INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Department of Electronics & Eletrical Communication Engineering

Vision and Intelligent Systems

EC69211 – Image and Video Processing Laboratory

Experiment – 1

BMP File Format

Submitted by:

Bbiswabasu Roy (19EC39007)

Jothi Prakash (19EC39023)

CONTENTS

Objective:

1. Reading a given BMP file and producing its header information and pixel array as output
2. Taking BMP header and pixel array as input and writing it to BMP file
3. Setting a particular channel of a given image to zero and displaying the image

Theory:

A bitmap image is a raster image (containing pixel data as opposed to vector images) format. Each pixel of a bitmap image is defined by a single bit or a group of bits. Hence, it is called the bitmap or a map of bits and pixels.

A BMP file consists of 4 main parts –

File Type Data

- This block is a BMP Header labeled as BITMAPFILEHEADER. This is the starting point of the BMP file and has 14 bytes width. This header contains a total of 5 fields of variable byte width.

Image Information Data

- This is a DIB Header must be used to specify the color and image information. Unlike BITMAPFILEHEADER, there are many types of info headers. Each header has different byte-width but for compatibility reasons, we use BITMAPINFOHEADER.
- This header is 40-bytes wide and contains a total of 11 fields of variable byte widths.

Color Pallet

- This block contains the list of colors to be used by a pixel. This is an indexed table with the index starting from 0. The integer value of the pixel points to the color index in this table and that color is printed on the screen.

Raw Pixel Data

- This block contains binary numbers dedicated to representing the unique color values of each individual pixel. Depending on the bpp of the BMP image, a byte can contain color values of multiple pixels or multiple bytes can be used to represent the color value of a single pixel.

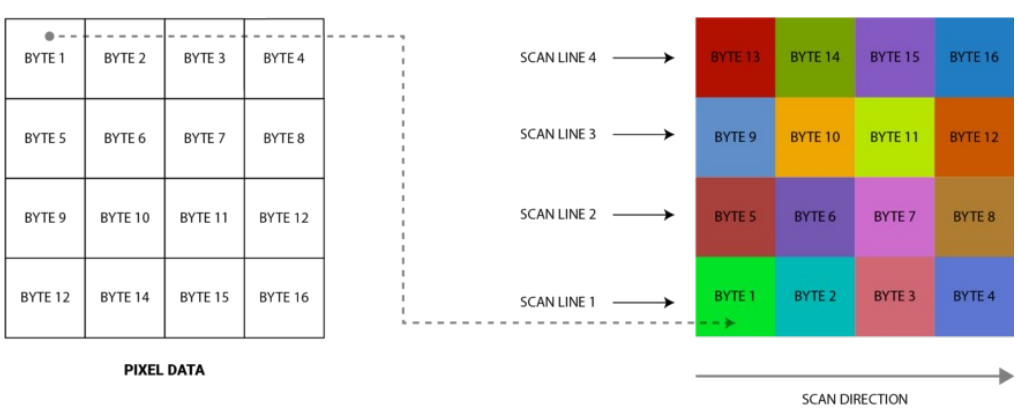
It must be noted that the pixel data stored in BMP file does not have the same ordering as the pixlels of the image to be displayed. The coordinates of a BMP image start from the bottom-left corner much like a normal graph or northeast quarter of the 2D cartesian coordinate. Hence, BMP follows the bottom-up approach for scanning.

The first pixel of the last row in the BMP image is represented by the first byte in the Pixel Data. Then the scan moves forward by mapping the next pixel with the next byte in the Pixel Data until the last pixel in that row.

The figures below show the BMP file structure and how pixel data is stored in the BMP file.

Bitmap File Structure			
Block	**Field**	**Width**	**Description**
BITMAPFILEHEADER **Fields:** 5 **Width:** 14 bytes	FileType	2 bytes	A 2 character string value in ASCII. It must be 'BM' or '0x42 0x4D'
	FileSize	4 bytes	An integer (unsigned) representing entire file size in bytes (number of bytes in a BMP image file)
	Reserved	2 bytes	To be utilized by an image processing application. Initialized to '0' integer (unsigned) value.
	Reserved	2 bytes	To be utilized by an image processing application. Initialized to '0' integer (unsigned) value.
	PixelDataOffset	4 bytes	An integer (unsigned) representing the offset of actual pixel data in bytes.
BITMAPINFOHEADER **Fields:** 11 **Width:** 40 bytes	HeaderSize	4 bytes	An integer (unsigned) representing the size of the header in bytes. It should be '40' in decimal.
	ImageWidth	4 bytes	An integer (signed) representing the width of the final image in pixels.
	ImageHeight	4 bytes	An integer (signed) representing the height of the final image in pixels.
	Planes	2 bytes	An integer (unsigned) representing the number of color planes. Should be '1' in decimal.
	BitsPerPixel	2 bytes	An integer (unsigned) representing the number of bits a pixel takes to represent a color.
	Compression	4 bytes	An integer (unsigned) representing the value of compression to use. Should be '0' in decimal.
	ImageSize	4 bytes	An integer (unsigned) representing the final size of the compressed image. Should be '0' in decimal.
	XpixelsPerMeter	4 bytes	An integer (signed). Should be set to '0' in decimal to indicate no preference of the target device.
	YpixelsPerMeter	4 bytes	An integer (signed). Should be set to '0' in decimal to indicate no preference of the target device.
	TotalColors	4 bytes	An integer (unsigned) representing the number of colors in the color pallet.
	ImportantColors	4 bytes	An integer (unsigned) representing the number of important colors. Ignored by setting '0' in decimal.
COLOR TABLE **Fields:** 4 x entries **Width:** 4 x entries	Red	1 bytes	An integer (unsigned) representing Red color channel intensity.
	Green	1 bytes	An integer (unsigned) representing Green color channel intensity.
	Blue	1 bytes	An integer (unsigned) representing Blue color channel intensity.
	Reserved	1 bytes	An integer (unsigned) reserved for other uses. Should be set to '0' in decimal
PIXEL DATA			An array of pixel values with padding bytes. A pixel value defines the color of the pixel.

semi-optional



Algorithms:

Following algorithm was used to read the BMP file header and pixel array:

1. Open the required file in read binary format
2. Keep reading appropriate number of bytes from the file as per the header format and store all the header information in dictionary bmp_header
3. If number of colors mentioned in header is non-zero, read and store the color table in bmp_header
4. Declare a pixel_array of dimension height x width x 3
5. For each pixel (i, j) in the image, do
6. If color table exists, set file pointer to the index of color table to which current pixel points. Else do not update file pointer
7. Read 3 bytes starting from where file pointer currently points and store them in pixel array location (height – i, j) as RGB (though they are read as BGR from file)
8. Return the bmp_header and pixel_array

Following algorithm was used to write the BMP file header and pixel array:

1. Open the required file in write binary format
2. Keep writing the bmp_header with appropriate number of bytes as per the header format
3. If the image has color table, write the colors with same indexing as stored in the bmp_header
4. Also, store the index of each color in a dictionary color_index
5. For each pixel (i, j) in pixel_array, do
6. Get RGB components of the image from the pixel_array[i][j]
7. If image is 8-bit color indexed format, get index of current pixel color from color_index and write the 8-bit index to where file pointer is pointing. Else, write the 24-bit color to where file pointer is pointing following the order BGR

Following algorithm was used to manipulate particular channel of *corn.bmp* file:

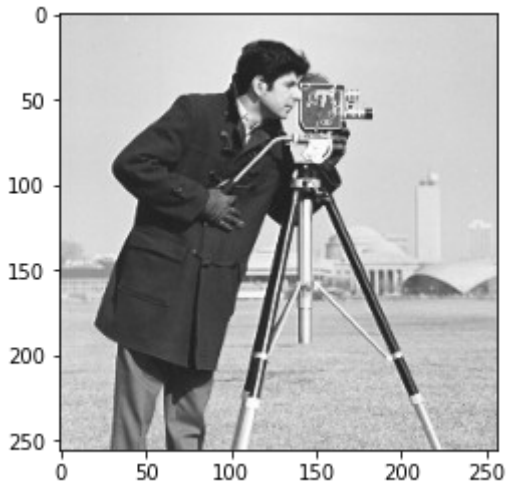
1. Take the color channel to be removed as input remove_color
2. Get bmp_header and pixel_array using read_bmp() function
3. Define a mask to be applied to each of the colors in the color table
4. For remove_color = ‘red’, mask = 00 00 FF FF. For remove_color = ‘green’, mask = 00 FF 00 FF. For remove_color = ‘blue’, mask = 00 FF FF 00.
5. Update the color table stored in bmp_header by ANDing mask with each element of color table
6. For each pixel in pixel_array, remove corresponding color component by setting that value to be 0
7. Write the manipulated image by calling write_bmp() function with updated bmp_header and pixel_array

Results:

The read_bmp() function was called for each of the given files while the following bmp headers and images were obtained as output:

```
type : BM
size_bytes : 66614
reserved_1 : 0
reserved_2 : 0
offset : 1078
dib_header_size : 40
width_pixels : 256
height_pixels : 256
colour_planes : 1
bits_per_pixel : 8
compression_method : 0
raw_image_size_bytes : 65536
horizontal_resolution : 0
vertical_resolution : 0
num_colors : 256
important_colors : 0
```

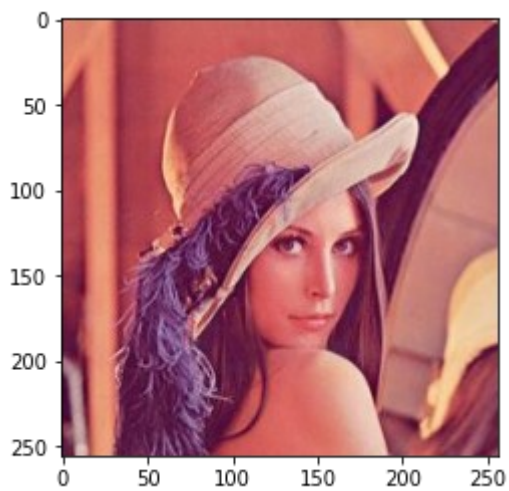
cameraman.bmp header



cameraman.bmp pixel array plot

```
type : BM
size_bytes : 196662
reserved_1 : 0
reserved_2 : 0
offset : 54
dib_header_size : 40
width_pixels : 256
height_pixels : 256
colour_planes : 1
bits_per_pixel : 24
compression_method : 0
raw_image_size_bytes : 196608
horizontal_resolution : 3780
vertical_resolution : 3780
num_colors : 0
important_colors : 0
```

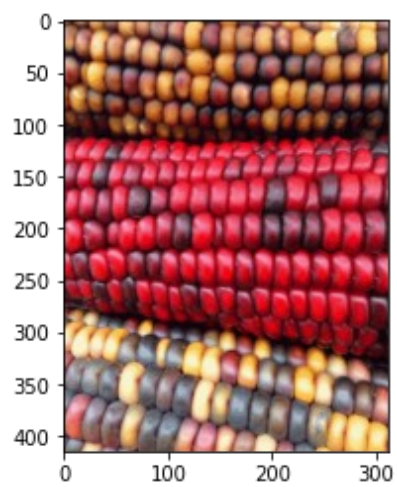
lena_colored_256.bmp header



lena_colored_256.bmp pixel array plot

```
type : BM
size_bytes : 130558
reserved_1 : 0
reserved_2 : 0
offset : 1078
dib_header_size : 40
width_pixels : 312
height_pixels : 415
colour_planes : 1
bits_per_pixel : 8
compression_method : 0
raw_image_size_bytes : 129480
horizontal_resolution : 0
vertical_resolution : 0
num_colors : 256
important_colors : 0
```

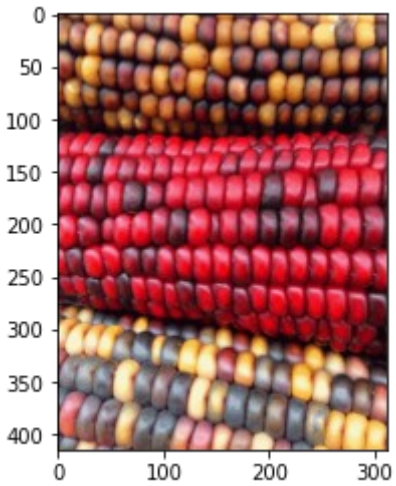
corn.bmp header



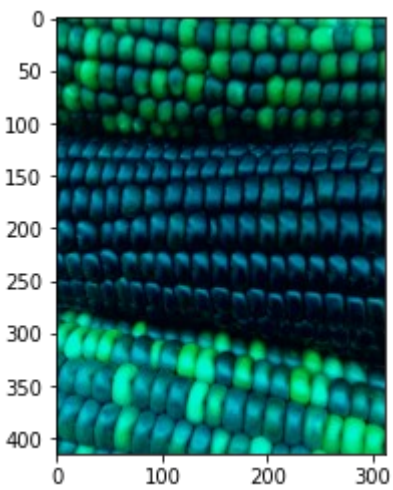
corn.bmp pixel array plot

The write_bmp() function was called by passing the headers and pixel arrays obtained from read_bmp() function for given images. The output images generated by write_bmp() function were found to be exactly same as the input images, thereby confirming that the function was implemented properly.

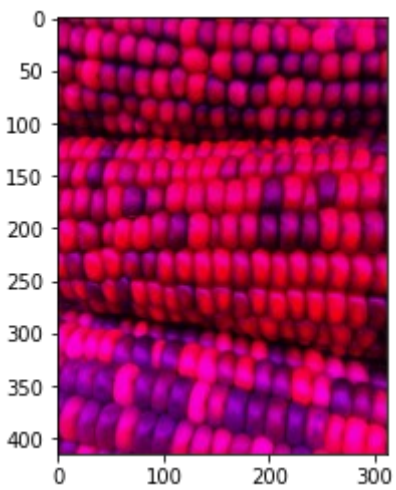
The file *corn.bmp* was read using the read_bmp() function after which a particular channel was set to all zeros and the output data was written to new file using write_bmp() function. The output images thus obtained are as follows:



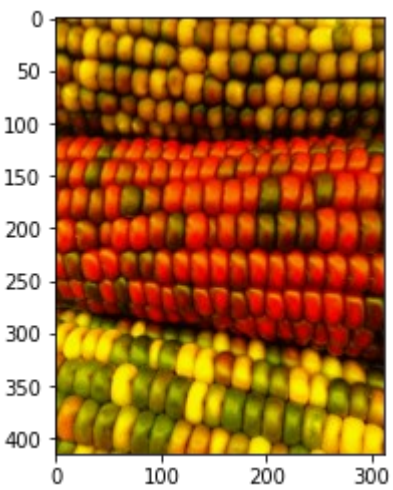
Original corn.bmp



corn.bmp with red=0



corn.bmp with green=0



corn.bmp with blue=0

Discussion:

- Although the image *cameraman.bmp* was seen as a grayscale image, fundamentally there was no difference in the header or data format of *cameraman.bmp* and *corn.bmp*. Both files were actually 8-bit indexed image format where color table of *cameraman.bmp* contained only colors with equal RGB components (which is equivalent to grayscale intensity levels) while that of *corn.bmp* contained colors with unequal RGB components giving rise to non-gray colors.
- If number of different colors used in image is much smaller compared to number of pixels in image, color-indexed image format can help in reducing the size of the file to great extent. As an example, we consider an 512x512 image which uses 256 different RGB colors. If we store the image in 24-bit color format, the pixel array will take (512*512*3 B =) 768 kB while if we store it in 8-bit color indexed format, it will take (4*256 + 512*512*1 B =) 257 kB.
-