Digital Halftoning with Ordered and Error Diffusion Dithering

Thalles Santos Silva[*]
Hélio Pedrini[†]

Abstract

The ability to represent an image with the least possible number of bits while preserving most of its details is a very useful technology. This project presents a comparison work on 2 classical digital image processing techniques for image quantization. Ordered and Error Diffusion Dithering. We examine the implementation trade-off for each technique, as well as distinct strategies that aim of maximizing the overall quality of the quantized image.

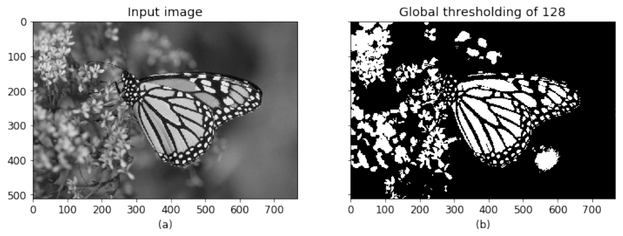

Figure 1. Result of a global threshold operation. Threshold was set to 128..

1. Configuração

Este projeto foi desenvolvido em Python 3 utlizando Jupyter Notebooks [1] para execução. As principais bibliotecas utilizadas são NumPy [2] e OpenCV [3] para processamento e matplotlib [4] para visualização dos resultados. O Notebook é dividido em subseções análogas às sessões contidas neste documento.

2. Digita Image Halftoning

2.1. Ordered dithering

Ordered dithering is an image processing algorithm commonly used to display a continuous image on a display of smaller color depth.

The main difference between dithering and uniform thresholding is the perceptual result. Global thresholding, where a given pixel is set to 0 if below a number, or 255 otherwise, produces results with a poor level of details. Figure 1 gives a sense of a typical global binarization operation.

For dithering, we want to maintain certain characteristics of global thresholding. The output image is still desired to be binary. However, we also want the output to preserve more details of the original image.

In this context, halftone dithering aims to produce a binary image such that the output segmentation keeps the maximum level of details as possible. The general idea is that we want to represent big dark regions of the image with large black blobs. Similarly, we want to represent large brighter portions of the input image with very small dark pixels. In summary, larger black dots correspond to darker regions on the image while small black pixels correspond to brighter portions of it. Here, we present 2 techniques to perform digital image dithering.

2.2. Halftone Patterning Dithering

The basic idea with halftone patterning is to use a group of pixels to represent a given intensity. In other words, for each possible value of the image, we create and display a binary cluster of pixels in its place.

Let's consider a 3x3 pattern grid as an example. The idea is that we are going to create multiple patterns in such a way that each one will have one more black pixel than the previous one. By doing this, we can represent up to 10 different intensity patterns.

However, one such problem regards what patterns to use. If we use patterns with horizontal, vertical or diagonal lines, they will leave artifacts in the resulting image. As we will see shortly, to avoid these problems we must choose patterns that form a "growth" sequence. Figure 2 shows some of the patterns that must be avoided when doing halftoning.

In Figure 3, we present a set of good patterns for a 3x3 matrix. They follow the "growth" requirement we talked about in the previous paragraph. As a result, in large areas of constant values, the patterns will be mostly artifact-free. That is because darker pixels of the input image are replaced

[*]Is with the Institute of Computing, University of Campinas (Unicamp). **Contact**: `thalles753@gmail.com`

[†]Is with the Institute of Computing, University of Campinas (Unicamp). **Contact**: `helio@ic.unicamp.br`

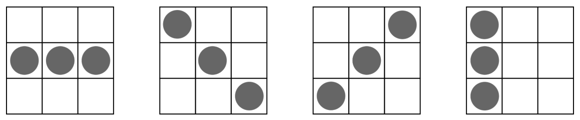

Figure 2. Set of patterns that must be avoided in halftoning.

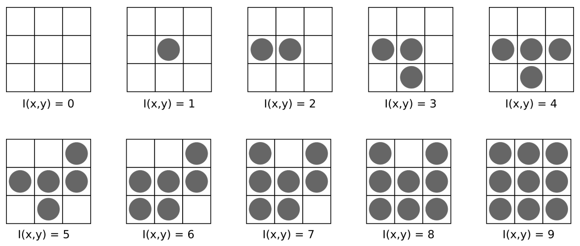

Figure 3. Set of patterns prevents artifacts on the resulting halftone image.

by patterns with most black dots. In the same way, brighter pixels of the input image should are replaced by patterns with most white pixels.

Also, by substituting each pixel in the image by one of the patterns in Figure 3, the output image will triple its size in each direction. In the same way, if we use patterns from a 4x4 grid, the image will have four times of its width and height. In summary, this method trades spatial resolution for intensity resolution.

2.3. Implementation Details and Results

To implement halftone patterning the first requirement is to make sure both, the image and the patterns have the same pixel range. To do that, we normalize the image so that it follows this condition. For the 3x3 patterns displayed in Figure 3 for instance, the input image pixels should be in the range [0-9]. In the same way, if we use a 4x4 patterning, the pixel range must fall between [0-16]. After that, we quantize the pixels so that each of them gets replaced by its corresponding pattern. Figure 3 also displays the rules adopted to identify the right patterns.

Figure 4 shows the results of halftone patterning using 2 set of kernels.

2.4. Threshold Ordered Dithering

Halftone patterning has some limitations. Most specific, the enlarged output image is not desired when the goal is to print the result or to store it. It turns out that we can represent the set of patterns in Figure 3 as a matrix. Figure 5 shows 2 such matrices. Then, we can use these matrices as thresholds to binarize the image.

To do that, we divide the original image into blocks. Then, we overlay each block with one of the pre-defined thresholding masks from Figure 5. For each pixel on the

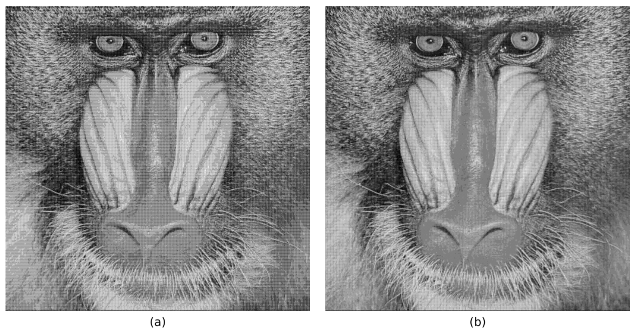

Figure 4. Halftoning output image. (a) using the 3x3 pattern from Figure 3 (b) Set of 4x4 patterns derived from the filter in Figure 5.

block, we compare it with the corresponding value of the filter. It works like a stepwise function, if a given pixel is smaller than the corresponding filter value, we set it to 0, otherwise, we set it to 255. Then, we repeat this operation for each block in the image. See results in Figure 11.

2.5. Intuition

To understand how halftoning works, let's take a look at a toy example. Consider the halftoning kernel M44 defined at Figure 5. The M44 kernel is known as the Bayer halftoning filter. If we look carefully at M44, we will see that it follows a well-defined pattern. That is, the values of 0,1,2,3...15 in the kernel are arranged in a pattern that looks like small crosses. That is the "growth" sequence that prevents checkboarding effects.

$$M_{3\times3} = \begin{array}{|c|c|c|}\hline 6 & 8 & 4 \\ \hline 1 & 0 & 3 \\ \hline 5 & 2 & 7 \\ \hline \end{array} \qquad M_{4\times4} = \begin{array}{|c|c|c|c|}\hline 0 & 12 & 3 & 15 \\ \hline 8 & 4 & 11 & 7 \\ \hline 2 & 14 & 1 & 13 \\ \hline 10 & 6 & 9 & 5 \\ \hline \end{array}$$

Figure 5. The patterns described in Figure 3 can be represented as thresholding matrices.

To validate this intuition, let's apply the thresholding Bayer filter to a set of image patches with constant intensity values.

For the first image patch, we choose a constant pixel value of 10. An intensity value of 10 makes up for a very dark image patch (considering pixel range from 0 to 255). As a result, after applying the halftoning Bayer mask, we should expect much more dark pixels in the result.

As we suspected, in Figure 6 (a), only 1 pixel (out of 16 total) got lighted up - that is only 6.25% of the total image pixels. Now, if we progressively increase the intensity of the image patch, that is, if we make it more and more lighter (by increasing its pixel intensities), we should anticipate that the number of dark pixels would diminish. That is exactly

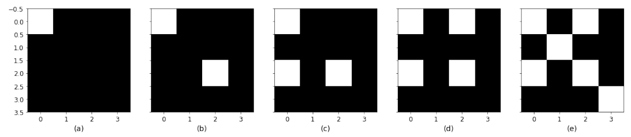

Figure 6. Applying the Bayer filter to a set of constant value image patches.

what Figure 6 (b),(c), (d) and (e) gives us. As we increase the pixel intensities of the image patch from 30 to 90, the number of darker pixels gets smaller. Also, note the pattern at which the kernel lights up the white pixels - it looks like an X shape!

Now let's consider an image with half black and half white pixels (a binary image) - Figure 7. If we apply the Bayer mask to it, we should foresee no changes at all to the outcome image. In other words, the first half of the image (only composed with 0s) would not be activated by the mask. That follows because, in this region of the patch image, all pixel values are smaller than the values of the kernel. Likewise, the other half of the image, composed solely of white (255) pixel values should all be activated (since they are all greater than the respective threshold values from the filter).

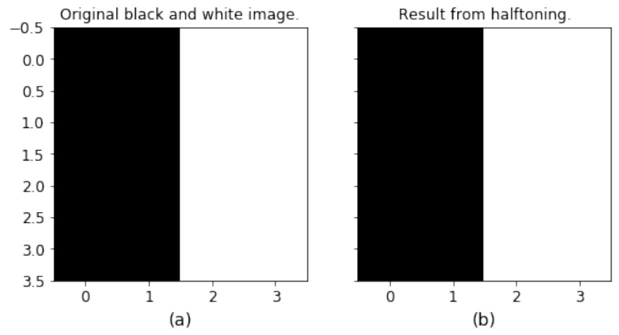

Figure 7. Applying the Bayer mask to a hafl black half white image patch.

3. Error-diffusion dithering

Note that in our previous attempts, we quantized each pixel independently. Put it differently, the process of assigning a 0 or a 255 to a given pixel was a function of that pixel alone.

The main idea with error diffusion is that we quantize each pixel using a neighborhood operation. For each quantized pixel, we push the residual (error) to the pixels in the neighborhood that will be quantized next. Here the quantization process is a simple thresholding. And the error is simply the difference between the quantized pixel and the pixel from the image. This way, we minimize the overall average error between the quantized and the original input image.

Once we calculate the residual for a given pixel, we proportionally distribute the error to the neighboring pixels in the image. This proportion may vary. Here, we analyze the Floyd and Steinberg method. Basically, this method proposes a way to push the error forward to the neighboring pixels. Figure 8 describes how Floyd and Steinberg distributes the error.

Note that when we push the error forward, we are really changing the pixel intensities of the input image. Thus, as we continue this process, eventually, some pixel will be changed so that it exceeds the threshold value of 128. If this occurs, we set this pixel to 255 (white). It is important to note that in this situation, the residual will be negative. That is, the residual would be 255 minus the current pixel value. A likely negative result. As a consequence, the error that is pushed forward will still change the neighboring pixels. However, this time, it will actually decrease their intensity values - which in turn could prevent some pixel to be above the threshold and to become white.

	f(x,y)	7/16
3/16	5/16	1/16

Figure 8. Floyd and Steinberg error distribution pattern.

In summary, instead of having a well-defined pattern that controls which pixels turn black or white, the process is governed by the error that is produced by the quantization of a given pixel.

Figure 9 shows the result of error diffusion. If we zoom in to a very dark portion of the segmented image, we can see that the distribution of the white dots follows our initial assumptions. They are sparse and yet they do not obey a specific pattern. Likewise, if we zoom in to a lighter region of the image, the distribution of dark pixels is now sparse and with no regularity.

4. Implementation details

4.1. Ordered dithering

To perform Ordered and Error Diffusion dithering we carried out some pre-processing operations.

For Pattern and Threshold Ordered Dithering we used the 2 pre-defined filters in Figure 5. Pattern Dithering always outputs a larger image. For Threshold dithering, depending on the dimensions of the input, we padding the

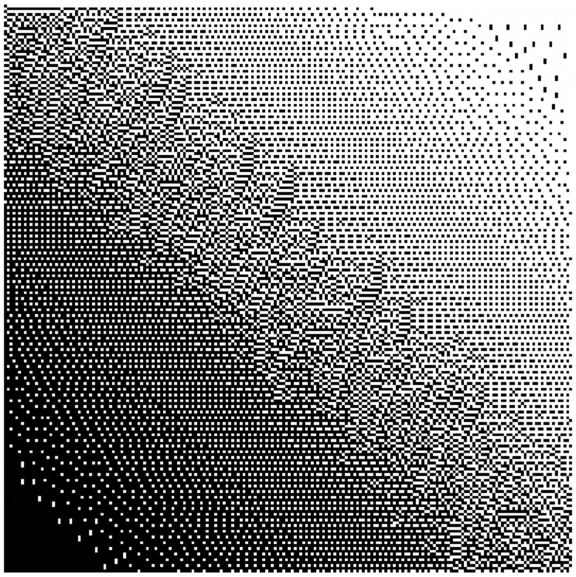

Figure 9. Error-diffusion dithering results.

right and bottom sides of the image with 0s (before applying the filter). This ensures that the operations would be successfully done to all pixels and the output image will have the same dimensions as its input counterpart.

For Threshold dithering, we had 2 options to apply the kernels. We could scan the image, block by block in a loop, or we could create a mask of the same size as the input image and apply it once (with a vector operation). Each approach has advantages and downsides. We opted to create a thresholding mask and use a vector operation to perform dithering. Also, since the kernels will be used as a threshold mask, we need to normalize the filters or the image so that they have the same pixel range. We chose to normalize the filter instead of the image using the following equation depicted in Figure 10.

$$T(i,j) = 255\frac{I(i,j)+0.5}{N^2}$$

Figure 10. Converting filter I to thresholding kernel T.

Figure 11 shows the results of applying kernels M33 and M44. Note how the filter M33 tends to leave checkboard artifacts in the output image.

4.2. Error Diffusion dithering

For Error Diffusion dithering, we also performed a preprocessing zero-padding operation. Because of the way Floyd and Steinberg push the error forward, padding is only necessary to the right and bottom sides of the input image. Note that, after dithering, we only care about the original area of the image.

When performing Error Diffusion, the order in which we apply the filter matters. Different ways of applying the same filter may output better results. Figures 12 and 13 show the results of Error Diffusion dithering with 2 sliding patterns. In subfigures (b) we applied the filter from left to right. In subfigures (c), the same filter was applied in a serpentine/zig-zagging fashing. Note that the zip-zagging pattern produces better results than the standard left-to-right.

5. Conclusion

In summary, this project explored the two techniques for digital image dithering - Ordered and Error Diffusion dithering. We investigated different strategies for each algorithm as well as the implementation of trade-offs.

References

[1] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016. 1

[2] Oliphant Travis E. A guide to numpy, 2006. [Online; accessed 15/04/2018]. 1

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 1

[4] John D. Hunter. Matplotlib: A 2d graphics environment, 2007. [Online; accessed 15/04/2018]. 1

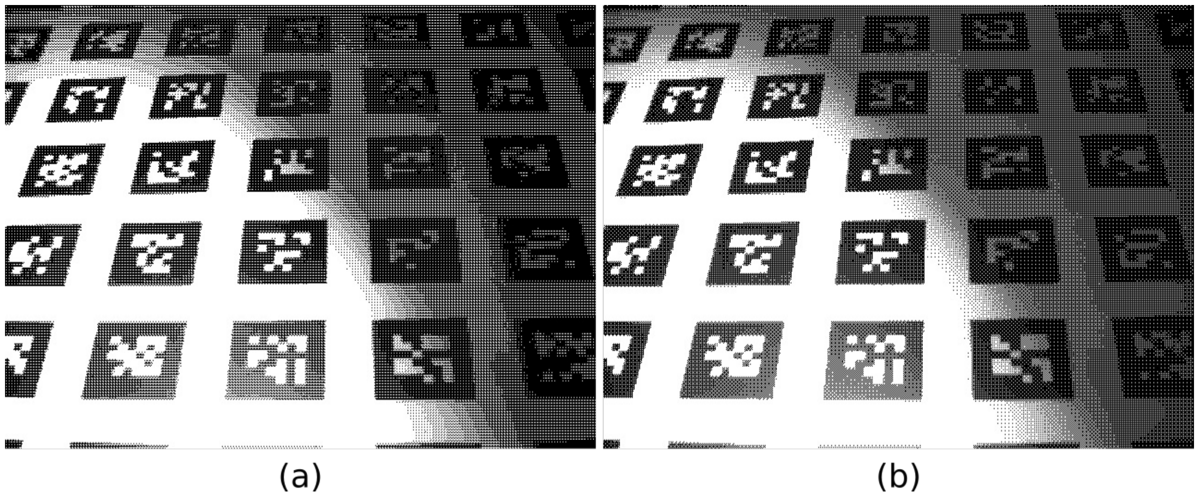
Figure 11. (a) digital halftoning using filter M33. (a) digital halftoning using filter M44.

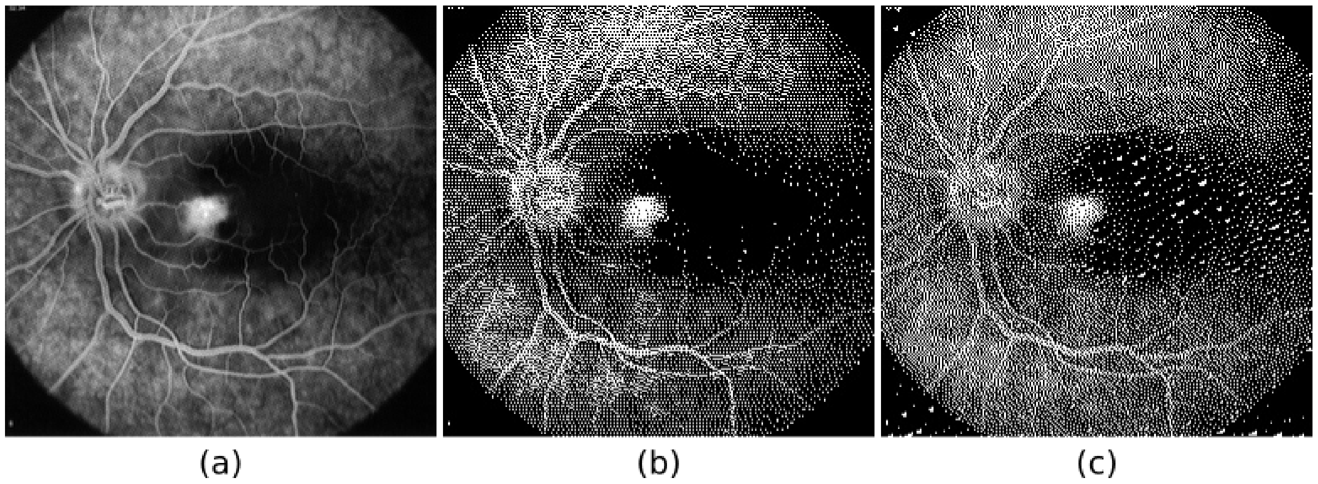
Figure 12. Error Diffusion dithering with different filter pattern application. (a) input image. (b) Error diffusion with serpentine/zig-zag filter sliding. (c) Error diffusion with left to right filter sliding.

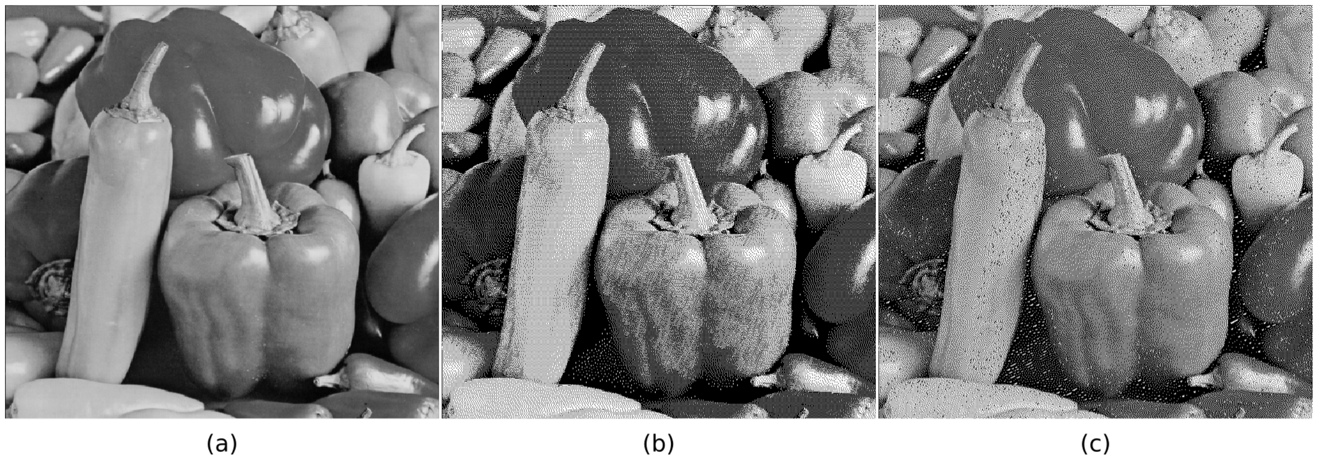
Figure 13. Error Diffusion dithering with different filter pattern application. (a) input image. (b) Error diffusion with serpentine/zig-zag filter sliding. (c) Error diffusion with left to right filter sliding.