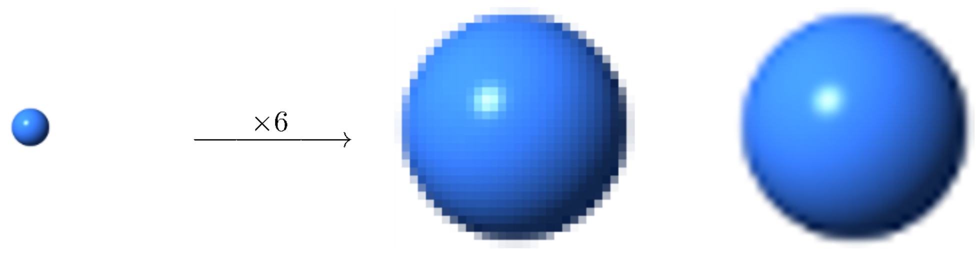
apply the pixel (m, n) into the square of pixels $(6m + k, 6n + l)$ with $k, l \in \{0, 1, \ldots, 5\}$. This corresponds to None in GIMP. It seems the only natural solution but the results are in general visually poor because we see the enlarged edges of the squares of the pixels. A better solution is to assign the color to the pixels $(6m, 6n)$, use them as nodes and interpolate the rest of the pixels. Exact cubic interpolation would be expensive (an image 640×400 requires 256000 nodes) while B-spline approximation is affordable. How can we manage B-splines in this $2D$ setting? If $f = f(x, y)$ is the function giving the color, (2.42) generalizes promptly to approximate it by

$$(2.44)\qquad \sum_{k=1}^{4}\sum_{l=1}^{4} Q_k(x - x_j)Q_l(y - y_m)f(x_{j+3-k}, y_{m+3-l}) \qquad \text{for} \quad x_j \le t \le x_{j+1}, \quad y_m \le t \le y_{m+1}.$$

This is the Cubic method. As you suspect, Linear means piecewise linear approximation. Finally Sinc is an interpolation method related to the sinc function, a kind of smoothing of (2.8). Here you can see an example of enlarging an image without and with B-splines. Which one do you prefer?



Original 30×30 None 180×180 Cubic 180×180

For the avid reader, a last brief comment about non-interpolating cubic curves. Probably you have heard the name *Bézier curves*. They are based in the following fact: Given $P_0, P_1, P_2, P_3 \in \mathbb{R}^2$, the curve $\sigma : [0, 1] \longrightarrow \mathbb{R}$

$$(2.45)\qquad \sigma(t) = (1 - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2(1 - t)P_2 + t^3 P_3.$$

joins P_0 and P_1 and pass somewhat close to P_1 and P_2. The segments P_0P_1 and P_2P_3 are tangent to the curve. The Bézier curves are composed by curves like this. Many interactive applications use them because it is quite intuitive to use P_1 and P_2 as *control points* that when moved change locally the aspect of the curve.

Suggested Readings. Lagrange interpolation, splines and B-splines are discussed in the second chapter of [SB02]. Look up this book for a mathematically spotless treatment of several numerical analysis methods. There is a lot of information about Bézier curves and their relatives on the internet and on texts oriented to the applications. I have found very clear the exposition in [Bus03].

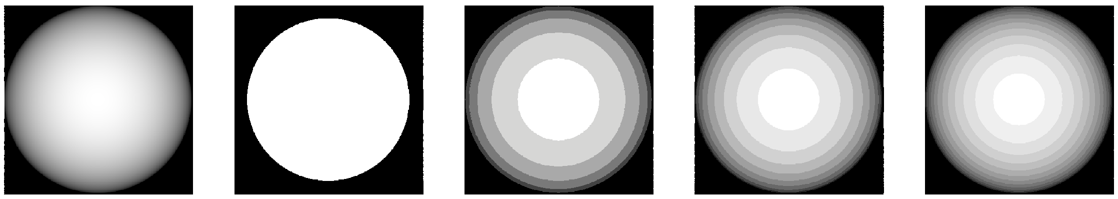
2.1.4 Dithering

If I say that *dithering* consists of improving A/D conversion adding noise to the analog signal you will think I am crazy. We are not talking about me but truly this seems idiotic until one sees an example with images or sound. The possibilities to include sounds here are not very informative (they reduce to boo, miaow, ha, ...) then we focus only on the case of images.

Consider the semi-spherical shaped function

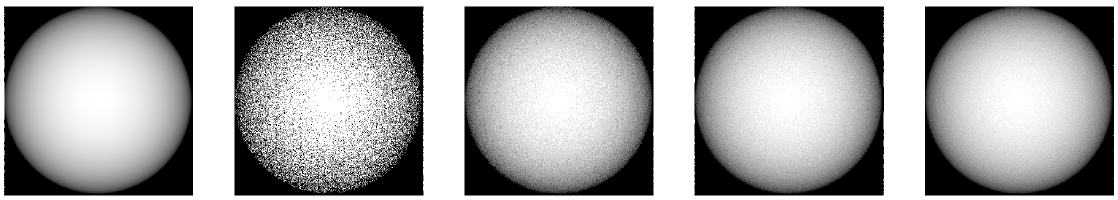
$$f(x,y) = \sqrt{\max(1-x^2-y^2,0)} \qquad \text{for} \quad x,y \in [-1,1]. \tag{2.46}$$

Let A be the map performing the spatial quantization with a grid 400×400. In this way A is a 400×400 matrix storing orderly the values of f. Let us assume that the value of $f(0,0)$ is clamped[3] to $0.999\ldots$ and hence the elements of A are real numbers $a_{ij}\in[0,1)$. If we have a large palette of gray tones (theoretically an infinite one), we can represent faithfully the values in $[0,1)$ and plot A as a spherical gradient. In some situations we have a reduced palette, for instance in newspapers or GIF images. Let us say that we have to our disposal c gray tones labeled as $0,\ 1,\ 2,\ldots,\ c-1$ varying from black to white. Quantizing under this restriction, our digital assigns the color $\lfloor ca_{ij}\rfloor$ to the pixel (i,j). This is the natural option and it seems the only sound option. On the other hand, the following examples show that the result is utterly disappointing if c is small.



When $c=2$, as the graph of f is very steep near the unit circle, the values with $f>1/2$ determine a circle close to it, so more or less we have a white circle where we had a gradient. For greater, but still small, values of c, we see clearly rings corresponding to the quantized level that do not recall the original image.

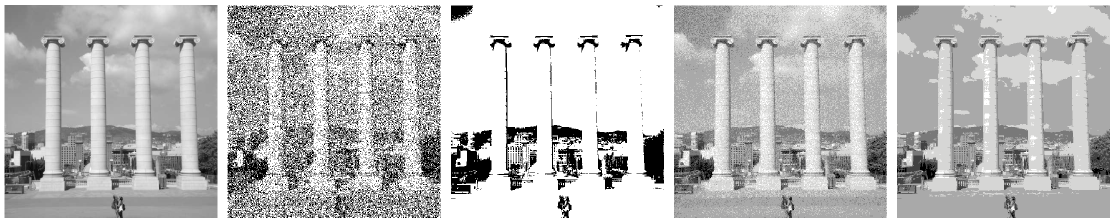
Now it goes the crazy idea. Let ξ_{ij} be independent random variables under a uniform distribution in $[0,1)$. The noisy discrete signal (matrix) $\widetilde{A} = A + (c-1)^{-1}\Xi$ with $\Xi = (\xi_{ij})$ has elements $\widetilde{a}_{ij}\in\big[0, c/(c-1)\big)$. Now, the digital image with color $\lfloor (c-1)\widetilde{a}_{ij}\rfloor$ at pixel (i,j) looks, no doubt, more satisfactory



[3]The term *clamping* or *clipping* is used in signal processing to mean that the signal f is constrained to an interval $[a,b]$ changing f by $\max(\min(f,b),a)$ usually to avoid overflow errors.

The result for $c = 14$ is not bad and for $c = 2$, although it is clearly defective, gives a better idea about the gradient than a perfect white circle.

Usual photographs are plenty of gradients but often what calls our attention are the sharp edges, then the brute force dithering we have just introduced, called *random dithering*, could be less effective than in the previous example



In these images (of size 500×500) for $c = 2$ without differing we lose completely the clouds and the sky, nevertheless the result may be more pleasant than the noisy image with dithering. For $c = 6$, without dithering the sky is artificial and with artifacts and the columns show an unrealistic aspect but somebody with a not so twisted taste could consider it preferable to the version with dithering that is more informative with respect to the details but contains visible grains of noise.

The are several methods to avoid the noisy aspect caused by random dithering. Essentially all of them substitute the randomness by something deterministic. We discuss here two of these methods. Both are quite popular, being a noticeable asset the ease to program the corresponding algorithms even in a low-level environment.

Firstly we present the most common form of *ordered dithering*. It uses a real matrix M_k of size $2^k \times 2^k$, called *Bayer matrix*, defined according the recurrence

$$(2.47)\qquad M_k = \begin{pmatrix} M_{k-1} & M_{k-1} + 2\cdot 2^{-2k} \\ M_{k-1} + 3\cdot 2^{-2k} & M_{k-1} + 1\cdot 2^{-2k} \end{pmatrix} \qquad \text{with} \quad M_0 = (0).$$

The entries of M_k are a rearrangement of $\{j2^{-2k}\}_{j=0}^{2^{2k}-1}$ that, according to the literature, maximize the average distance between consecutive entries where the last row/column is consecutive to the first one (I have not been able to find a proof in the bibliography). The first two Bayer matrices are

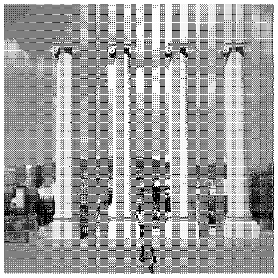
$$(2.48)\qquad M_1 = \frac{1}{4}\begin{pmatrix} 0 & 2 \\ 3 & 1 \end{pmatrix} \qquad \text{and} \qquad M_1 = \frac{1}{16}\begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}.$$

In principle a greater matrix is better but on the other hand, it has to be small in comparison to the size of the image. Once we have selected the matrix, we proceed as before but with the noise

$$\xi_{ij} = M_{i'j'} \qquad \text{with} \qquad i' \equiv i \pmod{2^k}, \quad j' \equiv j \pmod{2^k} \tag{2.49}$$

where $M_k = (m_{ij})$. In other words, the noise matrix is obtained now tiling the image size with M_k.

Let us see the results for 4×4 matrices ($k = 2$):

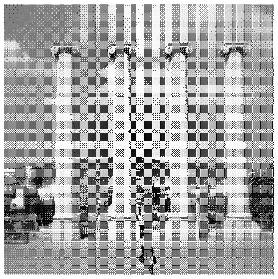

c = 2

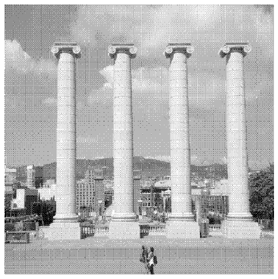
c = 4

c = 6

c = 8

The result is clearly much better than the one obtained with random dithering. With 16×16 matrices ($k = 4$, often employed in practice) the improvements are not very noticeable.


c = 2


c = 4

c = 6

c = 8

Although the results are rather good, cross-hatch pattern artifacts appear typically with this method. The result looks very often like fabric and, obviously, images with big variations between adjacent pixels can give weird results. One alternative is to combine both techniques adding a small random noise to reduce the visual impact of the pattern due to M_k. Another possibility is to use tiles with more involved forms instead of square matrices. We do not explore these possibilities but a completely different idea.

There is a collection of techniques generically known as *error diffusion dithering* that try to compensate the quantization error in each pixel distributing it among the values of neighboring pixels. By far the most common version is the *Floyd-Steinberg dithering* [FS76]. In it, the pixels are scanned from left to right and from top to bottom and in each pixel the quantization error is distributed according to the proportion indicated in the following scheme

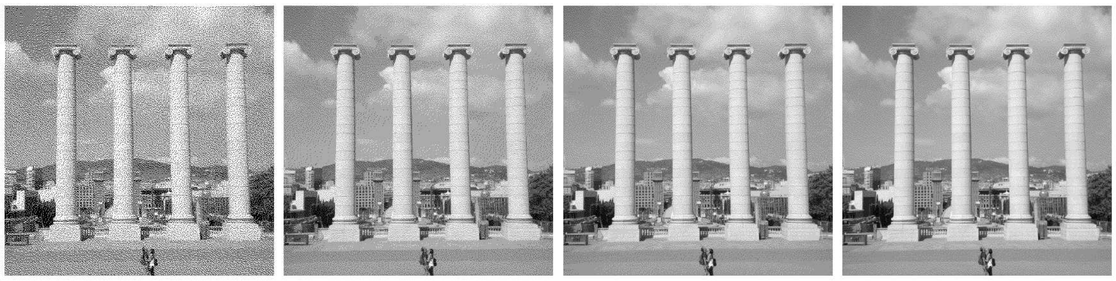
	■	7/16
3/16	5/16	1/16

The pixel marked in black is the one that has been quantized and the rest of the pixels in the scheme receive a part of the quantization error, of course before suffering themselves quantization. In formulas, the pass from a matrix A as before to its quantized version B with elements (colors) in $\{0, 1, 2, \dots, c-1\}$ is

$$\begin{cases} b_{ij} = \lfloor c a_{ij} \rfloor, & e_{ij} = b_{ij} - c a_{ij} + 1/2 \\ a_{i+1\,j} = a_{i+1\,j} + 7e_{ij}/16, & a_{i+1\,j+1} = a_{i+1\,j+1} + e_{ij}/16 \\ a_{i-1\,j} = a_{i-1\,j} + 3e_{ij}/16, & a_{i\,j+1} = a_{i\,j+1} + 5e_{ij}/16. \end{cases} \tag{2.50}$$

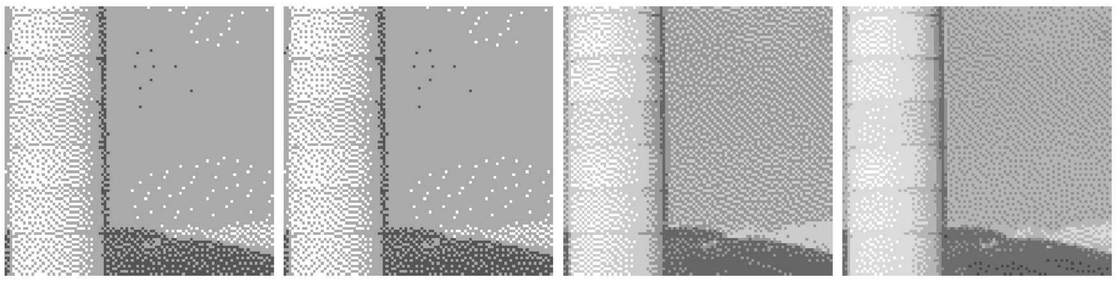
For pixels on the boundary one has to define in some way a_{ij} with $i, j \in \{0, N\}$. This is less important and simply skipping these values does not cause a problem.

The results are quite impressive given the simplicity of the method:



c = 2 c = 4 c = 6 c = 8

To better appreciate the quality of the result, here it is a zoom of the central 100×100 square of the images:



c = 2 c = 4 c = 6 c = 8

Probably you are wondering what is interesting in this choice of the coefficients in the scheme (other diffusion methods employ different neighboring pixels and coefficients). It turns out that for $c = 2$, when having a middle gray $a_{ij} = 0.5$, the result is a checkerboard pattern. For a greater number of color, the middle tones are represented in this optimal way replacing in the checkerboard black and white by the upper and lower adjacent colors.

For people that like whining, Floyd-Steinberg dithering has a serious drawback, the diffusion of the quantization error tends to form worm-like lines in the dithered image. An idea to minimize this effect is to raster the pixels in a funny (fractal) way instead following horizontal lines, another idea is to explore other diffusion schemes.

With a little of experimentation, surely a lot of tricks will come to your mind. You can find hundreds of ideas in the literature (even books like [Uli87]). Unfortunately some of them, although simple, are under patent laws (that also apply if you rediscover the idea on your own).

Suggested Readings. There is a lot of information about dithering in the internet. One of the most informative sites that I have found is `http://caca.zoy.org/study/`. Do not get wrong with the software under lavatory names hosted in the main page, it is done by serious programmers.