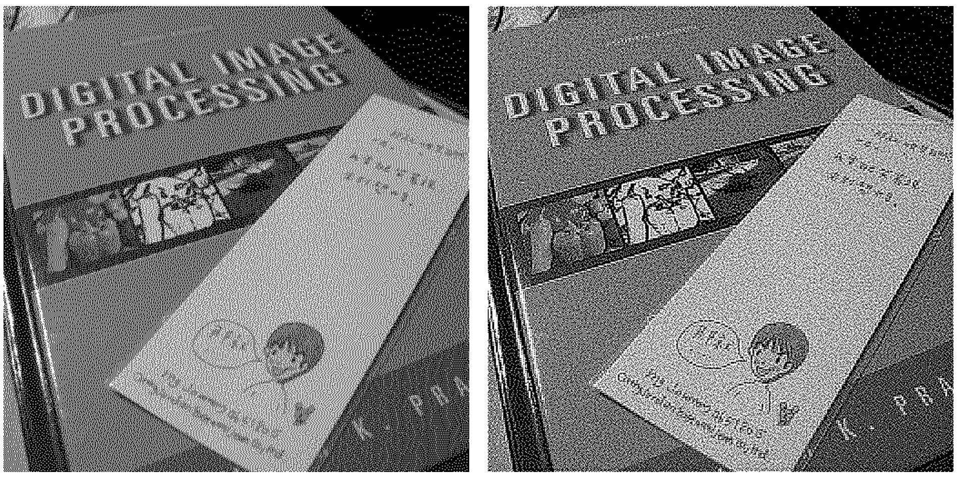
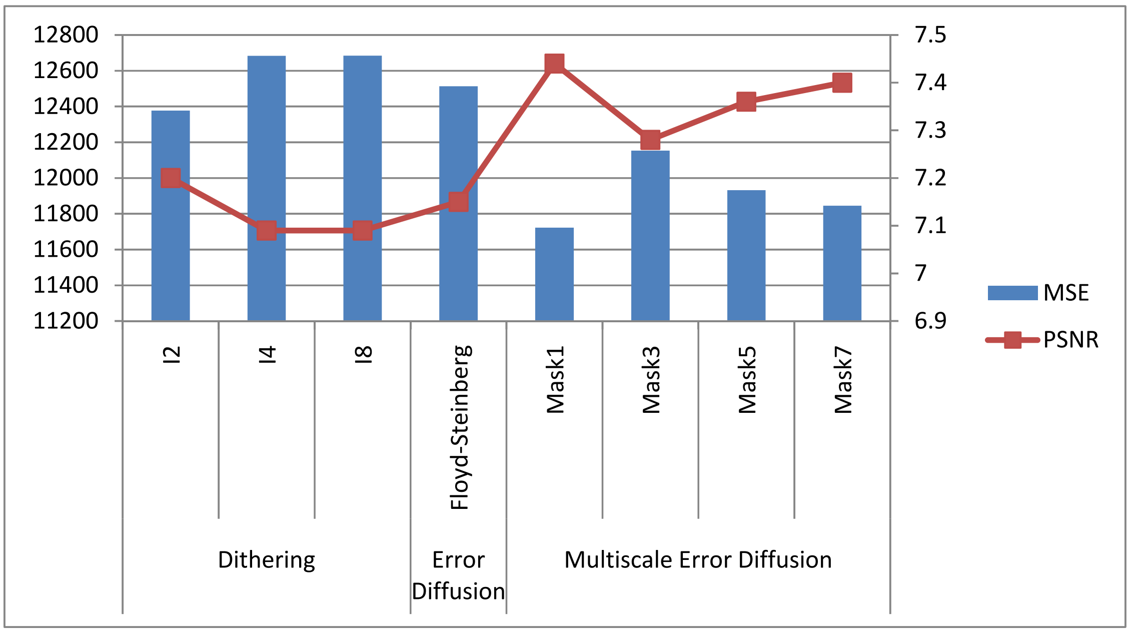


Fig 10 : *F-S Error Diffusion* method and *MED_mask7* method

VI. Discussion

MSE and PSNR value of different halftoning techniques :

	Dithering			**Error Diffusion**	**Multiscale Error Diffusion**			
	I2	**I4**	**I8**	**Floyd-Steinberg**	**Mask1**	**Mask3**	**Mask5**	**Mask7**
MSE	*12377.24*	*12683.2*	*12683.9*	*12513.77*	*11722.52*	*12152.65*	*11931.54*	*11845.75*
PSNR	*7.2*	*7.09*	*7.09*	*7.15*	*7.44*	*7.28*	*7.36*	*7.4*



What makes the MED algorithm perform a better result than other methods? The concept of MED algorithm is simple. The white pixels are given in output image based on the "Maximum Intensity Guidance" method, which is not a predetermined method. The method acted just like a human painter. We usually paint a picture from the most important part, and then the minor parts and so on. In MED algorithm, the most important part means that the brightest pixel in the bright region, which is determined by "Maximum Intensity Guidance" method. We jump from one pixel location to another, which may be spatially quite far away. This is the biggest difference from other halftoning methods.

As you can see, MED algorithm performs very good results at the *grayscale.raw* image and *lena.raw* image. By taking a careful look at the results image, we can find that the diffusion filter will affect the result, which is that the larger filter size can perform the better results. However, when we calculate the MSE and PSNR, we found that the MED_mask1 has smaller MSE (or larger PSNR) than MED_mask3, MED_mask5, and MED_mask7. The reason is that the MED_mask1 didn't diffuse the error to the neighboring pixels actually, so it acts a little like fixed-level halftoning. Because of this, the MSE of MED_mask1 is a little small than other MED methods. However, the MED_mask1 performs worst results in all MED methods.

By taking a careful look of all output images, compared with the F-S Error Diffusion method and the Multiscale Error Diffusion method, we can tell that the Multiscale Error Diffusion method can provide more detail than F-S Error Diffusion method. For example, the hair part of *lena.raw* image and the beard part of *baboon.raw* image are clearer which are generated by Multiscale Error Diffusion method. Besides, the title characters, which is "DIGITAL IMAGE PROCESSING", of *textbook.raw* image and the bookmarker boundary of *textbook.raw* image are preserved more detail in the result of Multiscale Error Diffusion method.

VII. Improvement of Multiscale Error Diffusion Algorithm

1. Block-based MED without Compensation

After I implemented and performed the Multiscale Error Diffusion algorithm, I found that the computation time of this method is much slower than other halftoning techniques. The reason is that the process of this algorithm is exhausted by going through the quadtree. At each iteration, we need to compare the intensity of specific four regions in every level, and we need to build the new quadtree and update the elements of every level. It is easy to see that the complexity of our algorithm is bounded by $O(N^2 \log N)$. For comparison, most existing methods are $O(N^2)$.

Because the computation time will increase greatly when the level number of quadtree increase, I tried to divide the original image into some blocks. I applied each block for Multiscale Error Diffusion algorithm and then combine the blocks into the whole output image. Although, the complexity of this block-based algorithm is also $O(N^2 \log N)$, we can reduce the program executing time greatly by this Block-based MED algorithm when the N usually not quiet big compared with N is equal to million or billion. In N less than 1024 cases, the block-based algorithm can perform a quiet impressive result than original algorithm.

➢ Algorithm

The Block-based Multiscale Error Diffusion algorithm is based on Multiscale Error Diffusion algorithm, so the key concept and the implement detail are all the same. There are only some difference, which is dividing the image into blocks. First, we must divide the original image into several blocks. Second, we run the original Multiscale Error Diffusion algorithm to each block. Finally, we combine all the blocks binary output into a whole binary output image, which size is equal to the original input image. In this Block-based algorithm implement, I use the 7×7 diffusion filter just like I used before.

➢ Results

As you can see, the block size will affect the result especially for small block size, which are less or equal four. Besides, we can easily found that there is a white grid line problem when block size greater than or equal to eight. The cause of this problem will discuss in following section. If the block size equal to the original image size, it performs just like original Multiscale Error Diffusion algorithm. Please see **Fig 11 ~ Fig 16**

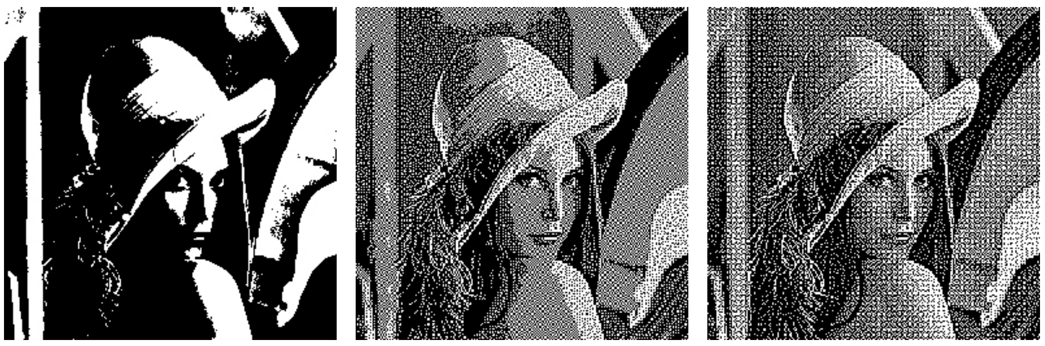


Fig 11 : *Lena.raw* image (256×256) with *Block-based MED* (Block size = *1*, *2*, *4*)

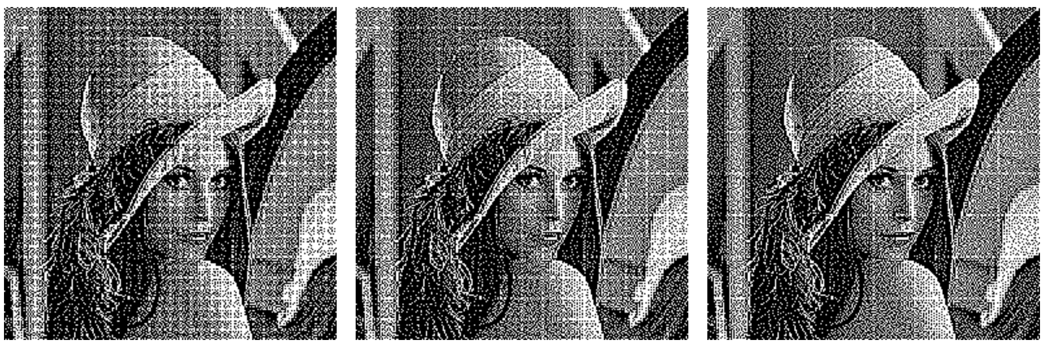


Fig 12 : *Lena.raw* image (256×256) with *Block-based MED* (Block size = *8*, *16*, *32*)

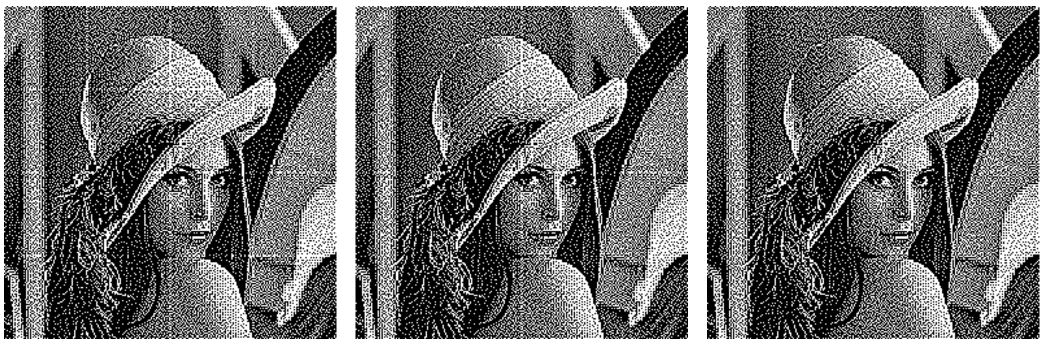


Fig 13 : *Lena.raw* image (256×256) with *Block-based MED* (Block size = *64*, *128*, *256*)

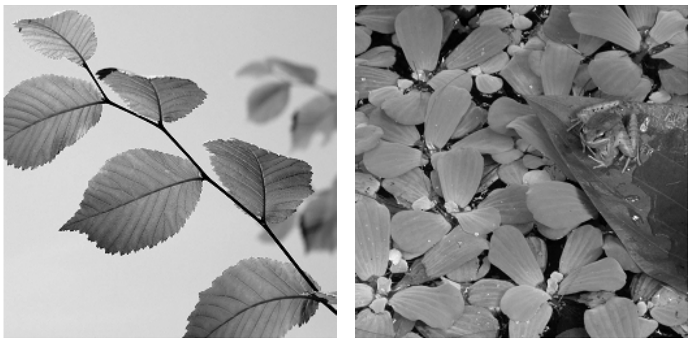


Fig 14 : Original *Leaf.raw* and original *forg.raw* image (256×256)

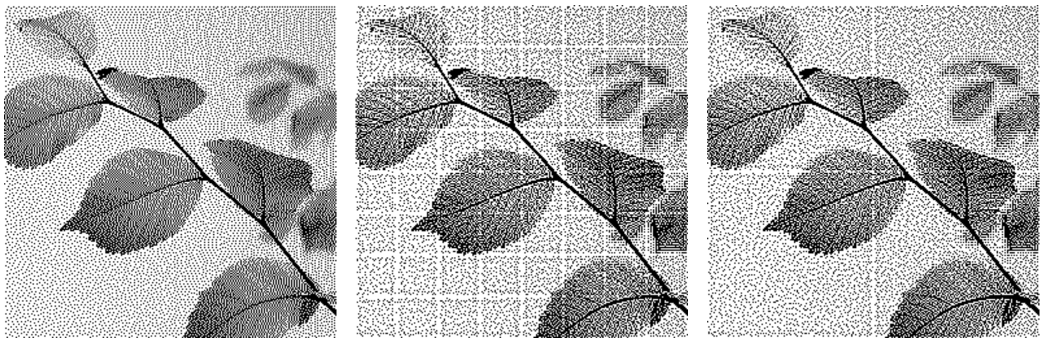


Fig 15 : *FS-Error Diffusion* and *Block-based MED* (Block size = *32*, *128*)

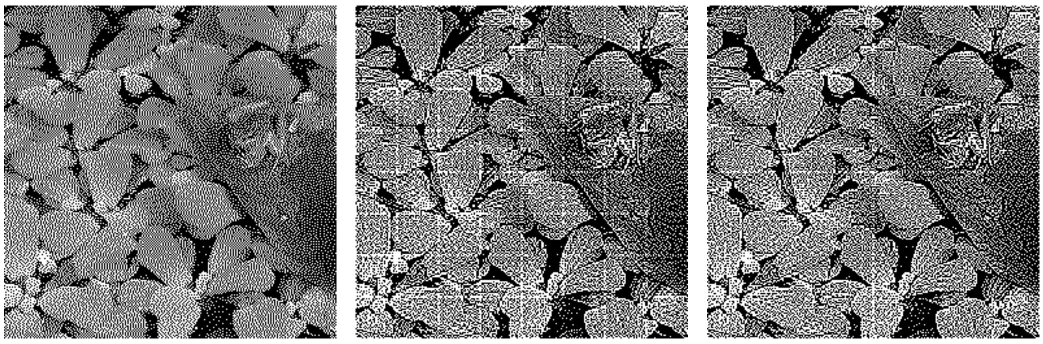
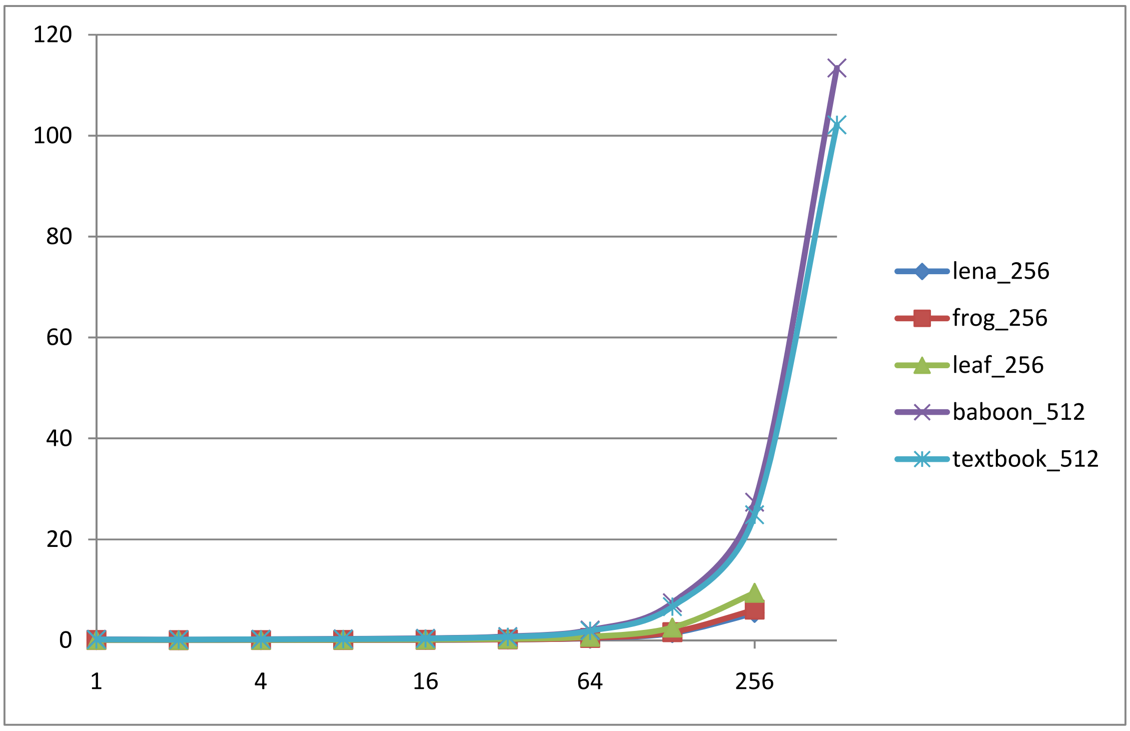


Fig 16 : *FS-Error Diffusion* and *Block-based MED* (Block size = *32*, *128*)

➢ Discussion

Execution time of Block-based MED with different block size :

		Block Size									
	Execution Time (Second)	1	2	4	8	16	32	64	128	256	512
Image Name	*lena_256*	*0.043*	*0.03*	*0.044*	*0.053*	*0.077*	*0.154*	*0.403*	*1.417*	*5.429*	
	frog_256	*0.058*	*0.037*	*0.05*	*0.063*	*0.089*	*0.175*	*0.458*	*1.616*	*6.093*	
	leaf_256	*0.069*	*0.052*	*0.068*	*0.102*	*0.141*	*0.262*	*0.699*	*2.529*	*9.41*	
	baboon_512	*0.203*	*0.121*	*0.209*	*0.278*	*0.402*	*0.759*	*2.042*	*7.435*	*27.376*	*113.419*
	textbook_512	*0.158*	*0.137*	*0.192*	*0.251*	*0.362*	*0.689*	*1.877*	*6.68*	*24.884*	*102.14*



As you can see from the result output images, if we assign the block size to 1, the block-based MED algorithm will act just like fixed-level halftoning. Moreover, if we assign block size to 2, which means that one block contain four pixels, the output image is very interesting, because the output result is a little like Dithering method with I_2 matrix, which is also contain four pixels in the I_2 matrix. In addition, if we set the block size equal to the image size, it means that we apply the original MED algorithm without dividing the input image into blocks. However, there are some white lines in the boundary of the block edge pixels when we apply other block size in this algorithm. The reason is that the original diffusion filter only considers the error diffusion of specific pixel, but it didn't consider that boundary pixels which are accepted the error from its neighbors. For example, we assume there is a worst case that all the neighbors of a boundary pixel diffuse errors to it from all directions, and then this pixel will decrease only $\frac{1}{12}+\frac{2}{12}+\frac{1}{12}+\frac{2}{8}+\frac{2}{8}=\frac{10}{12}$, not 1. In addition, the worst case in corner pixel only decreases $\frac{1}{12}+\frac{2}{8}+\frac{2}{8}=\frac{5}{12}$ from its neighbors, still not 1. Because the reduced value is smaller than 1, the boundary pixels tend to be quantized to 1, and then they will produce the white lines at block edges. Only the center region pixels will accept all the error from all directions and its worst case is decreasing 1. To fix this kind of problem, we should modify the error diffusion filter for boundary pixels to compensate the over-white tendency. Please look the better improvement of the MED algorithm in next section, which is "Block-based MED with Compensation."

From the execution time data and chart, we can easily found that the execution time increase exponentially with the block size. To get a reasonable quality of output binary image and reduce the execution time, the best choice is set the block size to $\sqrt{N}$ or $2\times\sqrt{N}$. For example, if we set the block size 32 for a 256×256 image, the execution time will reduce about 75 times and if we set the block size 32 for a 512×512 image, the execution time will reduce about 150 times compared with original MED algorithm. Moreover, we can find that the image dependence character of the Multiscale Error Diffusion algorithm from the data. The execution time of *leaf_256.raw* (about 9.41 seconds for block size equal to 256) is much longer than *lena_256.raw* (about 5.42 seconds for block size equal to 256) and *frog_256.raw* (about 6.1 seconds for block size equal to 256), because the *leaf_256.raw* image is brighter than other images. Because of this, program must distribute more white dots than other images, and then the execution time will increase, of course.

2. Block-based MED with Compensation

After I implemented the Block-based Multiscale Error Diffusion algorithm, I found that there are some white grids in the output image. The cause of this problem is that the diffusion filter is not suitable for boundary region (please see the above discussion section), so we must make some compensation in the block edge region to get a better result of halftoning. I thought and implemented three methods to do this compensation, which are (1) add random noise in block edge region, (2) post-processing by using 1-D median filter and (3) apply suitable diffusion filters for block-based MED algorithm. For the convenience, the following implements all use the 3×3 diffusion filter.

➢ Algorithm

1) Add Random Noise

This method is very simple. Because there are some over-white lines in the block edges, I decided to add some random black noise into the block edges. In boundary region, if the pixel will be quantized to 1, the program will turn it into 0 with probability of 0.2. How did I decide the 0.2 probability? I used try and error method (from 0.5, 0.4… to 0.1), and then picked out the parameter which can generate the best results.

2) Post-processing by Using 1-D Median Filter

Because the block edges are much whiter than center region, I implement a 1-D median filter with size 5 to go through the block boundary region only. The reason I applied the median filter is that I tried to eliminate the white line by using its neighboring pixels and the median filter can smooth the difference of desired regions.

3) Apply Suitable Diffusion Filters

The cause of the white grid line is that the boundary diffusion filter value is not suitable for edge pixels. The original 3×3 diffusion filters are

$$H_{center} = \frac{1}{12}\begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad H_{corner} = \frac{1}{5}\begin{bmatrix} 0 & 0 & 0 \\ 0 & -5 & 2 \\ 0 & 2 & 1 \end{bmatrix} \quad H_{side} = \frac{1}{8}\begin{bmatrix} 0 & 0 & 0 \\ 2 & -8 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

I changed the diffusion filters into

$$H'_{center} = \frac{1}{12}\begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad H'_{corner} = \frac{1}{12}\begin{bmatrix} 0 & 0 & 0 \\ 0 & -12 & 4 \\ 0 & 4 & 4 \end{bmatrix} \quad H'_{side} = \frac{1}{12}\begin{bmatrix} 0 & 0 & 0 \\ 4 & -12 & 4 \\ 1 & 2 & 1 \end{bmatrix}$$

With this diffusion filters, the boundary pixels have less probability to be quantized to 1, which is a white dot. Considering the error received from neighbors, we can easily know that we should enforce some specific elements of the mask and we also maintain the zero mean of the three types of diffusion filters.

➢ Results

To compare with the three compensation methods, I use *lena.raw*, *leaf.raw* and *frog.raw* (all 256×256) as input files. All the three compensation methods performed some impressive results. Please see **Fig 17 ~ Fig 21**

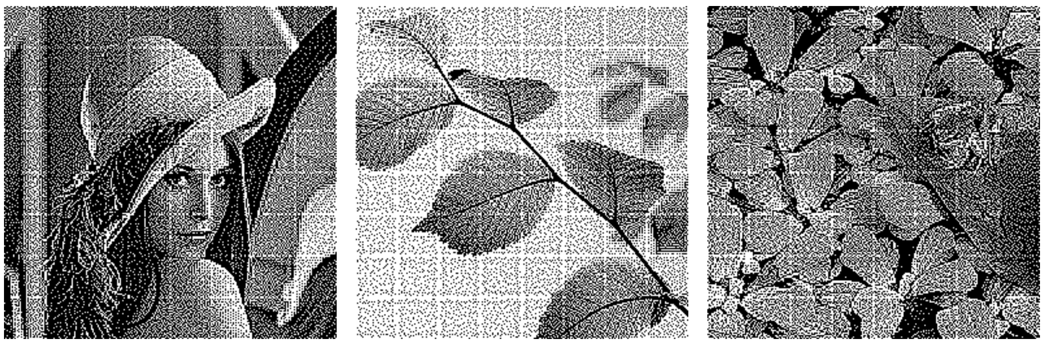


Fig 17 : *Block-based MED* (Block size = *32*) for *Lena.raw, leaf.raw and frog.raw*

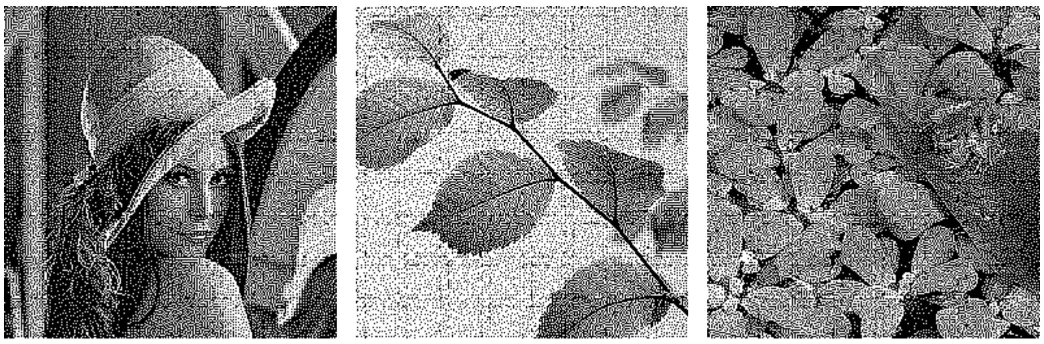


Fig 18 : *Block-based MED* (Block size = *32*) with *Random noise Compensation method*

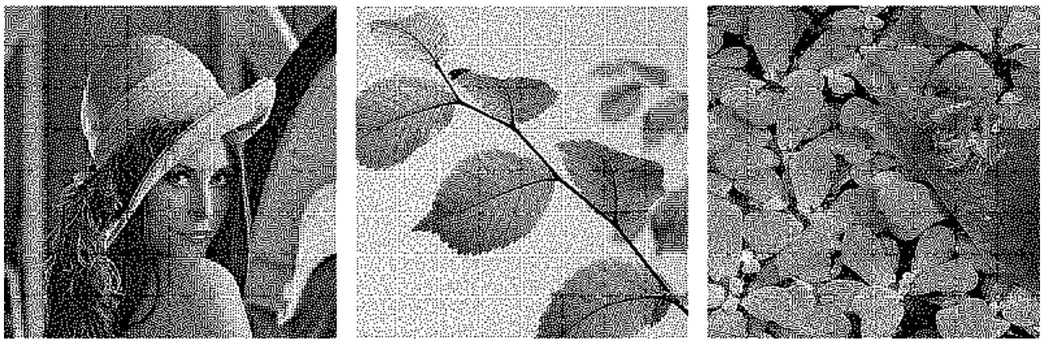


Fig 19: *Block-based MED* (Block size = *32*) with *1-D Median Compensation method*

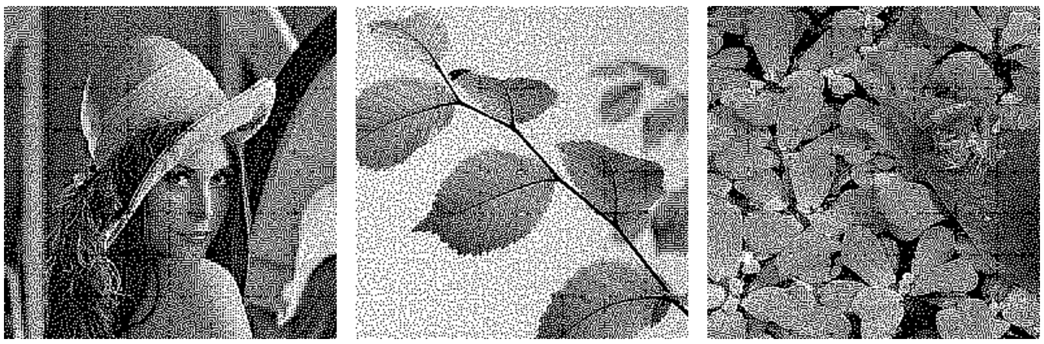


Fig 20: *Block-based MED* (Block size = *32*) with *Change Filters Compensation method*

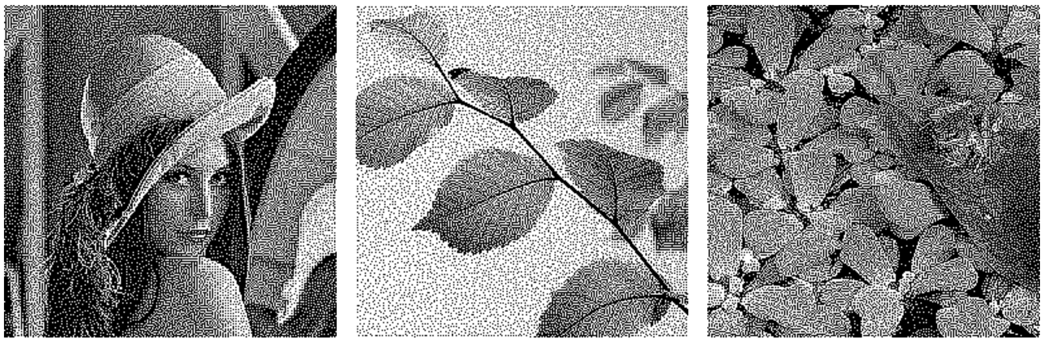


Fig 21: *Original MED method* for *lena.raw*, *leaf.raw*, and *frog.raw*

➢ Discussion

From the results, we can find that the best compensation method is Change Filters Compensation method, because this method fixed the fundamental cause of the white grid line problem, which is not suitable diffusion filter for boundary pixels. This problem existed in the original Multiscale Error Diffusion algorithm. The proportion of the boundary pixels in the whole output image is very small, so it is hard to see this problem. However, the Block-based MED algorithm will increase the proportion of the boundary pixels, so the white grid line problem is very obvious. All of the three compensation methods are very good, because they can eliminate the white grid line mostly.

After all the experimental results, using the Block-based MED algorithm (set the block size equal to 32) with Compensation method can reduce the execution time greatly and keep the quality of image detail.

REFERENCES

[1] I. Katsavounidis, C. C. J. Kuo, "A multiscale error diffusion technique for digital halftoning," *IEEE Trans. on Image Processing*, 6(3), 483-490 (1997).

[2] YH Fung, KC Lui, YH Chan "A Low-complexity High-performance Multiscale Error Diffusion Technique for Digital Halftoning," *Journal of Electronic Imaging*, 2007 – Citeseer