Assignment #1
Due: Week 5, August 26, 18:00
Marks: 15 marks

OBJECTIVE

Design a C or C++ (ANSI standard) program that implements the colour image processing chain converting images captured by CMOS image sensors into true colour RGB images.

BACKGROUND

To satisfy quality requirements, images captured by semiconductor sensors have to be processed before they are passed to higher stages of computer vision systems.
You are provided with two `bmp` files `test1.bmp` and `test2.bmp`, which contain Bayer Pattern CFA data directly captured by CMOS image sensors. The images have resolution 640x480 pixels. The Bayer pattern used in the CMOS sensor is shown below:

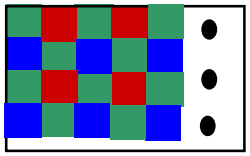


Fig. 1. Mosaic of the Bayer pattern

Although basic operations with BMP files is not the focus of this assignment and BMP file read/write functions have been implemented for you in the program template `assignment1.c`, it may be a good idea to get familiar with the BMP file format that is explained below for your information.

The `bmp` file format stores image data together with supplementary information in the structured binary file. The BMP format uses Litte Endian byte order for short int, int and unsigned int values. It contains three major segments as shown in Fig. 2.

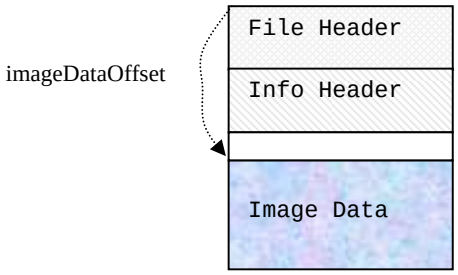


Fig. 2. The structure of BMP files.

The File Header provides general information that helps file type verification and also indicates the starting position of image data in the file. The File Header structure can be described as follows:

```
typedef struct
{
    char fileMarker1;       /* 'B' */
    char fileMarker2;       /* 'M' */
    unsigned int   bfSize;
    unsigned short unused1;
    unsigned short unused2;
    unsigned int   imageDataOffset;  /* Offset to the start of image data */
}FileHeader;
```

The first two characters 'B' and 'M' indicate that this is a BMP file and their presence must be verified before any further analysis takes place. If `'B''M'` file markers are correct, this is a BMP file and thus, the value of `imageDataOffset` indicates the start of image data in the file (see Fig. 2). However, before reading the image data, it is important to know the image width and its height.
This information is stored in the Info Header.

```
typedef struct
{
    unsigned int   biSize;
    int            width;            /* Width of the image */
    int            height;           /* Height of the image */
    unsigned short planes;
    unsigned short bitPix;             /* 24 for true colour */
    unsigned int   biCompression;
    unsigned int   biSizeImage;
    int            biXPelsPerMeter;
    int            biYPelsPerMeter;
    unsigned int   biClrUsed;
    unsigned int   biClrImportant;
}InfoHeader;
```

As images can be considered as 2D arrays, `height` indicates the number of rows and `width` indicates the number of columns. Each element of the array contains information about three color components blue, green and red which together can represent any complex color. Thus, each image data element can be described as the following structure:

```
typedef struct
{
    unsigned char  b;          /* Blue value */
    unsigned char  g;          /* Green value */
    unsigned char  r;          /* Red value */
}Pixel;
```

Note: All these structures need to be packed when they are defined (`#pragma pack(push, 1)`). Otherwise, your program may insert slack bytes.
Note: BMP files have Little Endian byte order. Test your computer platform first (compile platform platform_test.c and run it) to make sure it is also Little Endian.

Picture elements of true colour images are stored in the Image Data segment sequentially (see Fig. 3). The total number of `Pixel` elements is `width*height`.

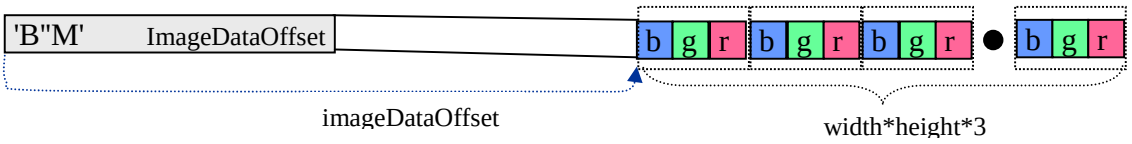


Fig. 3. The starting position and the internal structure of Image Data segment.

Picture elements of CFA images are stored in the Image Data segment sequentially too (see Fig. 4). The total number of `unsigned char` elements is `width*height` (the total number of bytes is 3 times less than for true colour images)

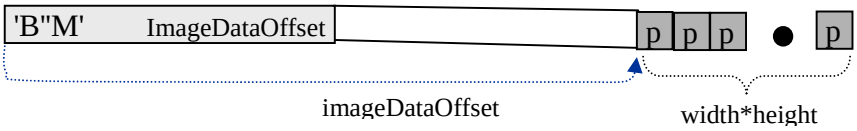


Fig. 4. The internal structure of Bayer pattern images.

DESIGN SPECIFICATION

Bayer pattern picture elements captured by the sensor have been stored in BMP files `test1.bmp` and `test2.bmp`. In this assignment you are provided with a template `C` file `assignment1.c` that:
- prompts the user to enter the input file name (`test1.bmp` or `test2.bmp`)
- reads content of a specified `bmp` file with CFA data into a 2D array
- calls the function `processCFAImage(…)` that you have to complete
- stores the produced true colour RGB image in the output `bmb` file.

There is no need to use OpenCV library for this assignment. You need to modify `assignment1.c` (or use it as a basis for your C/C++ solution) to implement a colour processing chain that includes the following stages:
- CFA Interpolation
- Colour Correction
- Gamma Correction
These stages may be implemented in the function `processCFAImage(…)`. However, it may be better to implement them as three separate functions called from `processCFAImage(…)` in the required sequence.

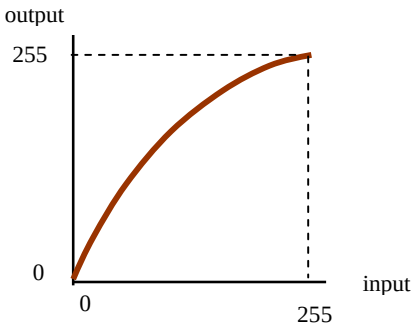
You can select any CFA interpolation algorithms for your implementation such as Nearest Neighbor, Bilinear, etc. The requirement is that it must produce an RGB image with the same width and height. The algorithms are not complex, but you need to take into account boundary conditions, as pixels around the image boundary may not have

neighboring pixels on one or two sides. Try to find the most efficient and simple solution to interpolate boundary pixels.

Colour Correction is a matrix operation. You should use the following matrix that was optimized for this CMOS image sensor:

$$\begin{bmatrix} 1.18 & -0.09 & -0.13 \\ -0.24 & 1.29 & -0.05 \\ -0.18 & -0.44 & 1.71 \end{bmatrix}$$

Gamma correction should be implemented using a look-up table as described in the lecture notes. The only difference is that the table must map 8-bit values onto 8-bit values. The table entries must be calculated using $\gamma = 0.5$ and implement the following transfer function:

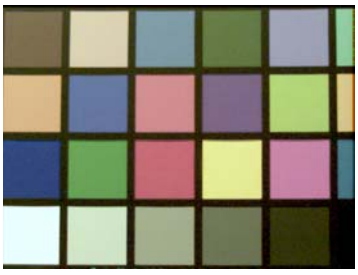


You need to find the expression how to calculate and fill the table.

The function `processCFAImage(…)` has the following prototype:

```
bool processCFAImage( unsigned char **cfaImage,
                      Pix ** rgbImage,
                      int width,
                      int height);
```

Where `width` and `height` are `cfa` (and also `rgb`) image sizes in pixels. `cfaImage` is a two dimensional array with CFA Bayer pattern values. `rgbImage` is a 2D array of `RGB` image samples that must be produced by your image processing chain. If there are no errors in the function, it returns `true`. The resulting `rgb` samples are stored in a `bmp` file. The produced `bmp` file can be displayed by any image viewer. For test1.bmp it should be similar to the image shown below



SUBMISSION

In this assignment you have to submit:
- `assignment1.c` (or `assignment1.cpp`) source file with appropriate comments
- `assignment1.txt` report

All submitted files must include an assignment header with your name, your student number and your email address. Anonymous submissions without these details will not be assessed and will not be marked.

The report must provide a one page description of the implemented colour processing chain, used algorithms, comment on image quality and conclusions. Do not add any separate `*.h` header files to your solution as it makes evaluation and marking of the assignment more complex. **Add the two files to a ".zip" archive for submission**.

On the Moodle site for the subject you will find a section with the title "Assignment Submission". Under "Assignment 1" upload your ".zip" file. You are allowed to submit up to 3 times before the deadline.

Suggestions regarding OpenCV

After submitting your assignment, you can modify your code to use OpenCV to make sure your development environment is set properly and to practice with OpenCV library. You can replace one-by-one some of the functions provided in assignment1.c with relevant functions from OpenCV to create 2D arrays, use operations with matrices, read BMP files and write into BMP files. This should simply you code.

There is no need to submit this version of your program. This exercise is helpful to get familiar with basics operations supported by OpenCV that you will need when you start working on next assignments.

NOTES:

1. Submit your assignment before the due date. Follow the submission requirements explained in the section SUBMISSION. You are allowed to submit up to 3 times before the deadline. Only the latest submission will be tested and marked.
2. SUBMISSION BY EMAIL IS NOT ACCEPTABLE
3. ASSIGNMENT FILES WITHOUT PROPERLY FILLED HEADERS WILL NOT BE MARKED
4. Enquiries about the marks can only be made within a maximum of 1 week after the assignment results are published. After 1 week the marks cannot be changed.