Programming Assignment #1

Announcement:	07 January 2020
Submission Deadline:	21 January 2020

Description

The goal of this assignment is to gain an understanding of demosaicing: converting the Bayer pixel pattern into a RGB representation where each pixel has red, green and blue color channels.

Demosaicing is typically the first step in a sequence of non-linear operations of the imaging pipeline that converts the raw output of the digital camera's sensor into a JPEG-compressed image which can then be displayed on the monitor. The figure below shows an example of a Bayer arrangement on the pixel array of an image sensor.

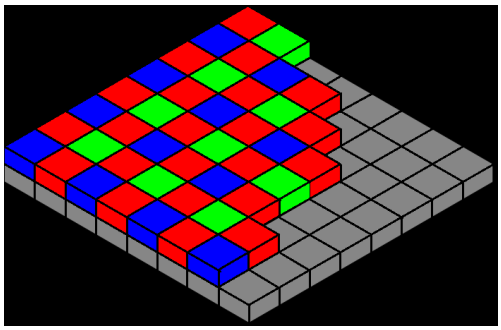


Figure 1: Bayer pattern

Part 1:

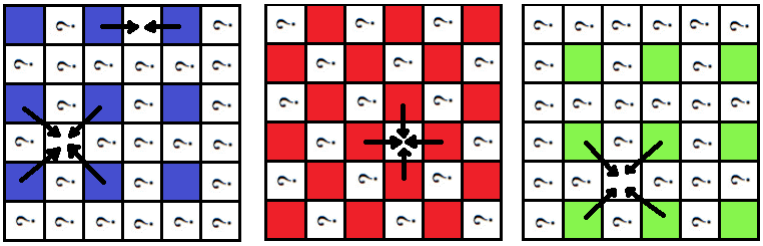
Implement a very simple linear interpolation [averaging the 4 or 2 nearest neighbours] approach that converts the Bayer arrangement to RGB components for each pixel as shown in the figure below [note that the figure is an example and does not reflect the Bayer arrangement used by the digital camera]. You should create a kernel for each channel and use cv::filter2D(...) instead of using loops. The "mosaic" image was created by taking the original color image and keeping only one color component for each pixel, according to the standard Bayer pattern:

B R B R. . .

R G R G. . .

B R B R. . .

. . ,



This method produces artifacts. To visualize these artifacts compute the image of the **squared differences between the original and reconstructed values for each pixel, summed over the three color channels**.

Original, demosaic, root squared difference.

Finally, show a close-up of some patch of the reconstructed image where the artifacts are particularly apparent and explain the cause of these artifacts.

Part 2:

Bill Freeman proposed an improvement of the simple bilinear interpolation approach. Since the R channel is sampled at a higher rate than the G and B channels, one might expect interpolation to work better for R values. Then it would make sense to use the interpolated R channel to modify the interpolated G and B channels. The improved algorithm begins with linear interpolation applied separately to each channel, just as you have already done above. The estimated R channel is not changed, but G and B

channels are modified as follows. First, compute the difference images G-R and B-R between the respective interpolated channels. Mosaicing artifacts tend to show up as small "splotches" in these images. To eliminate the "splotches", apply *median filtering* to the G-R and B-R images. Finally, create the modified G and B channels by adding the R channel to the respective difference images.
Implement the above algorithm and visualize the quality of the results in the same way as for Part 1 by displaying the per-pixel difference image. Compare the output to that of Part 1. Are there visible improvements (especially in the close-up patch selected in Part 1)?

Hint: Implementing this method should take you six lines of code.

Submission (electronic submission through EAS only)

Please create a zip file containing your C/C++ or Python code and a readme text file (.txt).
In the readme file document the features and functionality you have implemented, and anything else you want the grader to know i.e. control keys, keyboard/mouse shortcuts, etc.

Additional Information

- Mosaic images and their originals can be downloaded from here: image_set.zip