Image Processing Lab

Batch 9:

Ajay Viswanathan [09EC3509]

Sourin Sutradhar [09EC3514]

Experiment 1
BMP File Format

Problem Objective:

Write C/C++ modular functions to read / diagonally flip / write BMP image files. All functions must support at least 24-bit RGB, and 8-bit grayscale image formats.

(a) *ReadBMP*:
a) Input: Filename of input image
b) Output: BMP header structure, Image pixel array loaded onto memory

(b) *ConvertFlipGrayscale*:
a) Input: Image pixel array
b) Output: Grayscale-converted and diagonally flipped (transposed) pixel array

(c) *WriteBMP*:
a) Input: Filename of output (grayscale) image, BMP header structure, Image pixel array.
b) Output: BMP file written on disk

Use the above functions to read a 256 × 256 24-bit RGB coloured Lena image, and write it as an 8-bit grayscale onto a different file on the disk.

Brief Theory:

Bitmaps are defined as a regular rectangular mesh of cells called pixels, each pixel containing a colour value. They are characterised by only two parameters, the number of pixels and the information content (colour depth) per pixel. There are other attributes that are applied to bitmaps but they are derivations of these two fundamental parameters.

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.

The BMP file format is capable of storing 2D digital images of arbitrary width, height, and resolution, both monochrome and colour, in various colour depths, and optionally with data compression, alpha channels, and colour profiles.

Structure of BMP file

Bitmap file in RAM or ROM contains a header, which should be omitted (we do not check it, assuming that the bitmap already has the required format). After the header there is data section, containing information on pixels' colours. Single byte (8 bits) contains colour information on two pixels, 4 MSB concerns pixel on the left side, 4 LSB - pixel on the right side. Colour is encoded as a 4-bit address in colour table (which can be found in BMP header).

The order of the pixels in BMP file is as follows: from left to right, from bottom to top (first pixel is from lower left corner of the picture). In the first approach, the picture can be displayed upside down, just to test the reading data from memory.

Each line is filled with zeros at the end, so each line has a length of multiple of 32 bits. In this example filling is not used, since each line has 256 pixels, i.e. exactly 32 groups of 32 bits.

Basic BMP File Format			
Name		**Size**	**Description**
Header		14 bytes	Windows Structure: BITMAPFILEHEADER
	Signature	2 bytes	'BM'
	FileSize	4 bytes	File size in bytes
	reserved	4 bytes	unused (=0)
	DataOffset	4 bytes	File offset to Raster Data
InfoHeader		40 bytes	Windows Structure: BITMAPINFOHEADER
	Size	4 bytes	Size of InfoHeader =40
	Width	4 bytes	Bitmap Width
	Height	4 bytes	Bitmap Height
	Planes	2 bytes	Number of Planes (=1)
	BitCount	2 bytes	Bits per Pixel 1 = monochrome palette. NumColors = 1 4 = 4bit palletized. NumColors = 16 8 = 8bit palletized. NumColors = 256 16 = 16bit RGB. NumColors = 65536 (?) 24 = 24bit RGB. NumColors = 16M
	Compression	4 bytes	Type of Compression 0 = BI_RGB no compression 1 = BI_RLE8 8bit RLE encoding 2 = BI_RLE4 4bit RLE encoding
	ImageSize	4 bytes	(compressed) Size of Image It is valid to set this =0 if Compression = 0
	XpixelsPerM	4 bytes	horizontal resolution: Pixels/meter
	YpixelsPerM	4 bytes	vertical resolution: Pixels/meter
	ColorsUsed	4 bytes	Number of actually used colors
	ColorsImportant	4 bytes	Number of important colors 0 = all
ColorTable		4 * NumColors bytes	present only if Info.BitsPerPixel <= 8 colors should be ordered by importance
	Red	1 byte	Red intensity
	Green	1 byte	Green intensity
	Blue	1 byte	Blue intensity
	reserved	1 byte	unused (=0)
	repeated NumColors times		
Raster Data		Info.ImageSize bytes	The pixel data

Pixel storage

The bits representing the bitmap pixels are packed in rows. The size of each row is rounded up to a multiple of 4 bytes (a 32-bit DWORD) by padding.

For images with height > 1, multiple padded rows are stored consecutively, forming a Pixel Array.
The total number of bytes necessary to store one row of pixels can be calculated as:

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

ImageWidth is expressed in pixels.

The total amount of bytes necessary to store an array of pixels in an n bits per pixel (bpp) image, with 2n colors, can be calculated by accounting for the effect of rounding up the size of each row to a multiple of a 4 bytes, as follows:

$$\mathrm{PixelArraySize} = \mathrm{RowSize} \cdot |\mathrm{ImageHeight}|$$

ImageHeight is expressed in pixels. The absolute value is necessary because ImageHeight can be negative
The total bitmap image file size can be approximated as:

$$\mathrm{FileSize} \approx 54 + 4 \cdot 2^{\mathrm{bpp}} + \mathrm{PixelArraySize}$$

For BPP ≤ 8 (because for pixels larger than 8 bits, the palette is not mandatory)

Only images with 8 or fewer bits per pixel must account for the palette. 16bpp images (or higher), may omit the palette part from the size calculation, as follows:

$$\mathrm{FileSize} \approx 54 + \mathrm{PixelArraySize}$$

For Bits per Pixel > 8.

In the formulas above, the number 54 is the combined size of the 14 byte bitmap file header and the 40 byte popular WindowsDIB header – the BITMAPINFOHEADER (some other DIB header versions will be larger or smaller than that as described by the table above) and the expression $4*2^n$ is the size of the colour palette in bytes.

This total file size formula is only an approximation, since the size of the colour palette will be $3*2^n$ bytes for the OS/2 DIB header version BITMAPCOREHEADER, and some files may define only the number of colours needed by the image, potentially fewer than 2^n.

Experimental Results

Input: Image pixel array

Output: Grayscale-converted and diagonally flipped (transposed) pixel array

Discussion and Inferences:

1) Windows bitmap files are stored in a device-independent bitmap (DIB) format that allows Windows to display the bitmap on any type of display device. The term "device independent" means that the bitmap specifies pixel colour in a form independent of the method used by a display to represent colour. The default filename extension of a Windows DIB file is .bmp.

2) Our code is highly modular. We have made provisions for an 8-bit image to run this program and flip it, while also allowing 24-bit images to be converted to grayscale and flipped.

3) We always use fseek before writing image pixel data. This is to account for the various optional headers that may be present in the file and to avoid overwriting over them. Although in the final output file we ignore all those optional headers and just retain the file header, bitmap header and colour table along with the pixel data.

4) Advantages of bitmap files include the following:

    - Bitmap files may be easily created from existing pixel data stored in an array in memory.
    - Retrieving pixel data stored in a bitmap file may often be accomplished by using a set of coordinates that allows the data to be conceptualized as a grid.
    - Pixel values may be modified individually or as large groups by altering a palette if present.
    - Bitmap files may translate well to dot-format output devices such as CRTs and printers.

5) Bitmap files, however, do have drawbacks:

    - They can be very large, particularly if the image contains a large number of colours. Data compression can shrink the size of pixel data, but the data must be expanded before it can be used, and this can slow down the reading and rendering process considerably. Also, the more complex a bitmap image (large number of colours and minute detail), the less efficient the compression process will be.

    - They typically do not scale very well. Shrinking an image by decimation (throwing away pixels) can change the image in an unacceptable manner, as can expanding the image through pixel replication. Because of this, bitmap files must usually be printed at the resolution in which they were originally stored.