Assignment 2 - Computational Fabrication

Federico Pallotti

May 2022

1 Thresholding Method (Given)

This first naive method, implemented with a simple threshold, set at 0.5, is computationally fast, since its a pixel processing method and it does not take into account any information regarding neighbouring pixels. This first method can lead to either decent results or almost complete loss of information depending on the input image and its composition and texture.

1.1 Naive implementation

As one can see from the output images, "Lena" picture preserves some features that can make the subject of the image still recognizable. On the other hand, the "Shading texture" becomes a two-tone image, losing a lot of information.

1.2 Thresholding with noise

Adding some noise to the threshold can lead to better results. Especially for the "Shading texture", which now contains the information about the transition from black to white. Also "Lena" picture shows some improvements in the texturing and depth representation.

1.3 Computational time

To process the two images with Thresholding are 112 and 114 milliseconds, while with the noise they go up to 119 and 118 milliseconds. This method is indeed the fastest out of all the others.

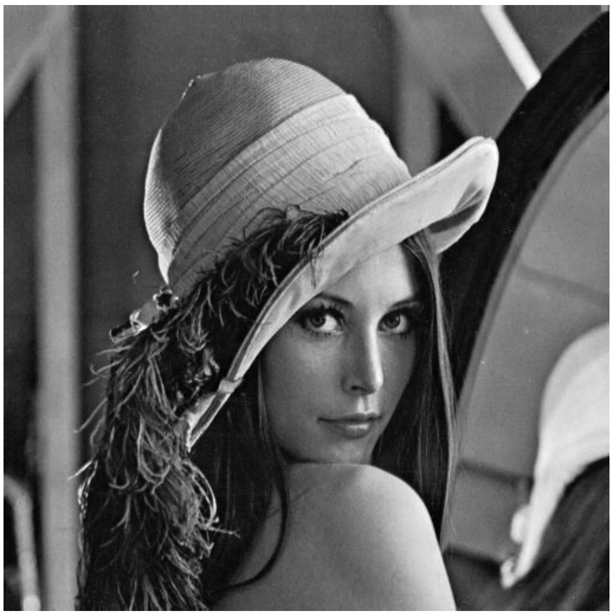
Figure 1: Original input image

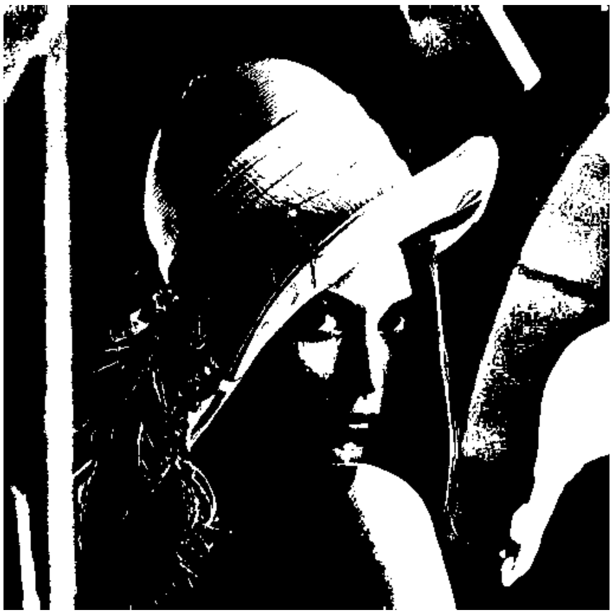
Figure 2: Thresholding method output image

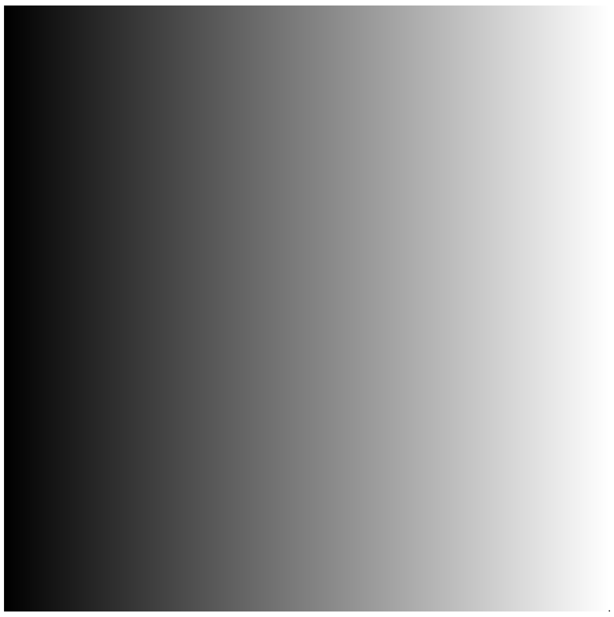
Figure 3: Original input image

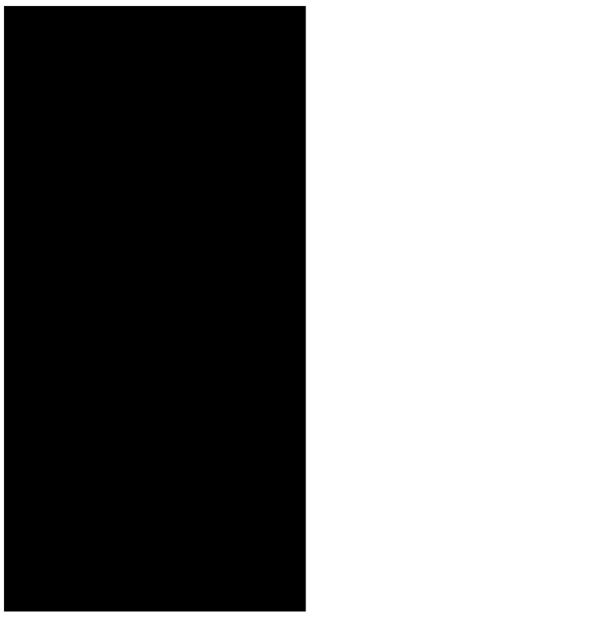
Figure 4: Thresholding method output image

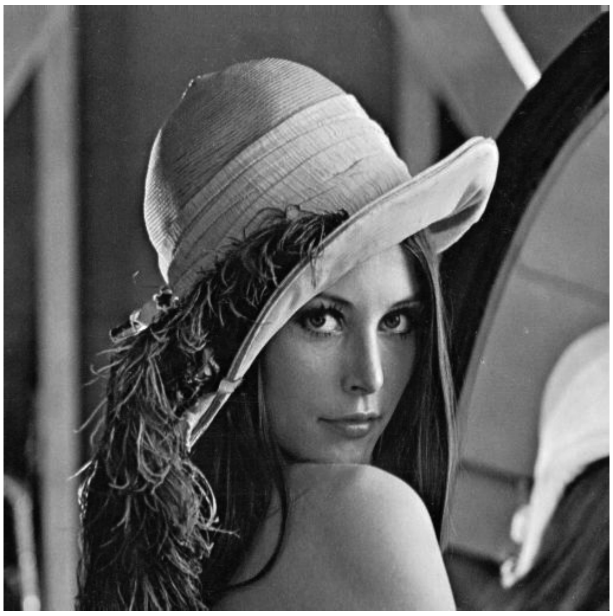
Figure 5: Original input image

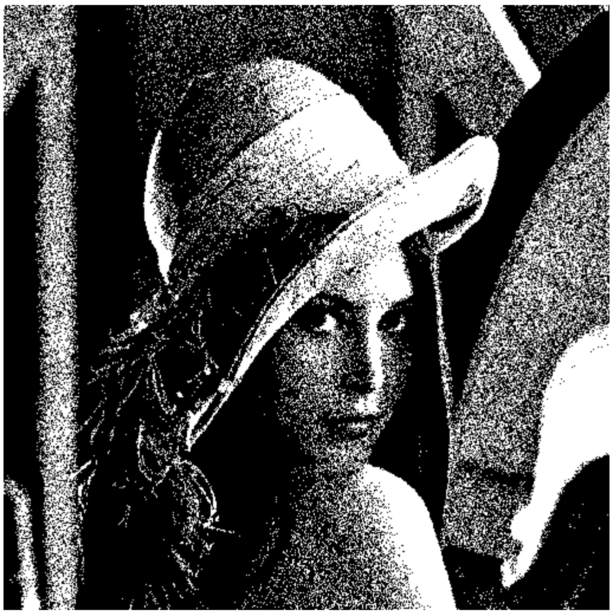
Figure 6: Thresholding with noise method output image

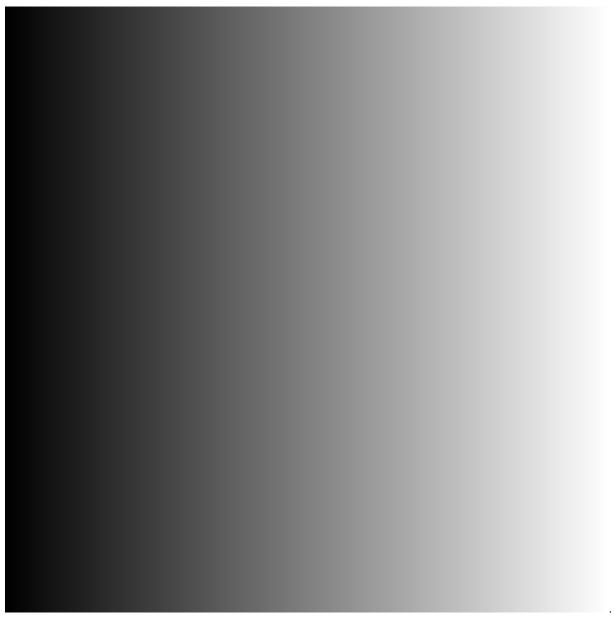
Figure 7: Original input image

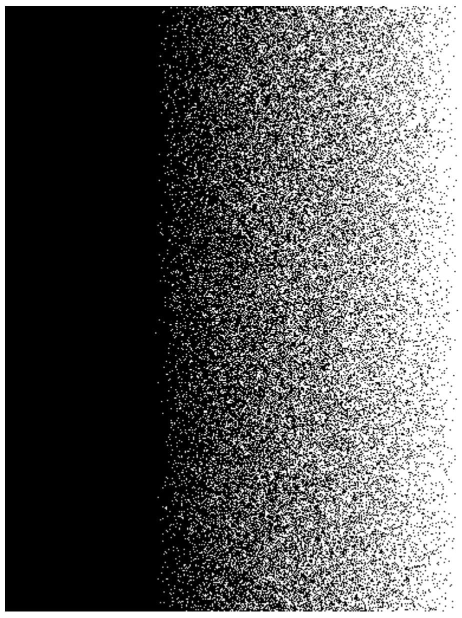
Figure 8: Thresholding with noise method output image

2 Dithering Method

With this method we try to reduce the quantization error locally, by splitting the image into small squares, inside which, the average error is close to zero. The size of the squares is defined by the adopted dithering matrix, which also defines a different threshold for each pixel inside the considered square.

Observing the result, one can easily see an improvement in the quality of the processed output, with respect to the Thresholding method, especially using a 4x4 matrix.

Still is it possible to notice some artefacts like the pattern of the squares used to process the image. This can be easily noticed in the second test input image, since it's more of a regular pattern.

2.1 Computational time

To process the two images with Dithering 3x3 are 128 and 123 milliseconds, while using the 4x4 matrix 113 and 117 milliseconds. It looks like there's a bit of an increase in time using the 3x3 method, probably because more square are needed to fill the entire area of the image.

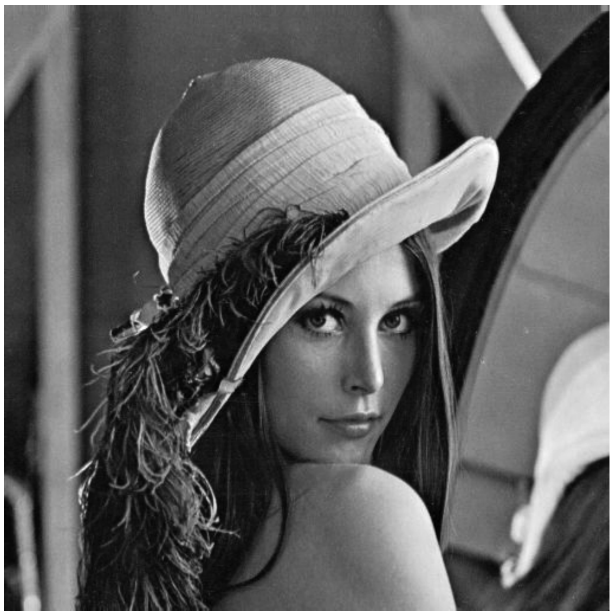

Figure 9: Original input image

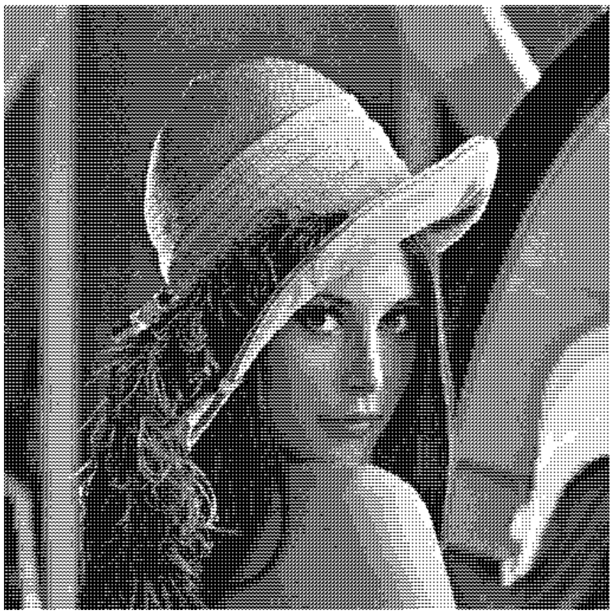

Figure 10: Dithering method using a 3x3 matrix output image

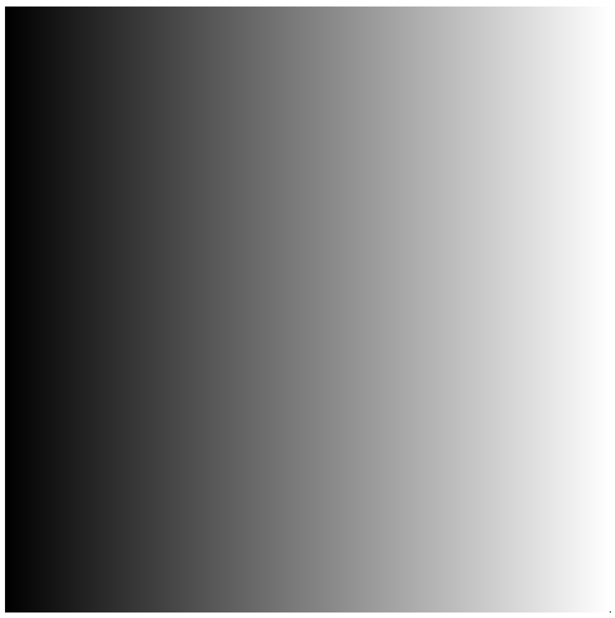

Figure 11: Original input image

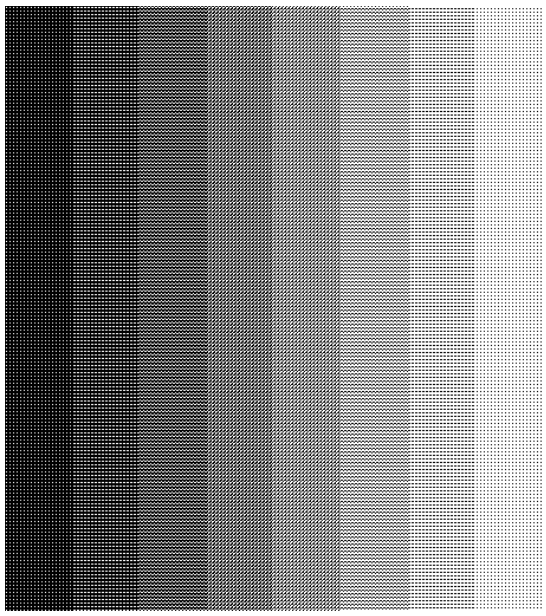

Figure 12: Dithering method using a 3x3 matrix output image

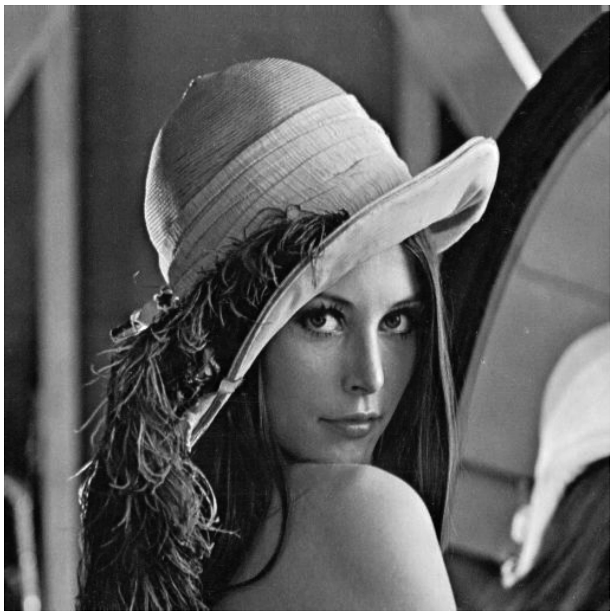

Figure 13: Original input image

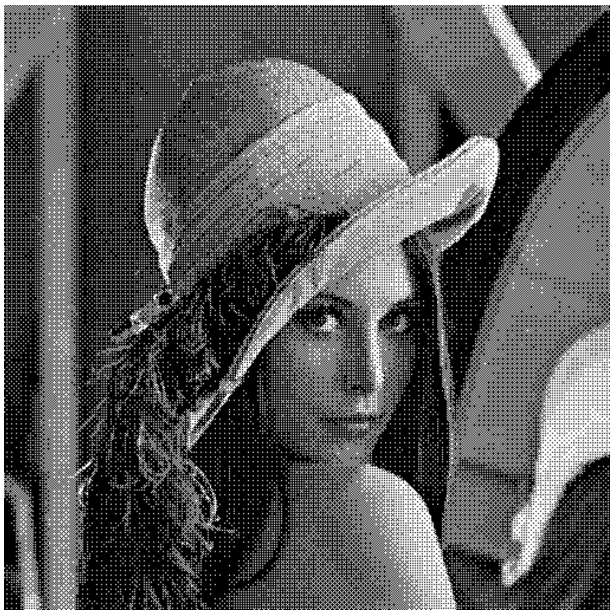

Figure 14: Dithering method using a 4x4 matrix output image

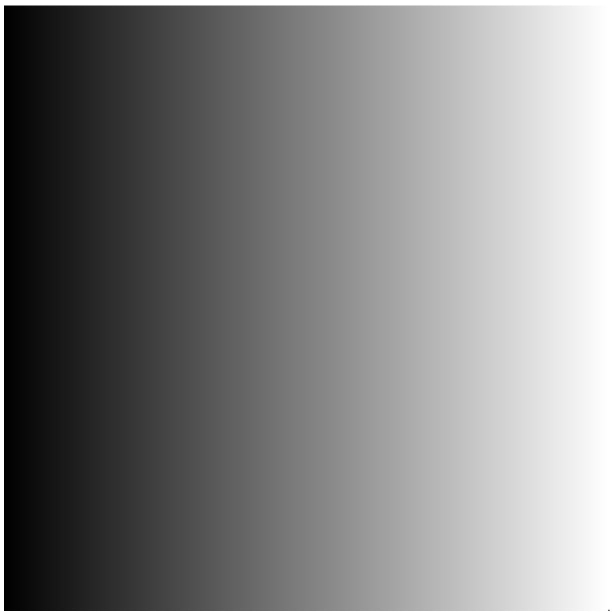
Figure 15: Original input image

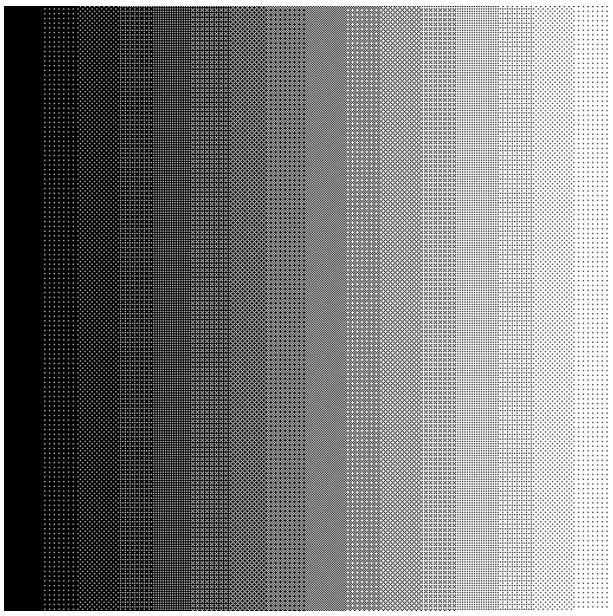
Figure 16: Dithering method using a 4x4 matrix output image

3 Error Diffusion Method

This method belongs to the second class of Halftoning methods, in which also neighbouring pixels' values are taken into account, using a scan-line propagation error. We can clearly see that the quality of the processed output image improves. I also implemented a "noised" version of the method, without any big improvement but still the best result so far, due to the lost of the pattern in the output images.

3.1 Computational time

To process the two images with Error diffusion method 147 and 145 milliseconds are needed, while using the the "noised" Error diffusion method 148 and 149 milliseconds. As expected, this method, belonging to the second class of methods, is the most computationally expensive out of all the others.

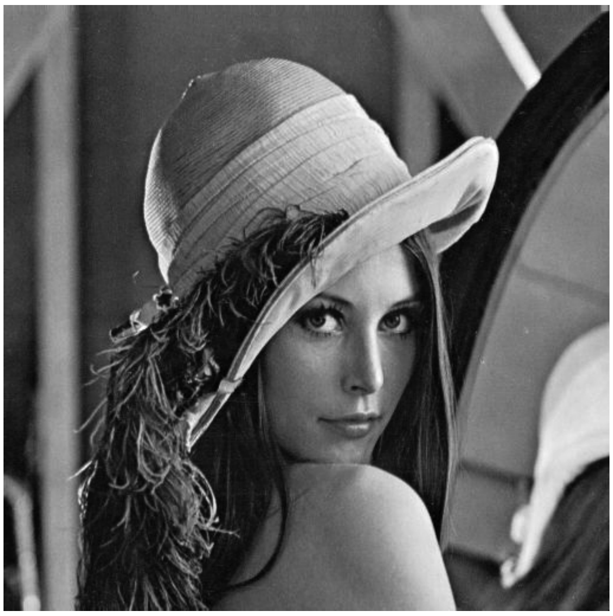
Figure 17: Original input image

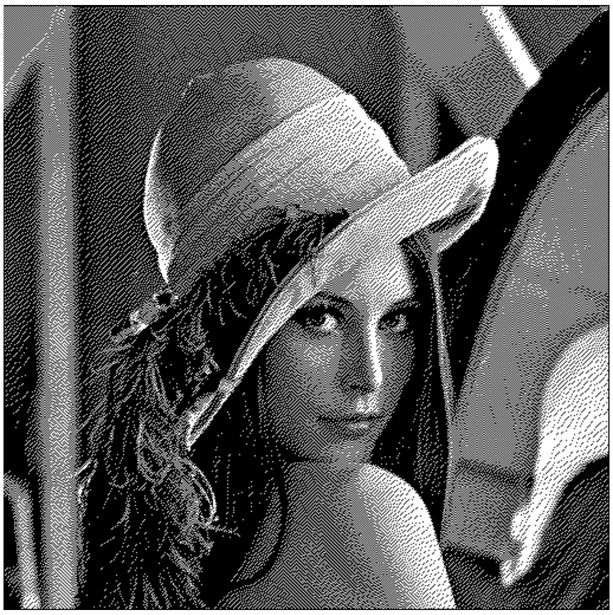
Figure 18: Error diffusion method output image

Figure 19: Original input image

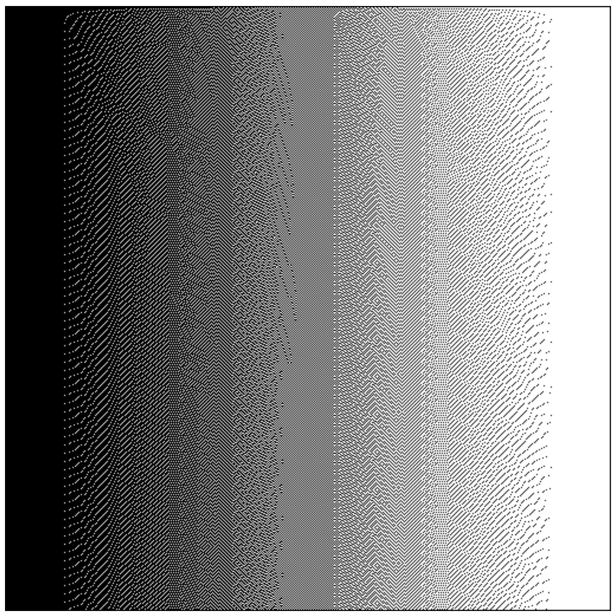
Figure 20: Error diffusion method with output image

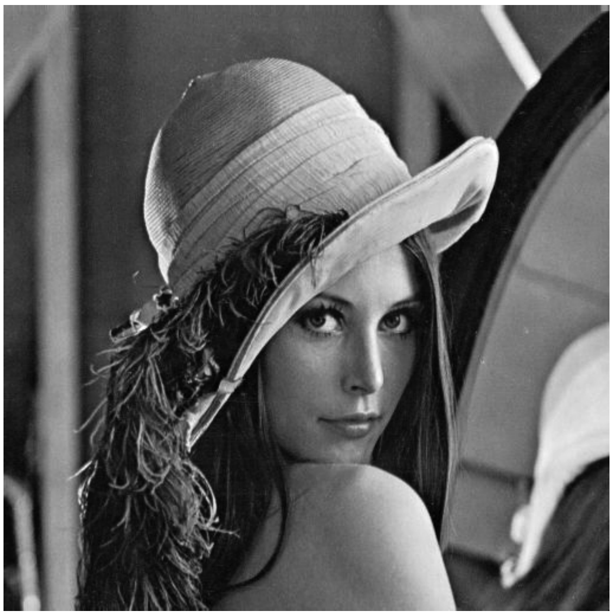

Figure 21: Original input image

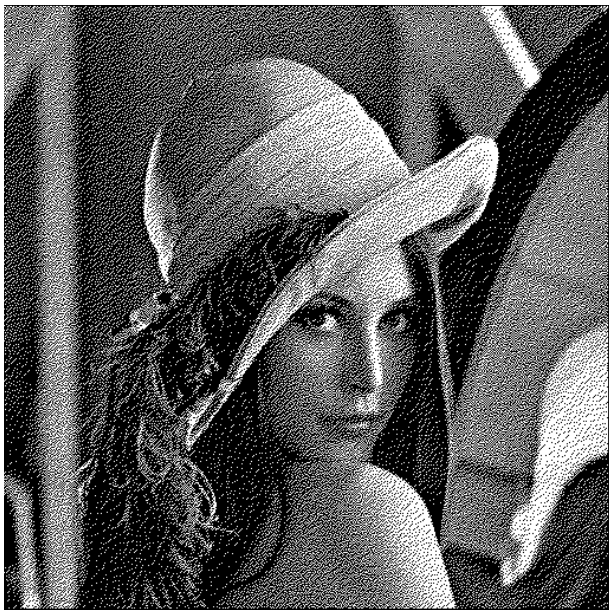

Figure 22: Error diffusion method with noise output image

Figure 23: Original input image

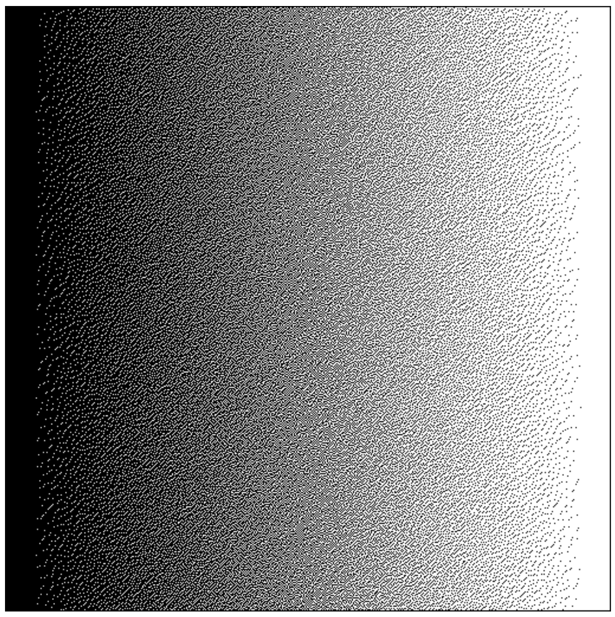
Figure 24: Error diffusion method with noise output image

4 Bonus part

4.1 Tone-Dependent Weights Floyd-Steinberg Error-Diffusion

In the presented method we use three different set of weights based on which range of greylevel value does the input pixel belongs. The idea is to differentiate the strategy for shadows, midtones, and highlights pixels. As shown by the following results, there's not such a big improvement with respect the naive Error diffusion method.

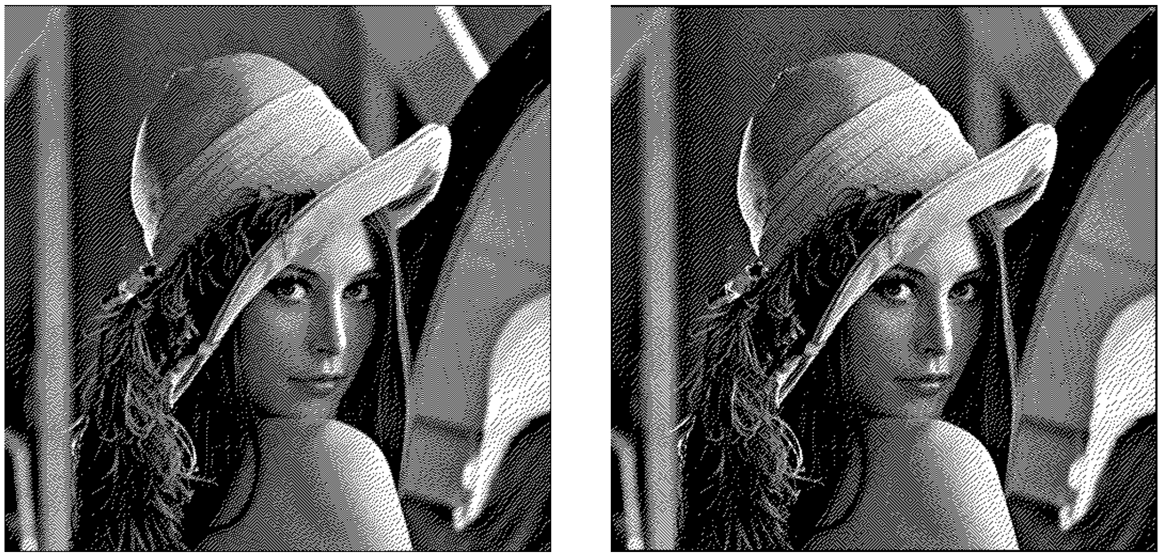

Figure 25: Using the tone-dependent weights (on the right) with respect to using the original Error Diffusion method(on the left)

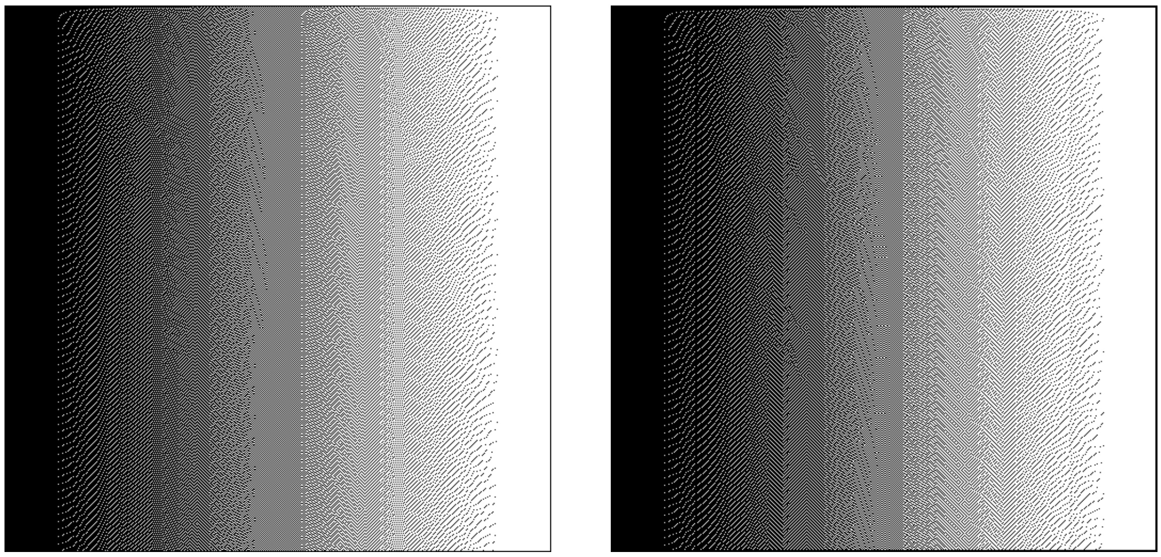

Figure 26: Using the tone-dependent weights (on the right) with respect to using the original Error Diffusion method(on the left)

4.2 Error-Diffusion using Hilbert curve

The presented methods aims to change the order in which the pixel are being processed. Instead of going row by row, indeed, we perform a maze-like pattern, computed using a function that maps the 2D position of the pixel inside the image space, into a 1D space. The results show a bit of improvement, especially in the texture image, where the vertical pattern is no longer present. In general we can say the images get a more smooth transition between the different shadings of the scene.

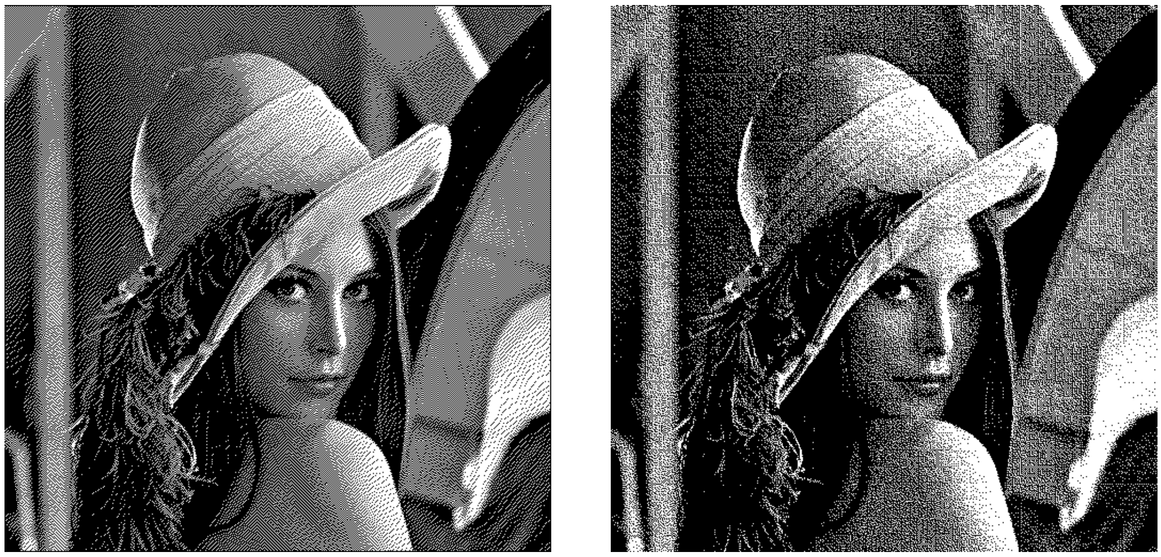

Figure 27: Using the Hilbert curve (on the right) with respect to using the original Error Diffusion method(on the left)

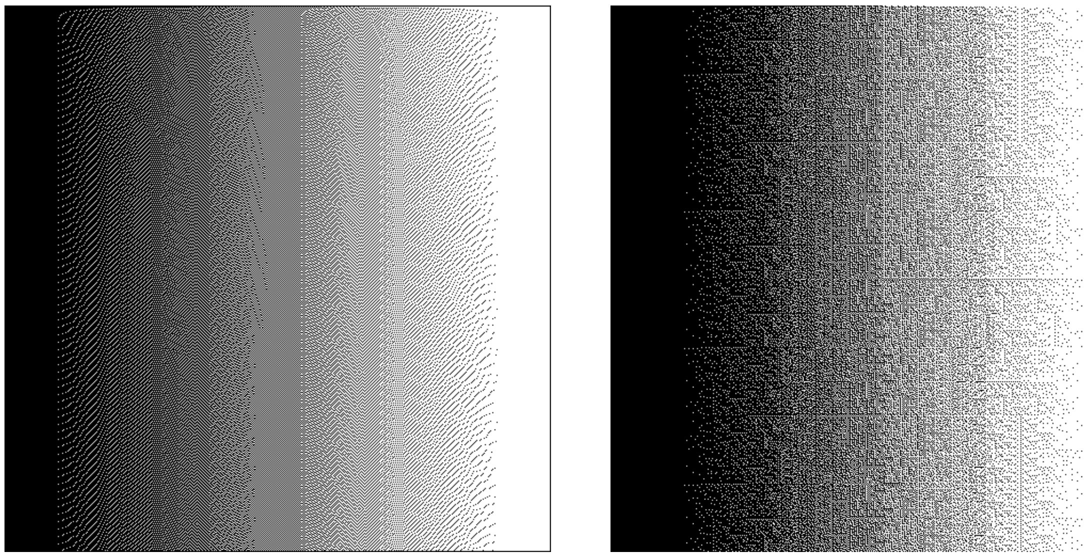

Figure 28: Using the Hilbert curve (on the right) with respect to using the original Error Diffusion method(on the left)

4.3 Tone-Dependent Weights Diffusion using Hilbert curve

Here we apply both techniques, with no such improvement.

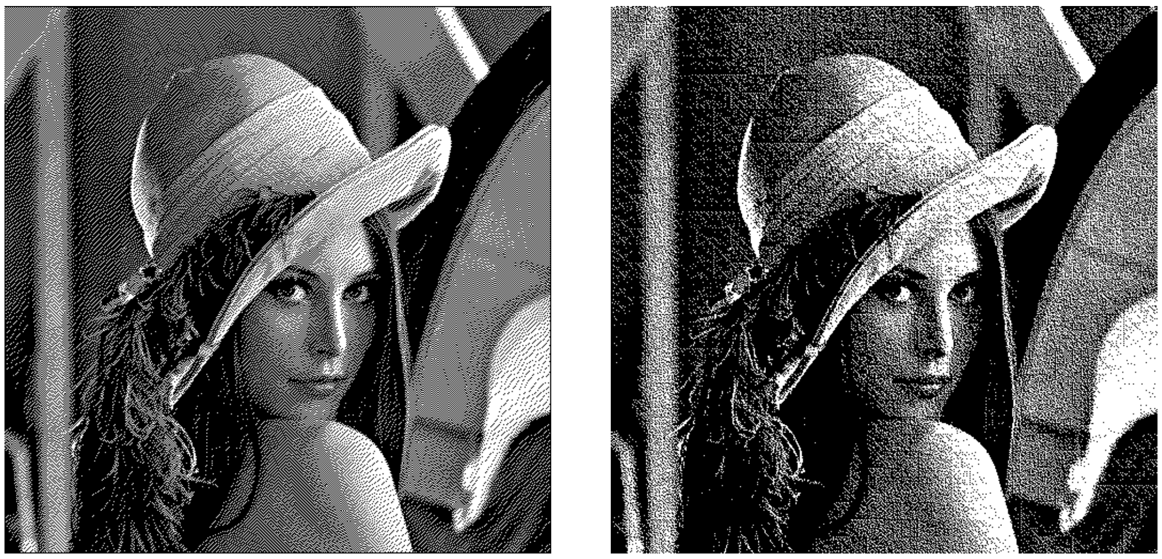

Figure 29: Using both the Hilbert curve and tone-dependent weights (on the right) with respect to using the original Error Diffusion method(on the left)

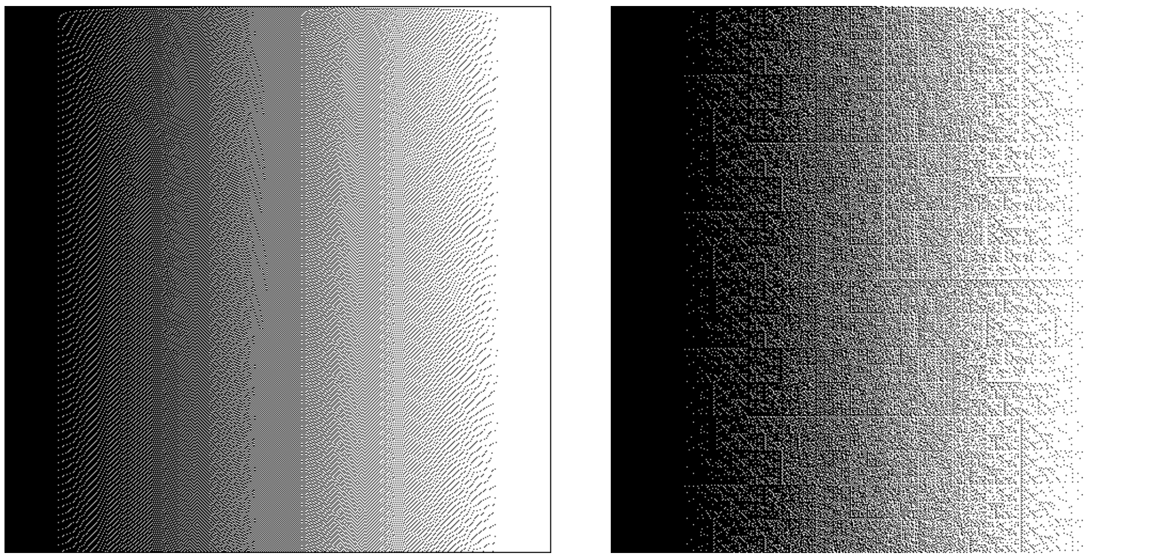

Figure 30: Using both the Hilbert curve and tone-dependent weights (on the right) with respect to using the original Error Diffusion method(on the left)