**National Taiwan University of Science and Technology**

**Electrical Engineering Department**

**REPORT 1**

**Multimedia Signal Processing**

Student: **LE VIET HUNG**

ID Number: **M10607812**

****

October 23, 2017

Contents

1. Problem statement …………………………………………………………………………… 3

2. Proposed solution ……………………………………………………………………………. 3

2.1 Deterministic thresholding ………………………………………………………….. 3

a) Title …………………………………………………………………………… 3

b) Code …………………………………………………………………………... 4

c) Result ……………………………………………………………………......... 4

d) Discussion …………………………………………………………….............. 5

2.2 Ordered thresholding …………………………………………………………........... 5

a) Title …………………………………………………………………………… 5

b) Code …………………………………………………………………………... 9

c) Result …………………………………………………………………….........10

d) Discussion ……………………………………………………………........... 11

2.3 Error diffusion …………………………………………………………....................12

a) Title ………………………………………………………………………… 12

b) Code …………………………………………………………………………..14

c) Result …………………………………………………………………….........18

d) Discussion ……………………………………………………………............ 18

3. References ...........…………………………………………………………………………... 18

1. **Problem statement**

* **Point Process – Ordered Dithering using the Classical-4 && Bayer-5 Dither Array**

Write an algorithm to convert the Gray Scale Image (0-255 Range) to Binary Image (0-1 Range) using the mentioned dither array.

* **Neighborhood Process – Error Diffusion**

In error-diffusion, three kernels are widely used Stucki (1981), Jarvis (1976), and Floyd-Steinberg (1975).

Write an algorithm to convert the Gray Scale Image (0-255 Range) to Binary Image (0-1 Range) based on the mentioned error diffusion kernels.

1. **Proposed solution**

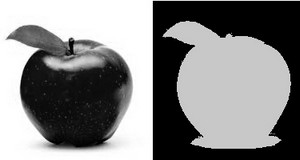
There are many methods to convert the Gray Scale Image to Binary Image such as ordered dithering, error diffusion, thresholding and so on. The simplest way is “Thresholding” which differentiates the pixels we are interested in from the rest.

* 1. **Deterministic thresholding**

1. **Title**

In the first part, I am going to apply the “Deterministic Thresholding” method to convert a grayscale image to binary image. One application example of thresholding is to separate out regions of an image corresponding the objects which we want to analyze.

Thresholding method differentiates the pixels we are interested in from the rest (which will eventually be rejected), we perform a comparison of each pixel intensity value with respect to a threshold. Once we have separated properly the important pixels, we can set them with a determined value to identify them. For example, we can assign them a value of 0 (black), 255 (white) or any value that suits our needs.

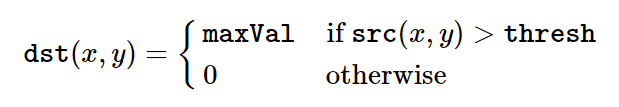


**Figure 1.** Applying thresholding method to differentiate the image of an apple

There are 5 types of thresholding operations:

* Threshold Binary
* Threshold Binary and Inverted
* Truncate
* Threshold to Zero
* Threshold to Zero and Inverted

For easier demonstration purpose, I would like to use the Threshold Binary operation applying to Lena image file. The Threshold Binary operation can be easily expressed as:



Therefore, if the intensity of the pixel src(x,y) is higher than thresh, then the new pixel intensity is set to a MaxVal. Otherwise, the pixels are set to 0.

1. **Code**

clc

clear all

%Read the image and change to gray scale image

I = imread('lena.JPG');

G = rgb2gray(I);

%Show the image

figure

imshow(G);

title('Grayscale Image')

figure

imshow( G > 128)

title('Binary Image with threshold 128')

1. **Result**



**Figure 2.** Binary image with thresholding 128

1. **Discussion**

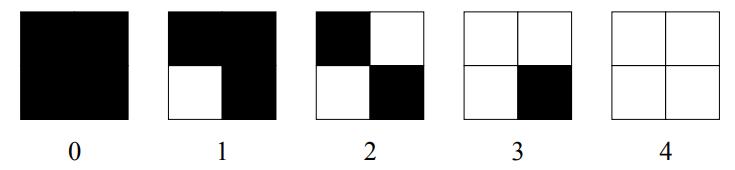
We should not use the threshold because the quality of image does not look good. The picture is unreadable and looks terrible. Therefore, we will move on to the concept of Dithering.

* 1. **Ordered Dithering**

1. **Title**

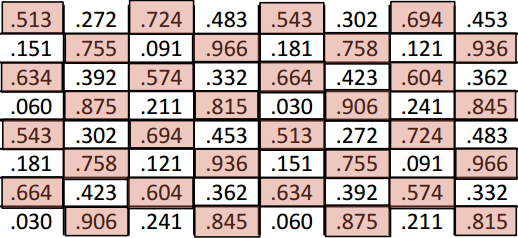
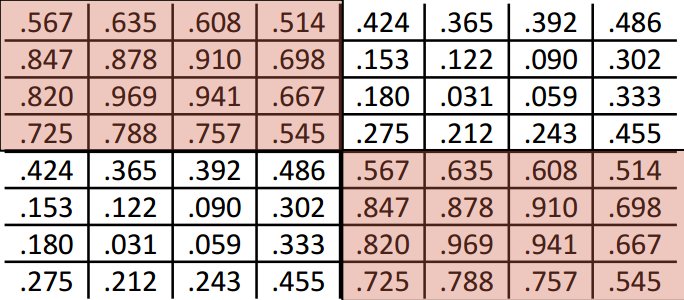
We will consider two dithering methods that are applied widely: Ordered dither and error diffusion dither.

The human visual system usually average a region around a pixel instead of sensing each pixel individually, we can create the illusion of many gray levels in a binary image in actuality only contains two gray levels. For example, using 2 x 2 binary pixel grids, we can represent 5 different “effective” intensity levels, as illustrated in Figure 3.



**Figure 3.** Five different patterns of 2 x 2 binary pixel grids.

In dithering, we replace the blocks of original image with these types of binary grid patterns. Ordered dithering consists of comparing blocks of the original image to a 2-D grid of thresholds called a dither pattern. In the report, I will take classical-4 and bayer-5 dither arrays as an example of dither pattern.



**Figure 4**. Classical-4 dither array and bayer-5 dither array

The values in the dither matrix are floating-fixed numbers, but are typically different from each other. Some decorrelation from the quantization error is achieved because the threshold changes between adjacent pixels.

**Methods:**

At the beginning, the Bayer-5 array and Classical-4 are floating-point matrices, so we need to multiply the whole matrices to 255 and round them into integer matrices, which mean that the matrix consists of all integer values.

%Create bayer array

bayer=[.513 .272 .724 .483 .543 .302 .694 .453;

.151 .755 .091 .966 .181 .758 .121 .936;

.634 .392 .574 .332 .664 .423 .604 .362;

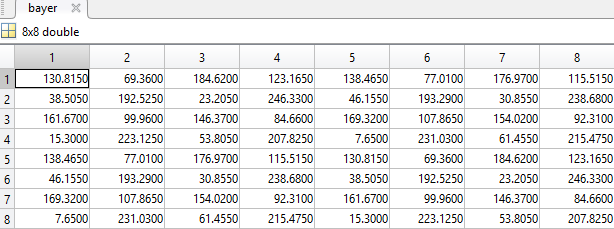
.060 .875 .211 .815 .030 .906 .241 .845;

.543 .302 .694 .453 .513 .272 .724 .483;

.181 .758 .121 .936 .151 .755 .091 .966;

.664 .423 .604 .362 .634 .392 .574 .332;

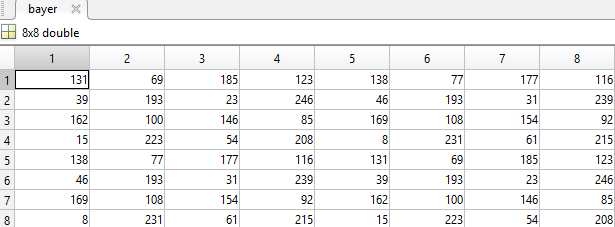
.030 .906 .241 .845 .060 .875 .211 .815]**\*255**;



**Figure 5**. Bayer-5 matrix

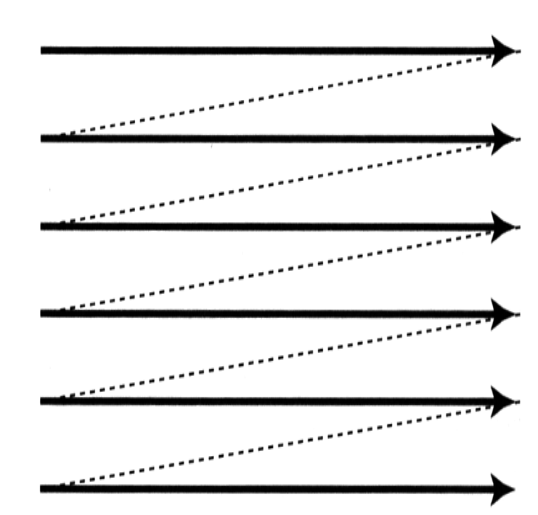
%Round the values

bayer = round(bayer);



**Figure 6**. Bayer-5 matrix with integer values

Because ordered dithering is using the point-process that means every pixel of original image will be compared to specified value of Dither array. To be easier for implementing the algorithm, I am going to use to Raster Scan technique to process all the pixel of original image.



**Figure 7**. Raster scan – processing path

I will use 2 for-loop the access every pixel from left to right, from top to bottom.

for R=1:row

for C=1:col

if G(R,C) > mask\_bayer(R,C)

Bayer\_Image(R,C) = 255;

else

Bayer\_Image(R,C) = 0;

end

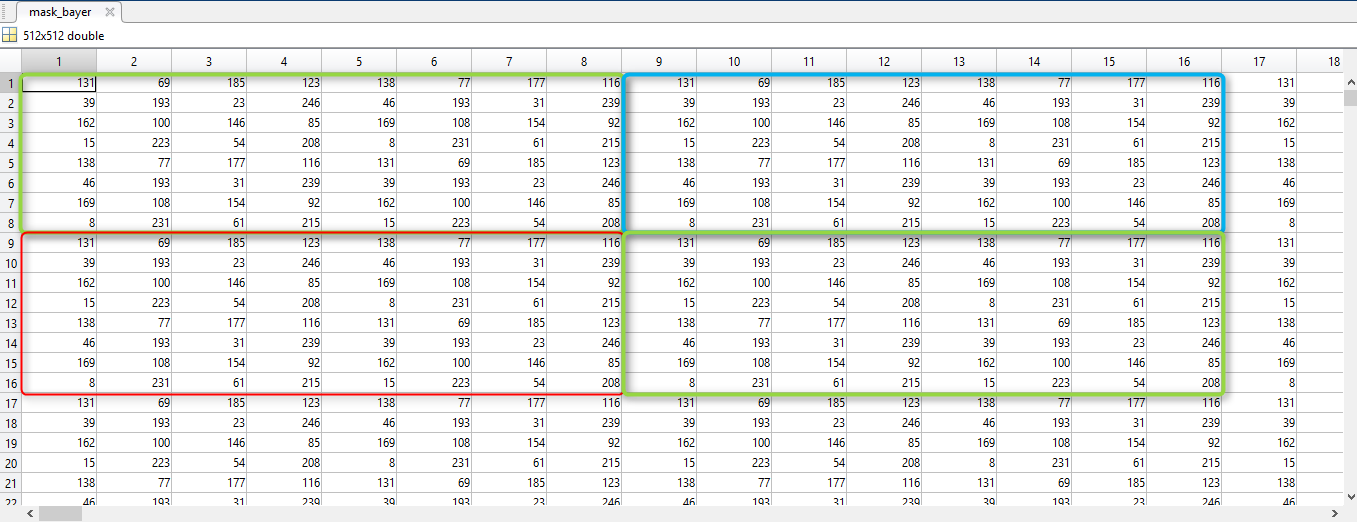
end

end

The “Lena.jpg” image has size 512 x 512 but the Classical-4 and Bayer-5 Dither arrays are 8x8 matrices. The fastest way to solve the problem is repeat copies of Classical-4 and Bayer-5 arrays.

mask\_bayer=repmat(bayer,row/8,col/8);

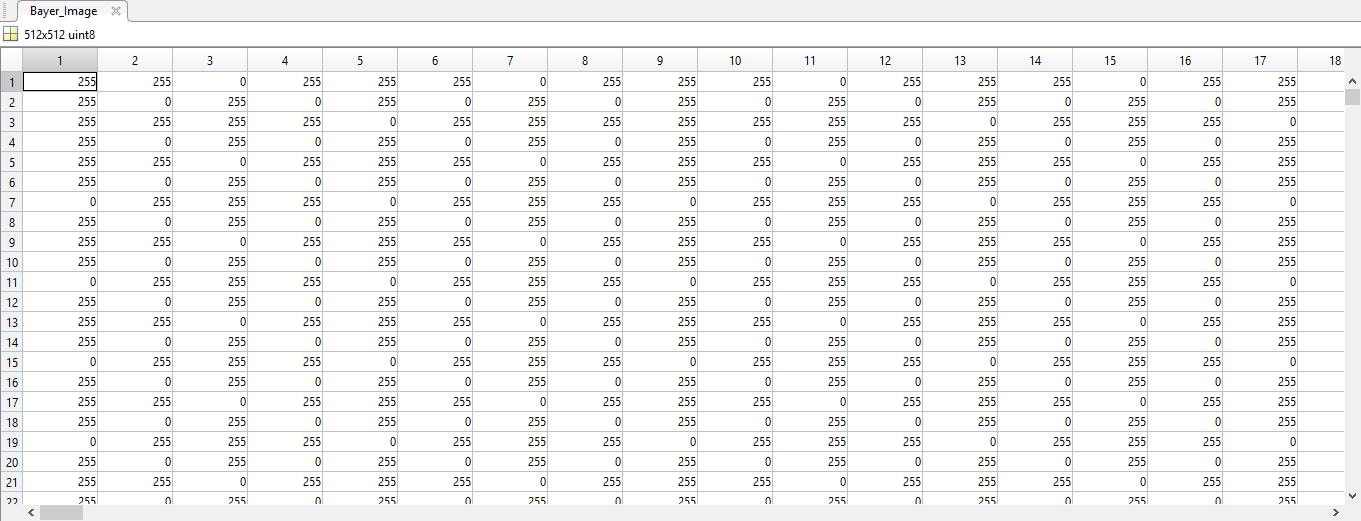
“Mask\_bayer” returns an array containing 512/8 = 64 copies of array in the row and column dimensions which means the Bayer-5 arrays will have size of 512 x 512.



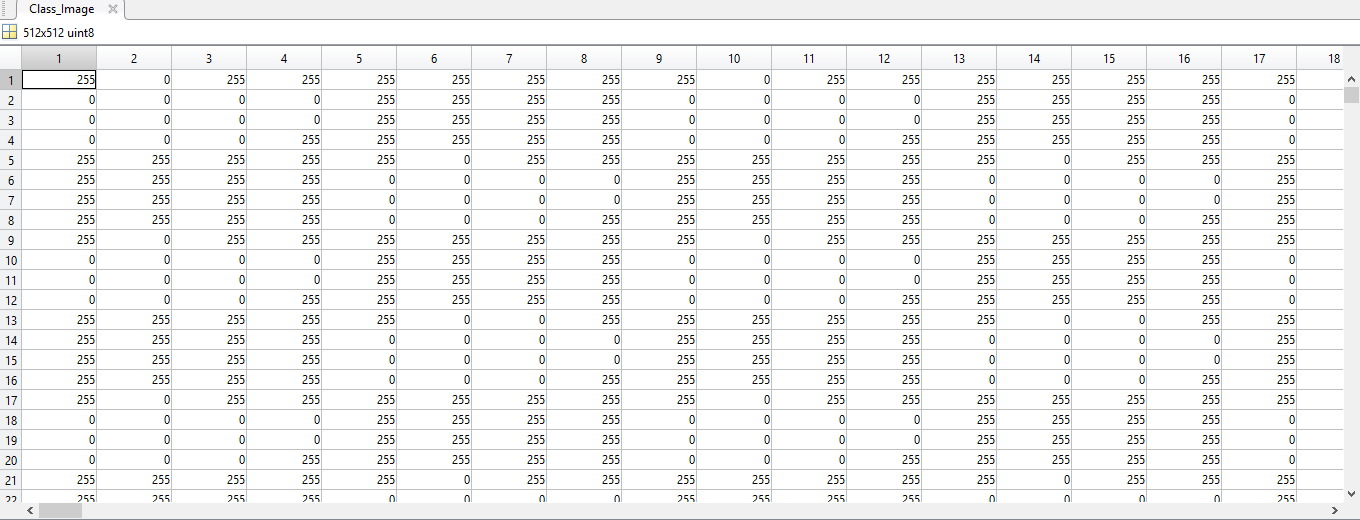
**Figure 8**. Bayer-5 arrays after duplicating matrix

The crucial ordered dither algorithm is that comparing the gray level of a pixel to a level in a dither matrix, depending on this comparison, the pixel is set to either black or white. In the real case, if gray level of a pixel is larger than value in dither matrix then the pixel is set to 255, else set to 0. The whole process is applied from left to right, from top to bottom of matrix.

Finally, we will get a new matrix including only 2 values 0 or 255. “0” means print out black ink, 255 means print out white ink.



**Figure 9.** The matrix of original image with Bayer-5 dither array



**Figure 10.** The matrix of original image with Classical-4 dither array

1. **Code**

clc

clear all

%Read the image and change to gray scale image

I = imread('lena.JPG');

G = rgb2gray(I);

%Create bayer array

bayer=[.513 .272 .724 .483 .543 .302 .694 .453;

.151 .755 .091 .966 .181 .758 .121 .936;

.634 .392 .574 .332 .664 .423 .604 .362;

.060 .875 .211 .815 .030 .906 .241 .845;

.543 .302 .694 .453 .513 .272 .724 .483;

.181 .758 .121 .936 .151 .755 .091 .966;

.664 .423 .604 .362 .634 .392 .574 .332;

.030 .906 .241 .845 .060 .875 .211 .815]\*255;

%Classical-4 Dither Array

classical = [.567 .635 .608 .514 .424 .365 .392 .486;

.847 .878 .910 .698 .153 .122 .090 .302;

.820 .969 .941 .667 .180 .031 .059 .333;

.725 .788 .757 .545 .275 .212 .243 .455;

.424 .365 .392 .486 .567 .635 .608 .514;

.153 .122 .090 .302 .847 .878 .910 .698;

.180 .031 .059 .333 .820 .969 .941 .667;

.275 .212 .243 .455 .725 .788 .757 .545]\*255;

%Round the values

bayer = round(bayer);

classical = round(classical);

%Get the value of row and column of image then assign to variable 'row',

%'col'

[row col] = size(G);

mask\_bayer=repmat(bayer,row/8,col/8);

mask\_classical=repmat(classical,row/8,col/8);

for R=1:row

for C=1:col

if G(R,C) > mask\_bayer(R,C)

Bayer\_Image(R,C) = 255;

else

Bayer\_Image(R,C) = 0;

end

end

end

for R=1:row

for C=1:col

if G(R,C) > mask\_classical(R,C)

Class\_Image(R,C) = 255;

else

Class\_Image(R,C) = 0;

end

end

end

%Show the desired image

figure

imshow(G);

title('Gray Scale Image')

figure

imshow(Bayer\_Image);

title('Ordered Dithering | Halftone Image by using Bayer-5 Dither Array')

figure

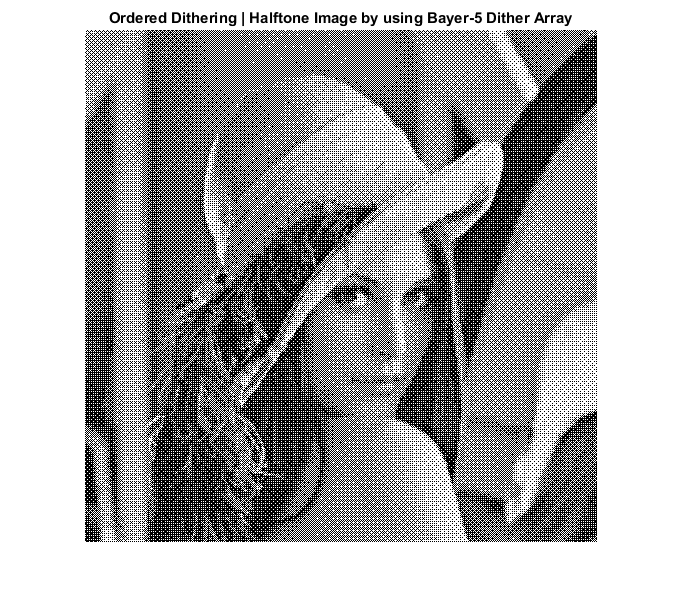
imshow(Class\_Image);

title('Ordered Dithering | Halftone Image by using Classical-4 Dither Array')

1. **Results**



**Figure 11.** Grayscale image and halftone image by using classical-4 dither array



**Figure 12.** Grayscale image and halftone image by using bayer-5 dither array

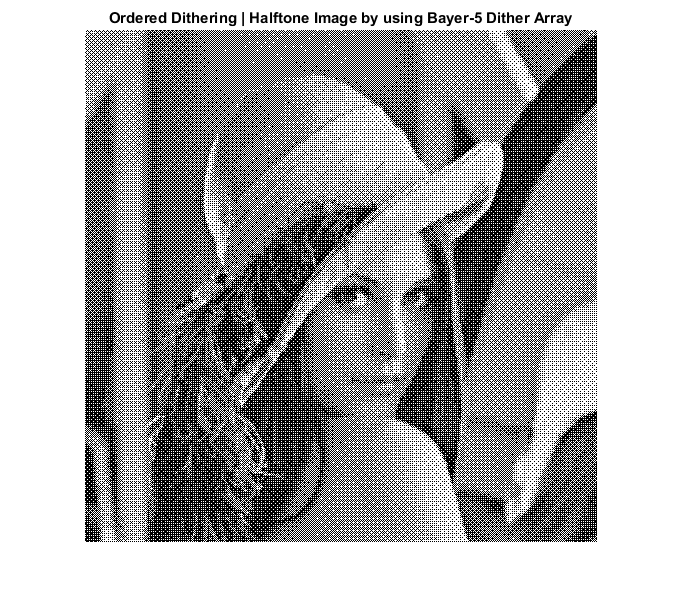
1. **Discussion**

At the beginning, I tried to convert gray scale image into binary image with simple method “Half toning” buy comparing each pixel of image to a certain thresholding value, for example 128. I finally got a very bad result due to bad distribution of numbers.



**Figure 13.** Grayscale image and halftone image with constant thresholding 128

It is clear that the halftone image created by bayer-5 dither array provides more decent detail rendition than created by classical-4 dither array. At normal viewing distances, the picture (left) is visibly noisier than the right one.



**Figure 11.** Half toning image creating by using classical-4 (left) and bayer-5 (right)

* 1. **Error Diffusion**

1. **Title**

**Error diffusion dither** algorithms spread the error (caused by quantizing a pixel) over neighboring pixels; the best known example of error diffusion dither is the "Floyd Steinberg" dither.

In error diffusion, the value of a pixel is compared to a single threshold of 50% gray. When black is 0 and white is 255, each pixel is compared to 128. The pixel is set to either black or white, based on that comparison. After outputting the dithered pixel, the algorithm calculates the difference between the source pixel and the output pixel (a simple subtraction), and then it spreads this difference (the "error") over neighboring pixels.

if (OLD(R,C) < T) %T: Thresholding = 128

NEW(R,C) = 0; %R: row, C: Column

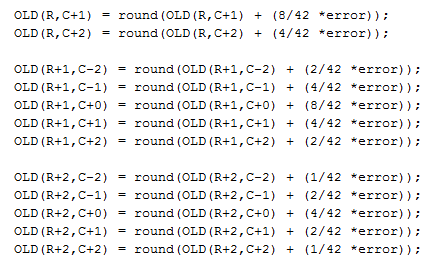
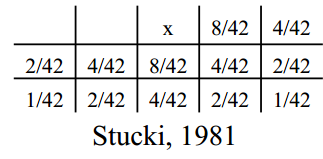
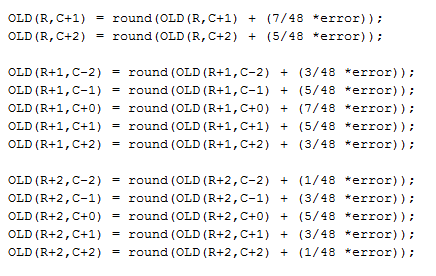
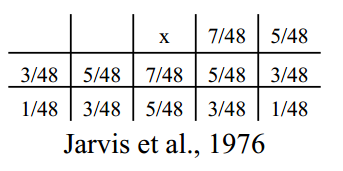
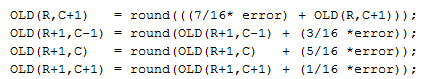
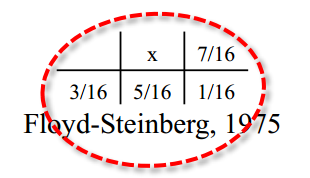
else

NEW(R,C) = 255; %OLD: the source pixel

end; %NEW: the output pixel

error = OLD(R,C)-NEW(R,C); %error: calculate the difference by simple subtraction

In other words, the algorithm changes the source image as it goes through it. The error is partitioned between pixels to the right of the current pixel and pixels below the current pixel. The "diffusion" part of the error diffusion algorithm is thus two-dimensional.



**Figure 12.** Implementing three kernels with code

1. **Code**

* Matlab Implementation of Error diffusion using Floyd's and Steinberg's

% Matlab Implementation of Error diffusion using Floyd's and Steinberg's...

clc

clear all

%Read the image and change to gray scale image

I = imread('lena.JPG');

G = rgb2gray(I);

%Declare some variables

T = 128 % Thresholding

%Apply Zeropadding to matrix

OLD = Zeropadding\_Floyd(G);

%Get the value of row and column of image then assign to variable 'row',

%'col'

[row col] = size(OLD);

NEW = zeros(size(OLD));

for R=2: (row-1)

for C=2 : (col-1)

if (OLD(R,C) < T)

NEW(R,C) = 0;

else

NEW(R,C) = 255;

end;

error = OLD(R,C) - NEW(R,C);

OLD(R,C+1) = round(((7/16\* error) + OLD(R,C+1)));

OLD(R+1,C-1) = round(OLD(R+1,C-1) + (3/16 \*error));

OLD(R+1,C) = round(OLD(R+1,C) + (5/16 \*error));

OLD(R+1,C+1) = round(OLD(R+1,C+1) + (1/16 \*error));

end

end

%De-Padding Image

NEW = DePadding\_Floyd(NEW);

%Show the desired image

figure

imshow(G);

title('Gray Scale Image')

%Show the halftoning image

figure

imshow(NEW);

title('Error diffusion using Floyd and Steinberg');

* Matlab Implementation of Error diffusion using Jarvis

% Matlab Implementation of Error diffusion using Jarvis

clc

clear all

%Read the image and change to gray scale image

I = imread('lena.JPG');

G = rgb2gray(I);

T = 128; % Threshold

%Apply Zeropadding to matrix

OLD = ZeroPadding\_Jarvis(G);

%Get the value of row and column of image then assign to variable 'row',

%'col'

[row col] = size(OLD);

NEW = zeros(size(OLD));

for R =3: (row-2)

for C = 3: (col-2)

if (OLD(R,C) < T)

NEW(R,C) = 0;

else

NEW(R,C) = 255;

end;

error = OLD(R,C) - NEW(R,C);

OLD(R,C+1) = round(OLD(R,C+1) + (7/48 \*error));

OLD(R,C+2) = round(OLD(R,C+2) + (5/48 \*error));

OLD(R+1,C-2) = round(OLD(R+1,C-2) + (3/48 \*error));

OLD(R+1,C-1) = round(OLD(R+1,C-1) + (5/48 \*error));

OLD(R+1,C+0) = round(OLD(R+1,C+0) + (7/48 \*error));

OLD(R+1,C+1) = round(OLD(R+1,C+1) + (5/48 \*error));

OLD(R+1,C+2) = round(OLD(R+1,C+2) + (3/48 \*error));

OLD(R+2,C-2) = round(OLD(R+2,C-2) + (1/48 \*error));

OLD(R+2,C-1) = round(OLD(R+2,C-1) + (3/48 \*error));

OLD(R+2,C+0) = round(OLD(R+2,C+0) + (5/48 \*error));

OLD(R+2,C+1) = round(OLD(R+2,C+1) + (3/48 \*error));

OLD(R+2,C+2) = round(OLD(R+2,C+2) + (1/48 \*error));

end

end

%DePadding Jarvis

NEW = DePadding\_Jarvis(NEW);

%Show the desired image

figure

imshow(G);

title('Gray Scale Image')

%Show the halftoning image

figure

imshow(NEW);

title('Error diffusion using Jarvis');

* Matlab Implementation of Error diffusion using Stucki

% Matlab Implementation of Error diffusion using Stucki

clc

clear all

%Read the image and change to gray scale image

I = imread('lena.JPG');

G = rgb2gray(I);

T = 128; % Threshold

%Apply Zeropadding to matrix

OLD = ZeroPadding\_Jarvis(G);

%Get the value of row and column of image then assign to variable 'row',

%'col'

[row col] = size(OLD);

NEW = zeros(size(OLD));

for R =3: (row-2)

for C = 3: (col-2)

if (OLD(R,C) < T)

NEW(R,C) = 0;

else

NEW(R,C) = 255;

end;

error = OLD(R,C) - NEW(R,C);

OLD(R,C+1) = round(OLD(R,C+1) + (8/42 \*error));

OLD(R,C+2) = round(OLD(R,C+2) + (4/42 \*error));

OLD(R+1,C-2) = round(OLD(R+1,C-2) + (2/42 \*error));

OLD(R+1,C-1) = round(OLD(R+1,C-1) + (4/42 \*error));

OLD(R+1,C+0) = round(OLD(R+1,C+0) + (8/42 \*error));

OLD(R+1,C+1) = round(OLD(R+1,C+1) + (4/42 \*error));

OLD(R+1,C+2) = round(OLD(R+1,C+2) + (2/42 \*error));

OLD(R+2,C-2) = round(OLD(R+2,C-2) + (1/42 \*error));

OLD(R+2,C-1) = round(OLD(R+2,C-1) + (2/42 \*error));

OLD(R+2,C+0) = round(OLD(R+2,C+0) + (4/42 \*error));

OLD(R+2,C+1) = round(OLD(R+2,C+1) + (2/42 \*error));

OLD(R+2,C+2) = round(OLD(R+2,C+2) + (1/42 \*error));

end

end

%DePadding Jarvis

NEW = DePadding\_Jarvis(NEW);

%Show the desired image

figure

imshow(G);

title('Gray Scale Image')

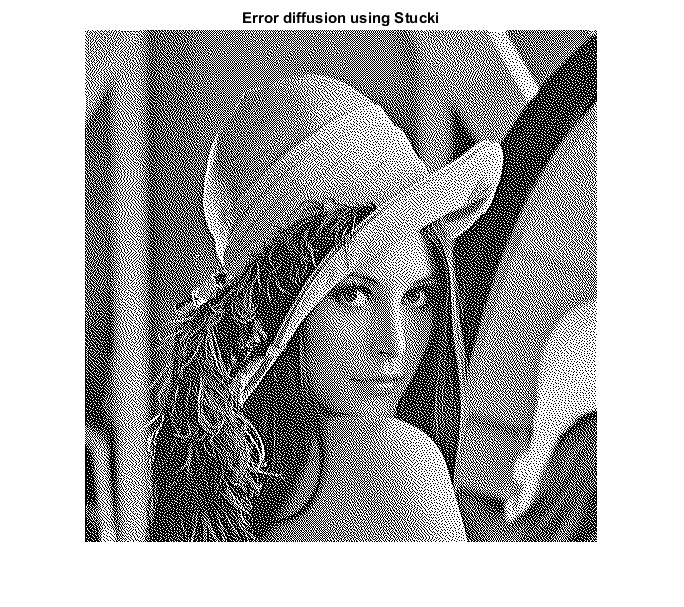
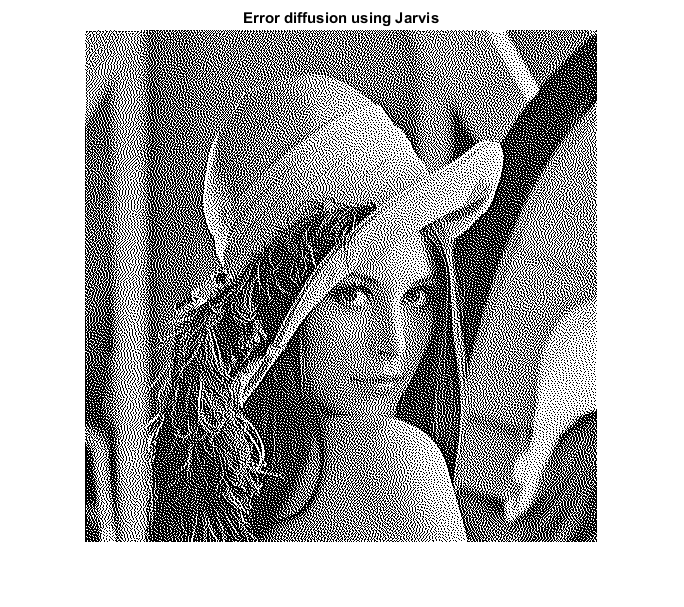
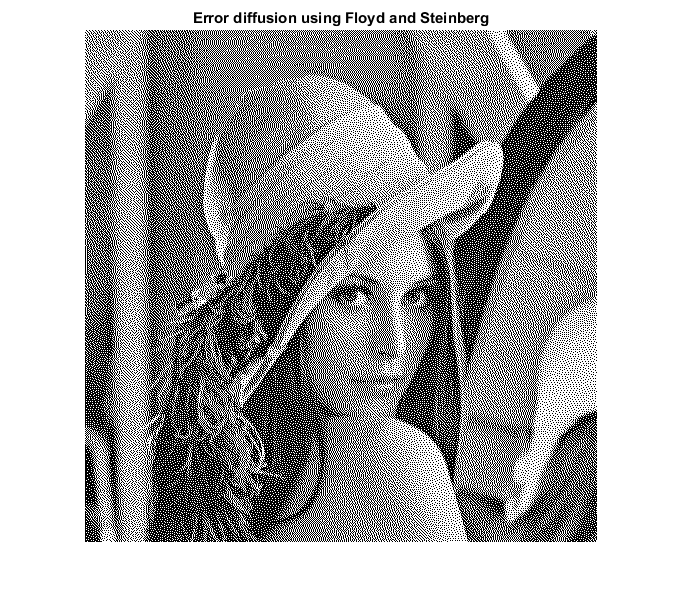
%Show the halftoning image

figure

imshow(NEW);

title('Error diffusion using Stucki');

1. **Results**



**Figure 13.** Half toning image using kernels Floyd (left), Jarvis (middle) and Stucki(right)

1. **Discussion**

After apply three kernels Floyd-Steinberg, Jarvis and Stucki, three desired images have very good visual quality.

There are many traditional approaches to assess the image quality based on error-sensitivity such as mean squared error (MSE) and peak signal-to-noise- ration (PSNR). However, these metrics has several limitations because these methods are based on pixel differences and do not take into account the spatial structure and human visual characteristics [1], so I will use the structural similarity (SSIM) as a widely used concept of image quality assessment.

|  |  |  |
| --- | --- | --- |
| STT | Kernels | SSIM Index |
| 1 | Floyd-Steinberg | 0.0289 |
| 2 | Jarvis | 0.0371 |
| 3 | Stucki | 0.0347 |

By using the SSIM index, in the “lena” image, the Jarvis kernel gives a better image quality than the others one with the highest index – 0.0371. Floyd-Steinberg kernel produces a little lower image quality than Stucki kernel. Therefore, in the “lena” picture, we can apply the Jarvis kernel to convert gray scale image into binary image with the improved image quality.

1. **References**

[1] L. L. Zhengyou Wang , Shuang Wu, Yanhui Xia, Zheng Wan and Cong Cai, "A New Image Quality Assessment Algorithm based on SSIM and Multiple Regressions," *International Journal of Signal Processing, Image Processing and Pattern Recognition,* vol. 8, pp. 221-230, 2004.

[2] http://users.ece.utexas.edu/~bevans/projects/halftoning/toolbox/

[3] https://engineering.purdue.edu/~bouman/grad-labs/Image-Halftoning/pdf/lab.pdf

[4] https://www.cs.princeton.edu/courses/archive/fall00/cs426/lectures/dither/dither.pdf