

Image Demosaicing for Bayer and Quad-Bayer Pattern

Brice CONVERS^a

^aGrenoble INP Phelma SICOM, brice.convers@grenoble-inp.org,

1. Introduction

Numerous RGB cameras in the commercial sector employ Color Filter Array (CFA (5)) technology. This technology involves an array of red, green, or blue filters placed atop the sensors, usually organized in periodic patterns. The incident light is consequently filtered by each filter, before being captured by the sensor. The typical process of acquiring images utilizes a predefined CFA pattern to allocate a color to each pixel of the sensor. You can observe these pattern in Fig 1. There are several types of Bayer pattern. These differ according to the order of each Red, Green and Blue pixel. In our project, we are only interested in the GRBG type, which means that the pixels in the pattern alternate: Green, Red, Green and Blue. There is an example in the equation 1. For other types of Bayer pattern, the logic remains the same.

The aim of this project is to reconstruct CFA images in order to recover a color image composed of RGB components for each pixel.

$$\text{Bayer_Pattern} = \left[\begin{array}{c|c} G & R \\ \hline B & G \end{array} \right] \quad (1)$$

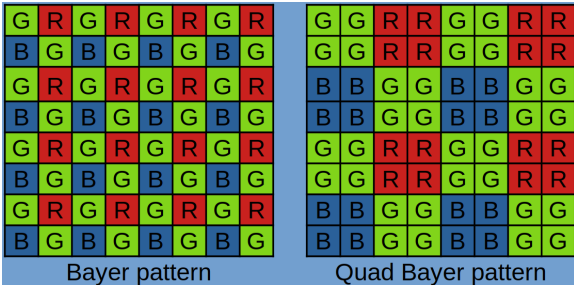


Figure 1: Visualization of two Bayer pattern

2. Project Details

2.1. Tested Images

In this project, we test our demosaicing algorithms on four images found on the dataset of the *National Gallery of Art, USA* (7). Since these images represent four paintings, it was necessary to add other real-life pictures to have more accurate tests. That is why two pictures taken in Japan have been included. (*img_5* was taken in Shikotsu lake in Hokkaido and *img_6* was taken in Shosenkyo). The paintings have a size of (1024×1024) and the pictures have a size of (5444×3629).

2.2. Implemented Methods

To retrieve an original image, there is a first implementation of a basic interpolation explained in (6). In this document, this method will not be explained. This will simply be a basic comparison for further tests.

After research on different demosaicing algorithms, a method proposed in (3) has been chosen for this project. We will call it as *Menon* method. Basically, this new method is based also on interpolation, and takes advantage of other information such as edges and the preponderant pixel type, which is in our case: The Green pixel. This method has been developed to solve a Bayer pattern problem visible on the left part of Fig 1. Since *Menon* method seems to be one of the most advanced algorithms, to solve the Quad Bayer pattern we will also use this method. However, we will first apply a transformation on the pattern to act as a Bayer pattern.

2.3. Code Architecture

In this part, we deal with the most important parts of the code: *Menon* method. You can find more information on the *README.py* in (1). The *Menon* method lives in the *menon.py* file. We will explain more about the algorithm in the next part. The *DataHandler.py* file is a class which manages all the data. On this file you can find entry point to execute multiple tests. All the functionalities such as plot or measure can be proceeded with this file. However, to fit the original project framework, it is also possible to execute the *Menon* method directly with the function **run_reconstruction** in the *brice_convers* “*reconstruct.py*” file. Finally, the *dataEvaluation.py* file is used to apply measures between original and retrieval images.

Originally, this project was to retrieve a colored image from a gray image produced by a camera sensor. This gray image is normally in the Raw format. However, for the sake of simplicity, we simulate this camera effect with a method called *direct* in *CFA* class to reconstruct Color Array Filter image (gray image). An example is provided on the Fig 2.



Figure 2: Example for one original image (left) and one raw image (right) generated from the original image

3. Focus on *Menon* Method (Bayer pattern)

In this part, we will explain the method chosen to demosaicing CFA images with Bayer pattern. We will only explain the main

characteristics of the method, and we will not develop the theoretical process to obtain it. The research paper which explains the whole process is available in (3).

3.1. Main Process

In this section, I explain how the algorithm works. The Fig 3 shows different steps to demosaicing a raw image. First, in the function *demosaicing_CFA_Bayer_Menon2007*, for each color, we split the original mask given by the method *get_bayer_mask* from the *CFA* class. We use the function *tensor_mask_to_RGB_mask*. Originally, this mask was a tensor with three dimensions: the first one for the image width, the second one for the image length, and the last one for the color pixel intensity. We suppose by convention that we sort color pixels by order: **RGB** (Red, Green and Blue)⁴. In the first part, we transform this mask into three tensors of dimension 2 for each color. For a color-related tensor, we have for each known pixel the intensity of the color, and for the rest of the pixels we have zero values.

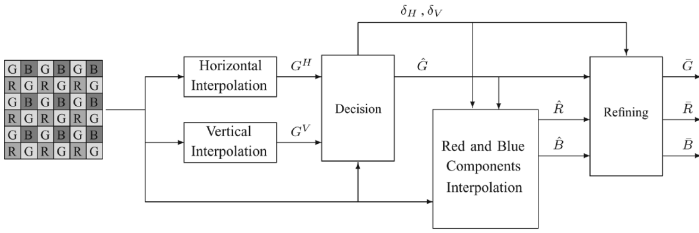


Figure 3: General Schema of Menon Method from (3)

3.2. Determine the Green Image

Then, we know that in a Bayer filter, we have twice as many green pixels as red or blue ones. That is why, firstly, we are going to determine the green image because we have more information. Secondly, we will determine red and blue images from this green prediction.

The goal of this first step is to determine the only green image by directional interpolation. With this technique, we determine the decision of where the most suitable direction of interpolations is. Then it adapts the interpolation according to this direction to make better results. To do so, we first compute estimates G^H and G^V for the green component, by a 1D-convolution according to the horizontal axis for G^H and vertical axis for G^V . The filter is a five coefficient **FIR** filter. Finite Impulse Response filter is a type of digital filter with a finite duration response to an input signal. Based on the research paper in (3), we take two filters as follows h_1 and h_2 .

$$h_1 = [0.0, 0.5, 0.0, 0.5, 0.0] \quad (2)$$

$$h_2 = [-0.25, 0.0, 0.5, 0.0, -0.25] \quad (3)$$

After the convolution of the green component both horizontally and vertically, resulting in two green images, a crucial decision must be made to choose the filtering direction that yields optimal performance. As elucidated in earlier sections, a notable characteristic of the images is the smoothness of color differences. These differences tend to change gradually, exhibiting abrupt variations primarily along edges. Consequently, an image typically exhibits higher gradient values for color differences across edges than along them. Exploiting this property allows the identification of edges in a natural image, enabling the decision-making step to determine the most suitable interpolation direction.

To determine the whole green image, we need to use the information of blue and red pixels. That is why we mix up this information with the chrominance. Within every red or blue location (where the sensor captured red or blue values), we compute the chrominance values (C^H for horizontal chrominance and C^V for vertical chrominance) as follows.

$$\text{Redpixels} : C^H = R - G^H, \text{ or } C^V = R - G^V \quad (4)$$

$$\text{Bluepixels} : C^H = B - G^H, \text{ or } C^V = B - G^V \quad (5)$$

We want to extract edges of these chrominances. So first we compute a difference between each pixel and its neighbors. To avoid any issue with boundaries, we use reflect padding on arrays. We take the absolute value of this difference because we want the value of this variation only, and it is not necessary to know the direction of this variation (positive or negative).

Then, we define a sufficiently large neighborhood with a size of (5, 5) called k according to the research paper in (3). This kernel will be used to weigh directional differences during convolution. It is possible to see the detail of k in the equation 6.

$$k = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 3.0 & 0.0 & 3.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 1.0 \end{bmatrix} \quad (6)$$

Finally, with this kernel k , we use convolution on C^H and C^V to obtain the pair of classifiers δ_H and δ_V . It provides an evaluation of the local fluctuations in color differences along the horizontal and vertical axes. Respectively, these assessments serve to estimate the orientation of edges. If the value of δ_V is less than δ_H , it suggests the presence of a horizontal edge as opposed to a vertical one. Regarding all red and blue pixels, we determine the green values based on the following criterion.

$$G = G_V : \text{if}(\delta_H > \delta_V) \text{ else } (G_H) \quad (7)$$

We can now easily determine the final version of the green image with this criterion.

In the following part, we are also going to use the classifier δ_H and δ_V because edges have been calculated for RGB colors. To find the blue image, we interpolate the color difference between R and G. For the red image, we are going to interpolate the difference between B and G.

3.3. Refinement Step

We previously obtained an image very close to the original. Nevertheless, despite a precise choice of edge directions, the reconstructed image may still exhibit various errors attributed to interpolation artifacts.

According to the research paper, (3) these errors may arise from the approximations made during filter design. Additionally, the low-pass characteristics of the filters employed for interpolating the green component and color differences can contribute to the introduction of such errors. Note that these artifacts mainly affect the regions with high-frequency content.

We suggest addressing these errors by leveraging the inter-channel correlation in the high-frequency bands of the three primary colors. A viable approach involves segregating low and high-frequency components within each pixel and substituting the high frequencies of the unknown components with those from the Bayer-known component. The low-frequency component remains

unaltered, as the low-frequency components of the color channels exhibit lower correlation. The general process can be found on the research paper(3).

4. Demosaicing Quad-Bayer Pattern

In this section, we will see how we retrieve an image from a Quad-Bayer pattern. As a reminder, we can visualize the Quad-Bayer pattern in Fig 1 (right). We can understand from this figure that a Quad-Bayer pattern is an underlined category of the Bayer pattern present in the same figure. Indeed, for each sort of pixel, we have four identical pixels which form a square area.

We already have a method that theoretically works very well (*Menon*) but only for Bayer filters. That is why, in this project, the idea was to transform a Quad-Bayer filter into a Bayer filter in order to use the same method. As we saw earlier, the structure of the Quad-Bayer suggests that we could simply downsample to transform each quadruplet of pixels of the same type into the same final pixel. This would reduce by 4 the number of pixels, i.e. a reduction in width and height by a factor of 2.

The pixels obtained would then be estimates made, for example, with an average. So, in theory, we would obtain a pixel that summarizes more information and would be a better estimate than a pixel drawn at random from the four. This would even reduce image noise. However, in our specifications, we want the same output size as the input. To achieve this, we use open-cv's *resize* function, which performs a simple interpolation to increase the size by 2. To sum up, the final pipeline chosen is: A first step of down-sampling to get back to a Bayer filter. Then we have the *Menon* method, which recolors the image. Then, we have an interpolation step to return to the same input size. We will see more about performances in the section of 5.

5. Results

In this section, we will look at possible results for the Bayer and Quad-Bayer patterns. We have tested two methods and another with the refinement variant. The first method is simple interpolation, which was coded as an example. Next, we look at the results for *Menon*'s method without refinement. Finally, we will look at the *Menon* method with the refinement step. To measure results quantitatively, we use the following three metrics: SSIM, PSNR and MSE.

- The SSIM (Structural Similarity Index) metric measures the structural similarity between two images. Since we use a method that uses edges, it is an important metric.
- The PSNR (Peak Signal-to-Noise Ratio) metric measures quality in terms of the signal-to-noise ratio.
- The MSE (Mean Squared Error) metric measures the average squared difference between pixels in two images. This metric is less informative compared to the others because it does not include the structure of the image or the noise. We will also perform this metric on each channel (R, G or B) to see the difference.

5.1. Bayer Filter

The results are summarized in Table 1 for Bayer pattern.

We can see that we obtained an SSIM score of **0.983** for the first image. This is an increase of **3.5%** and this is a good performance. However, this comes with an increase in the computational time by **39%**. We note that all other metrics show an

Paintings			
Method	Interpolation	Menon without refinement	Menon
SSIM	0.950	0.9827	0.9834
PSNR	34.63	39.82	39.95
MSE (all dimensions)	3.4×10^{-4}	1.0×10^{-4}	1.0×10^{-4}
MSE (Red Pixels)	4.1×10^{-4}	1.2×10^{-4}	1.3×10^{-4}
MSE (Green Pixels)	1.7×10^{-4}	6.9×10^{-5}	5.8×10^{-5}
MSE (Blue Pixels)	4.4×10^{-4}	1.1×10^{-4}	1.1×10^{-4}
Computational Time	1.7	2.1	2.8

Table 1: Results of *img_0* with a Bayer pattern

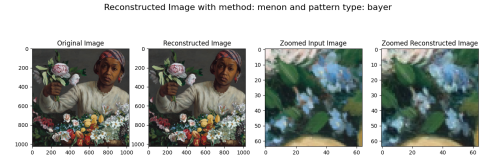


Figure 4: Demosaicing for the image called *img_1* with *Menon* method and refinement

increase in performance, and this example gives a good overview of the scores for the paintings.

We can see that the results are in line with the theory and that our final method performs best. More precisely, the *Menon* method works better than conventional interpolation. And when we activate the refinement option, we get even better results, even if it is not much. This is because in this method, we make even better use of information by managing contours and the preponderance of Green pixels. We also notice that the more complex our methods become, the longer they take. Finally, we note that we obtained a lower MSE for Green pixels. This is consistent with the fact that we have more green pixels and therefore more information to interpolate.

We obtained better results with real pictures, such as *image_5*. We have achieved an SSIM score of **0.995**. This is an increase of **4.7%**. Perhaps, the reason is that in a real picture, we can easier determine edges because textures are less blurred.

In the Fig 4, 5 and 6 we can observe the comparison of raw image and reconstructed image for two examples. We can see the result is very accurate, and it is almost impossible to see differences. In the Fig 4 or 5, we can slightly see some differences. For example, sometimes the shade is not identical. The Fig 6 is more difficult, and it is because of the SSIM score of **0.995**.

To conclude, we saw that we have been able to achieve excellent results with the *Menon* method we have implemented. We observed that the results were better on real photos than on paintings. This may be due to the fact that, with our examples, the contours were easier to determine, or the color ranges were easier to interpolate because they were more restricted (example *img_5*).

5.2. Quad-Bayer Filter

In this section, we will see the results of the Quad-Bayer filter method, and we will try to explain them.

In this part, we first study the influence of the interpolation type in the *cv2.resize* function. We see in Table 2 that for the *img_0* image test, the inter cubic method is the best for the last interpolation. That is why, this method has been chosen for the next part.

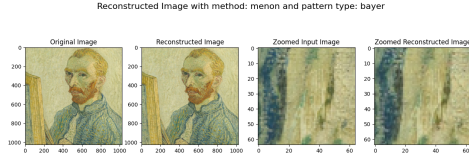


Figure 5: Demosaicing for the image called *img_2* with *Menon* method and refinement

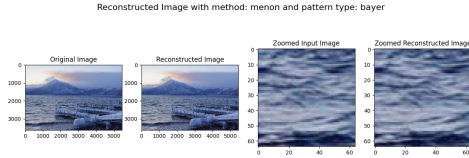


Figure 6: Demosaicing for the image called *img_5* with *Menon* method and refinement

It could be that this method works best because it takes a larger area of information (4×4) and is therefore a better estimate. We note that the *inter lanczos4* method, which takes an area of (8×8), gives similar results.

Then, still using our reference *img_0* image, we varied the method types to visualize the different results we obtained. The results are visible in the Table 3.

To begin with, we note that overall, the results for all the methods are much poorer than in the case of Bayer filter. This is also the case for the interpolation method proposed in the project. We explain this by the fact that a Quad-Bayer pattern certainly has hardware advantages. However, from an information-theoretic point of view, the color data is much more concentrated in clusters. Whereas in the Bayer filter case, the information is more diffuse. We therefore presume that during the various interpolations, regardless of the method, we have more uncertainty.

The new method for Quad-Bayer pattern is much faster than simple interpolation. One hypothesis is that since we are performing our complex algorithm (surely dynamically limiting) on an image halved in size, this makes the *Menon* method much faster because we interpolate to half the size at the end.

Next, we can see that the method gives better results than simple interpolation. We have an increase of 1.4%, which is not negligible for such high precision. However, the refinement part of the method does not really improve the results. It is hard to see why, but perhaps it is because we have reached an asymptote in terms of possible interpolations, and it is hardly possible to do better. As a reminder, it is theoretically more difficult to achieve this demosaicing because we have 4 times less information as we have divided our starting pixels by 4.

We also note that the MSE for green pixels is again less important. Again, this is an expected result, but what is surprising is that the difference is minimal with the other pixels.

Finally, we can see examples of image reconstruction in Fig 8, 9, 10 and 11. The results are very satisfactory, but when we zoom in we notice slight differences. These differences are similar to a blur applied to the image. This can be explained by the fact that, as we have down and up sampling, we lose detail in the image. In this section, we have not tested landscape images. As the sizes are not a multiple of two, there is a problem in the project code. This has not yet been fixed.

In conclusion, in this section, we have once again obtained better results for the Quad-Bayer pattern using the *Menon* method.

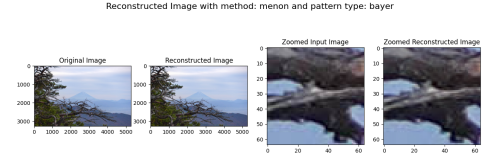


Figure 7: Demosaicing for the image called *img_6* with *Menon* method and refinement

Interpolation Methods for <i>cv2.resize</i> Function						
Name	INTER_NEAREST	INTER_LINEAR	INTER_AREA	INTER_NEAREST	INTER_CUBIC	INTER_LANCZOS4
SSIM	0.898	0.914	0.898	0.898	0.923	0.922

Table 2: Results of SSIM score for various interpolation method for the upsampling. It was *img_0* for the test.

In this part, the refinement mode is not really necessary. Furthermore, we noticed that the images were much less detailed than in the Bayer pattern part due to the conversion Quad-Bayer pattern into Bayer pattern.

6. Conclusion

To conclude this project, it was possible to achieve excellent results with a method detailed in (3). We observed that this improvement also went hand in hand with an increase in computational time for the Bayer pattern. As far as the Quad-Bayer pattern is concerned, the *Menon* method gives less good results, but this is unavoidable due to the choice of conversion between Quad-Bayer pattern and Bayer pattern.

During state-of-the-art research, deep learning techniques based on neural networks were found. These were not selected, however, as they require large databases for training. In addition, databases are often specialized and in our case, it would have been difficult to carry out our tests on both paintings and landscape photos.

References

- [1] Repository GitHub for the project, Brice Convers
- [2] Repository GitHub for the code implementation, colour-science
- [3] Research paper from IEEE which explain the Menon method, Daniele Menon; Stefano Andriani; Giancarlo Calvagno
- [4] Repository GitHub for the project framework, Mauro Dalla Mura; Matthieu Muller
- [5] Wikipedia page for Bayer filter, Wikipedia
- [6] Thesis Model Based Signal Processing Techniques for Nonconventional Optical Imaging Systems and the user's manual Pyxalis Image Viewer, Daniele Picone
- [7] National Gallery of Art, USA

Paintings			
Method	Interpolation	Menon whithout refine- ment	Menon
SSIM	0.910	0.923	0.923
PSNR	30.9	32.7	32.7
MSE (all dimen- sions)	7.9×10^{-4}	5.2×10^{-4}	5.2×10^{-4}
MSE (Red Pixels)	1.0×10^{-3}	6.0×10^{-4}	6.1×10^{-4}
MSE (Green Pixels)	2.7×10^{-4}	4.2×10^{-4}	4.1×10^{-4}
MSE (Blue Pixels)	1.1×10^{-4}	5.5×10^{-4}	5.5×10^{-4}
Computational Time (s)	157.9	2.39	2.54

Table 3: Results of *img_0* with a Quad-Bayer pattern

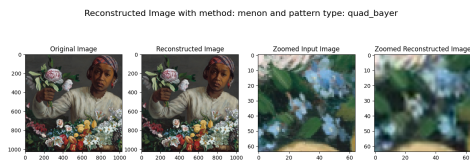


Figure 8: Demosaicing for the image called *img_1* with *Menon* method and refinement

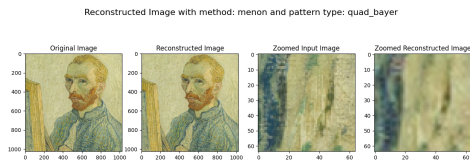


Figure 9: Demosaicing for the image called *img_2* with *Menon* method and refinement

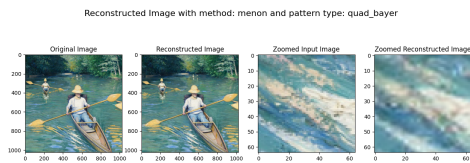


Figure 10: Demosaicing for the image called *img_3* with *Menon* method and refinement

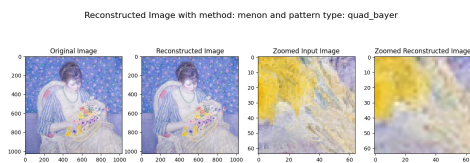


Figure 11: Demosaicing for the image called *img_4* with *Menon* method and refinement