Blurred lines

In a computer, an image is ultimately a table of numbers representing pixel intensity values. We can regard these values as a matrix and model some common image operations using linear algebra. A prominent example of such operations is blurring due to poor focus.

Consider first an $m\times 1$ vector $\boldsymbol{x}$ of pixel intensities, mapped to a vector $\boldsymbol{z}$ of intensities after blurring. We model each pixel in the new vector as a weighted average,

$$z_i = \frac{1}{4}x_{i-1} + \frac{1}{2}x_i + \frac{1}{4}x_{i+1}.$$

Pixel values that are beyond the boundaries, namely x_0 and x_{m+1}, are taken to be zero. We can express the relationship between original and blurred image as a matrix-vector product:

$$\boldsymbol{z} = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} & 0 & \cdots & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & \cdots & 0 \\ & \ddots & \ddots & \ddots & \\ 0 & \cdots & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \cdots & \frac{1}{4} & \frac{1}{2} \end{bmatrix} \boldsymbol{x} = \boldsymbol{B}_m \boldsymbol{x}, \tag{1}$$

where the m subscript indicates an $m \times m$ matrix.

Things are not much more complicated for actual 2D images. Let $\boldsymbol{X}$ be an $m \times n$ image and $\boldsymbol{Z}$ be a blurred version of it. We can first blur in the vertical direction by recalling the columnwise multiplication identity,

$$\boldsymbol{B}_m\boldsymbol{X} = \begin{bmatrix} \boldsymbol{B}_m\boldsymbol{X}_1 & \boldsymbol{B}_m\boldsymbol{X}_2 & \cdots & \boldsymbol{B}_m\boldsymbol{X}_n \end{bmatrix}.$$

Thus, $\boldsymbol{B}_m\boldsymbol{X}$ applies vertical blur to each column of the image.

To blur rows, we go in three steps. First, transpose the image to turn rows into columns. Second, apply blurring to each column as shown above. Finally, transpose the result back to restore the original orientation. Hence the fully blurred image is given by

$$\boldsymbol{Z} = \left[\boldsymbol{B}_n(\boldsymbol{B}_m\boldsymbol{X})^T\right]^T = \boldsymbol{B}_m\boldsymbol{X}\,\boldsymbol{B}_n^T = \boldsymbol{B}_m\boldsymbol{X}\,\boldsymbol{B}_n, \tag{2}$$

where the last step uses the symmetry apparent in (1).

Finally, we can simulate a larger amount of blur by applying our simple blur multiple times, as in $(\boldsymbol{B}_m)^k\boldsymbol{X}(\boldsymbol{B}_n)^k$ for some integer $k > 1$.

Preparation

Read Section 2.2. Then read the online documentation for `diag`, paying particular attention to the examples.

Goals

You will write a function that produces a blur matrix of requested size. This will be used to blur an image that you import as a matrix.

Procedure

Download the template script `Blur` and the template file `blurmatrix.m`.

1. The file `blurmatrix.m` has the first line

   ```
   function B = blurmatrix(n)
   ```

   Within the function, `n` is an input parameter describing the size of the matrix, and `B` is the result that is returned to the caller. Using the functions `ones` and `diag`, complete the definition of the function so that it returns $\boldsymbol{B}_n$ as in (1).

2. In the script, construct the matrix $\boldsymbol{B}_7$ using a call to `blurmatrix` and have it printed out (no semicolon) to show that it's correct.

3. Save an image to your working folder. (It should have between 400 and 1000 pixels in each dimension.) The template script has code for importing it, converting it to a floating-point matrix $\boldsymbol{X}$, and displaying it.

4. Apply blurring to your image 60 times in the vertical direction only, and display the result as in step 3.

5. Blur the image 60 times in each direction and display it.

Extras

E1. Look at the output of the final step. What creates the "black lines" around the edges? What happens to them as the number of blur applications increases?