Image Processing Laboratory

Experiment 1: BMP File Format

Report

Group-6

Submitted by

Deep Kiran Shroti (12EC35013)
Vishnu Dutt Sharma (12EC35018)

Contents

1 Introduction

The objective of the experiment is to read a Bitmap image file(grayscale or RGB), convert it to a grayscale image, if not one already and then writing it as image on the disk after flipping along the diagonal. This experiment was done without the help of OpenCV, using only library function in C++, in order to understand how image files are created and how they are manipulated and operated upon.
Bitmap or **BMP** files are quite old file format used by "Windows" operating system. BMP images can range from 1 bit per pixel (thus a black and white image) to 24 bits per pixel (providing 1.67 million colours). In the experiment we used an 8 bits per pixel (Grayscale image) and 12 bits per pixel (RGB color image) formats for operating upon.
Following are two parts of a BMP file:

1. **Header:** It contains information about file and image. This part can be broken into two parts:

   (a) **File Header**, which contains general information related to the file like type of the image file (**BM** for the Bitmap file) and Size of the file. Other fields are reserved and are not to be edited by the user.

Bitmap File Header			
Offset (hex)	Offset (dec)	Size (bytes)	Purpose
00	0	2	The header field used to identify the BMP and DIB file is *0x42 0x4D* in hexadecimal, same as BM in ASCII. The following entries are possible: • **BM** Windows 3.1x, 95, NT, ... etc. • **BA** OS/2 struct bitmap array • **CI** OS/2 struct color icon • **CP** OS/2 const color pointer • **IC** OS/2 struct icon • **PT** OS/2 pointer
02	2	4	The size of the BMP file in bytes
06	6	2	Reserved; actual value depends on the application that creates the image
08	8	2	Reserved; actual value depends on the application that creates the image
0A	10	4	The offset, i.e. starting address, of the byte where the bitmap image data (pixel array) can be found.

Table 1: Description of Bitmap file header contents

(b) **Information Header(BITMAPINFOHEADER)** , which contains information about the image, like *Width*, *Height* and *Bits per pixel* among other data.

Bitmap Information Header Windows BITMAPINFOHEADER			
Offset (hex)	Offset (dec)	Size (bytes)	Purpose
0E	14	4	the size of this header (40 bytes)
12	18	4	the bitmap width in pixels (signed integer)
16	22	4	the bitmap height in pixels (signed integer)
1A	26	2	the number of color planes (must be 1)
1C	28	2	the number of bits per pixel, which is the color depth of the image. Typical values are 1, 4, 8, 16, 24 and 32.
1E	30	4	the compression method being used. See the next table for a list of possible values
22	34	4	the image size. This is the size of the raw bitmap data; a dummy 0 can be given for BI_ RGB bitmaps.
26	38	4	the horizontal resolution of the image. (pixel per meter, signed integer)
2A	42	4	the vertical resolution of the image. (pixel per meter, signed integer)
2E	46	4	the number of colors in the color palette, or 0 to default to 2n
32	50	4	the number of important colors used, or 0 when every color is important; generally ignored

Table 2: Description of Bitmap info header contents

2. **Image Data:** It contains the pixel data or the color table contents which are to be manipulated to transform the image. The data starts from the address stored in the *offset* field of the BITMAPINFOHEADER. It was observed that for grayscale images, offset was generally 1078, while for RGB color images it was 54. The data is stored *Bottom-to-Top* and *Left-to-Right*, *i.e.* the data is stored in rows which start filling at bottom first and then keep filling to the top. The row size(in bytes) should be divisible by 4, otherwise it should be padded with zeros such that the row size become divisible by 4.

$$\text{RowSize} = \left\lfloor \frac{\text{BitsPerPixel}\cdot\text{ImageWidth}+31}{32} \right\rfloor \cdot 4$$

For flipping, contents are swapped about the diagonal of the image data while, for the color to grayscale conversion, grayscale value is calculated as,

$$Grayscale = R \times 0.30 + G \times 0.59 + B \times 0.11$$

2 Algorithm

There are three main operations int the program:

- **ReadBMP:** Function to read the BMP file and return the header. The header is saved in a data structure and the image data is loaded into memory after dynamic allocation of the memory.
  Step 1: Reading the header
  Step 2: Allocating size to the array according to the information given in the header
  Step 3: Loading the data in the memory
  Step 4: Return the header During these operations, we also read the data that may appear as unknown (*e.g.* between end of the header and start of the image data OR after the image data), so that during writing, we can use them again to avoid error in writing image.

```
ReadBMP(string inputFile){

        FILE *fptr; // File pointer
        BMPHEADER header; // File and Info header
        int paddingSize, channel;
        // paddingSize: Width of the image (including number of bits
        //     to be padded)
        // channel: Number of channel: 1 for grayscale, 3 for RGB

        // Reading the file
        fptr = fopen( inputFile.c_str(), "rb");
        cout<<"FileName␣:␣"<< inputFile<<endl;

        // If error occured in opening file
        if(fptr == NULL){
                cout<<"File␣Not␣found"<<endl;
                return header;
        }

        // Reading the header data
        fread( &header, sizeof(unsigned char), sizeof(BMPHEADER), fptr);


        // Counting the number of channels/Color depth
        channel = header.bpp/8;

        // Reading the extra data stored between header and pixel
        // data/color table
        extra = new unsigned char[ header.offset - sizeof(BMPHEADER) ];
        fread(&extra,sizeof(unsigned char),header.offset-sizeof(BMPHEADER),fptr);

        // Reading the pixel data/color table
        // Step 1: Allocating size
        pixelArray = new unsigned char**[ channel ];
```

```
for(int i = 0; i < channel; i++){
        pixelArray[i] = new unsigned char*[ header.height] ;

        for(int j = 0; j < header.height; j++)
                pixelArray[i][j] = new unsigned char[ header.width ];
}

// Calculating width/row size with padded size
paddingSize = header.width * channel ;
while( paddingSize % 4 )
        paddingSize++;

// Step 2: Reading the pixel data/color table

// In Bitmap file, data is stored starting from bottom-to-top,
// left-to-right
for(int i = header.height - 1; i >= 0 ; i-- ){
        // Going to the start of each row
        fseek( fptr, header.offset + paddingSize*i, SEEK_SET );

        // Reading the pixel data/color table
        for(int j = 0; j < header.width; j++ ){
           for( int k = channel - 1; k >= 0; k--){
                 fread(&(pixelArray[k][i][j]),sizeof(unsigned char) ...
                                ... , sizeof(unsigned char), fptr );
                }
        }
}

// Adding data from image/color table
fread( &footer, sizeof(unsigned char), header.size - header.offset ...
                ... - (channel * header.width * header. height ), fptr );

// Closing the file
fclose(fptr);

// Returning the header
return header;
```

- **ConvertFlipGrayscale:** We use the formula mentioned in the introduction section and convert the image (if RGB) to grayscale. Then using simple swapping, we flip the image along the diagonal.
  Step 1: If RGB image, calculate intensity for grayscale
  Step 2: Flip the first channel(R) along the diagonal and store in an array
  Step 3: If RGB, assign same value to remaining two channelsas in the first channel Step 4: Interchange the value of *Width* and *Height* in the header Step 4: Return the array thus

obtained

```
ConvertFlipGrayscale( BMPHEADER header){

        int channel, tempSwap, gray;
        // channel: Color Depth, 1 for grayscale, 3 for RGB
        // tempSwap: variable to help in the swapping for flipping
        //    (not being used here)
        // gray:  varible for storing grayscale value
        unsigned char temp; // variable for helping in swapping value
        //     of width and height in header

        // Finding number of channels/Color depth, 1 means 8bpp/grayscale,
        // 3 means 24bpp/RGB color image
        channel = header.bpp/8;

        // New array for storing grayscale converted and flipped image
        unsigned char ***myPixelArray = new unsigned char**[ channel];

        // Flipping pixel data/color table along the diagonal
        for(int i = 0; i < channel; i++ ){
                myPixelArray[i] = new unsigned char*[ header.width ];

                for(int j = 0; j < header.width; j++)
                        myPixelArray[i][j] = new unsigned char[ header.height ];
        }

        // Grayscale conversion for RGB color image
        if(channel == 3 )
                for(int i = 0; i < header.height; i++){
                        for(int j = 0; j < header.width; j++){

                                // Grayscale data = R * 0.30 + G * 0.59 + B * 0.11
                                gray = (pixelArray[0][i][j] * 0.30) +
                                          (pixelArray[1][i][j] * 0.59) +
                                          (pixelArray[2][i][j] * 0.11);

                                // Giving same value to all channels
                                pixelArray[0][i][j] = gray;
                                pixelArray[1][i][j] = gray;
                                pixelArray[2][i][j] = gray;
                        }
                }

        // Flipping the image
        for(int i = 0; i < header.height; i++)
           for(int j = 0; j < header.width; j++)
```

```
                myPixelArray[0][i][j] = ...
                    ... pixelArray[0][header.width- j-1][header.height-i- 1];


            // Assigning same value for all channels
            if( channel == 3 ){
                    myPixelArray[1] = myPixelArray[0];
                    myPixelArray[2] = myPixelArray[0];
            }

            // Swapping value of the width and height
            tempSwap = header.width;
            header.width = header.height;
            header.height = tempSwap;

            // Returning the pixel array
            return myPixelArray;
    }
```

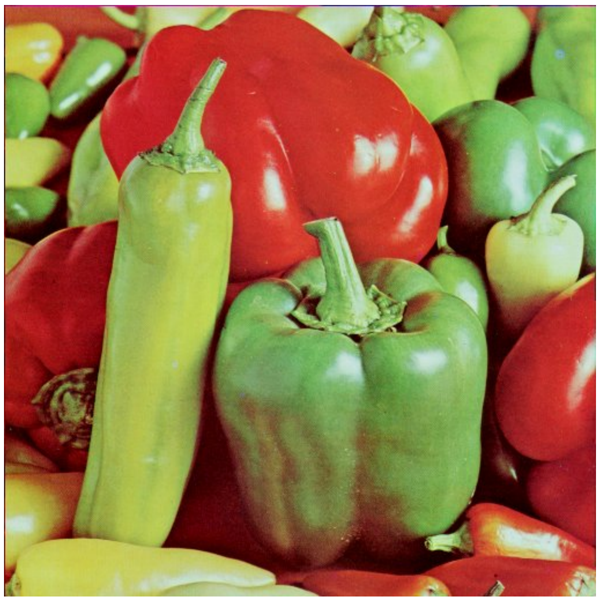
- **WriteBMP:** For writing the image, we use the header and the extra data to get a correct image. Only the width, height and the pixel data can be different from the original image.
  Step 1: Create new output file in writing mode
  Step 2: Write the header given as input after gray-scale conversion and flipping
  Step 3: Write the new image data and extra data(if any)

```
    WriteBMP(string outFile, BMPHEADER header){
            FILE *fptr;  //File pointer
            int channel, paddingSize;

            // channel: Number of channel: 1 for grayscale, 3 for RGB
            // paddingSize: Width of the image (including number of bits
            //     to be padded)

            // Finding number of channels/Color depth, 1 means 8bpp/grayscale,
            // 3 means 24bpp/RGB color image
            channel = header.bpp/8;

            // opening the Bitmap file
            outFile = outFile + ".bmp";
            fptr = fopen( outFile.c_str(),"wb");

            // Writing header to the file
            fwrite( &header, sizeof( unsigned char ), sizeof(BMPHEADER), fptr);
            // Writing extra data (that lies between end of headr and
            // offset) to the file
            fwrite(&extra,sizeof(unsigned char),header.offset-sizeof(BMPHEADER), ...
            ... fptr);
```

```
        // Finding the image width/row size including the number of
        // bytes to be padded
        paddingSize = header.width * channel;
        while( paddingSize % 4 )
                paddingSize++;

        // Writing the pixel data/color data to the image
        // In Bitmap file, data is stored starting from bottom-to-top,
        // left-to-right
        for(int h = header.height - 1 ; h >= 0 ; h--)
                {
                        // Going to the start of each row
                        fseek( fptr, header.offset + paddingSize*h, SEEK_SET );

                        // Writing data to rows in each channel
                        for(int w = 0; w < header.width; w++){
                                // Writing data in each channel by iterating
                           for( int k = channel - 1; k >= 0 ; k--)
                                 fwrite(&(pixelArray[k][h][w]),
                                                 sizeof(unsigned char),
                                                 sizeof(unsigned char),fptr);
                        }
                }

        // Writing extra data (if any, from original image ),
        // after the color table
        fwrite( &footer, sizeof(unsigned char), header.size - header.offset - ...
                ...( channel * header.width * header.height ), fptr );

        // Closing the file
        fclose(fptr);

}
```

3 Output Results

Following are the results for a colour and gray-scale images:


(a) Input Image

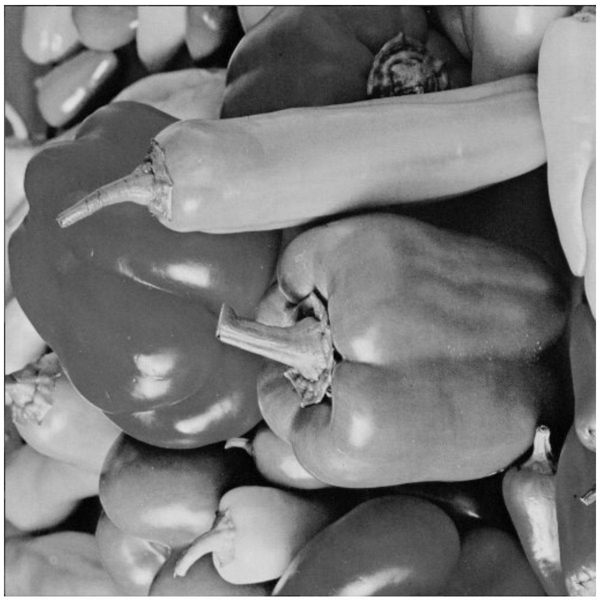
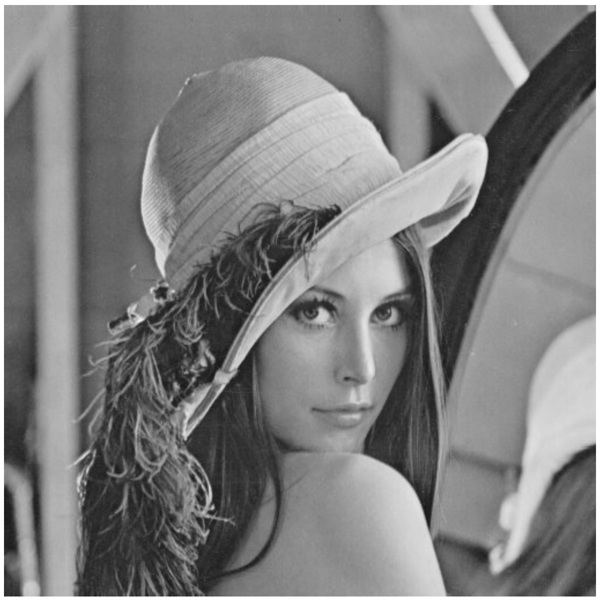

(b) Output Imgae

Figure 1: Results for a colour image


(a) Input Image

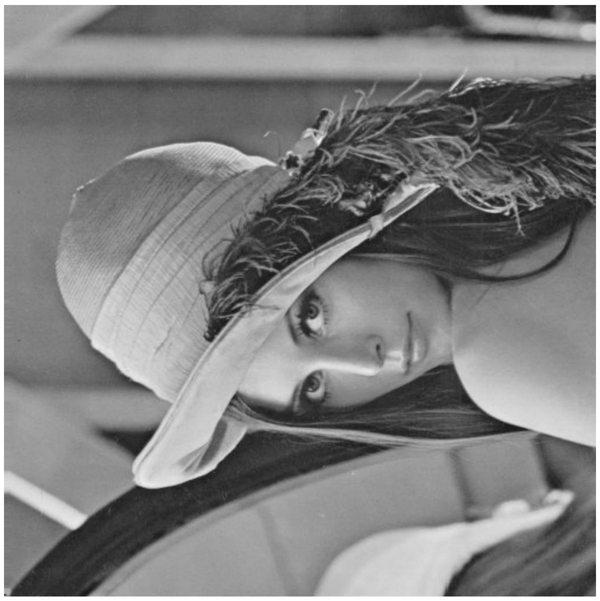

(b) Output Imgae

Figure 2: Results for a gray-scale image

4 Analysis

- The BMP file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device.
- BMP file format is well defined, the file format consists of file header, information header, and the pixel array. BMP file header gives important information such as height, width, size, offset and number of bits per pixel.
- Pixel array starts from different location in case of Red-Blue-Green(RGB) images and Gray-scale images. In case of RGB image the pixel array location is from 54 where as in case of Gray-scale images it is from 1078.
- The size of images can be calculated as *Header_Size + Channels*Height*Width* This holds true in case of RGB i.e. 54+3*512*512 which is equal to size of the image,
  but in case of some gray-scale the calculated size i.e. 1078 + 512*512 and the actual size has a difference of two bytes.
- Therefore, for output images we have used the same file header and just altered the pixel array, rest of the file data has been copied as it is.

5 Sources

[1] Wikipedia page on BMP File Format
`https://en.wikipedia.org/wiki/BMP_ file_ format`

[2] BMP image format, Paul Bourke, July 1998
`http://paulbourke.net/dataformats/bmp/`