INDIAN INSTITUTE OF TECHNOLOGY

KHARAGPUR

IMAGE PROCESSING LABORATORY

A REPORT ON

EXPERIMENT 01

Reading, Writing and Transformations of BMP Images

Name:	Prerana Priyadarshini	Sattenapalli Homa Surya Sagar
Roll. No.:	17EC35019	17EC35021

25.01.2021

Group No. 21

DEPT OF ELECTRONICS AND ELECTRICAL COMMUNICATION ENGINEERING

VISUAL INFORMATION AND EMBEDDED SYSTEMS

Table of Contents

Introduction

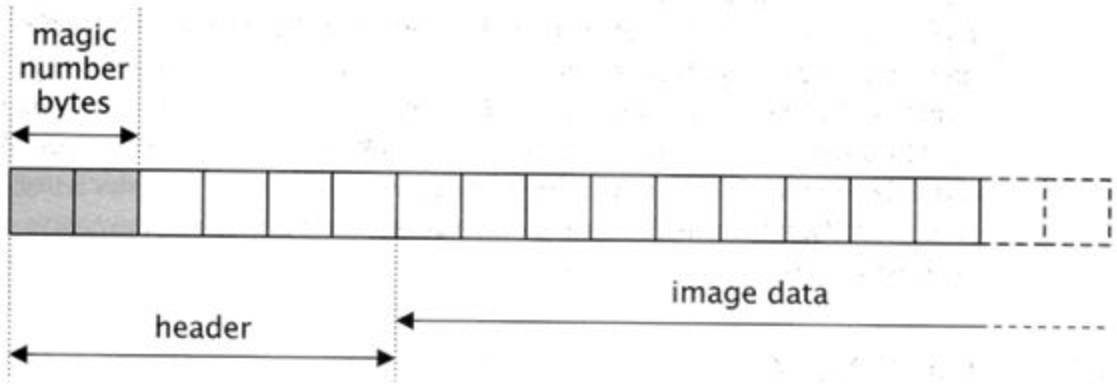
The objective of this experiment is to write C/C++ modular functions to read, perform transformations, and then write BMP image files, without using OpenCV. All functions must support 24-bit RGB and 8-bit grayscale image formats.

Bitmap File Format is a device independent file format used in Windows OS. The file format is well defined and does not change from device to device. It mainly consists of a Header section and the Image Data.

Header Section: Consists of two subsections - Bitmap File Header
Information Header or DIB Header

Structure name	Optional	Size	Purpose	Comments
Bitmap file header	No	14 bytes	To store general information about the bitmap image file	Not needed after the file is loaded in memory
DIB header	No	Fixed-size (7 different versions exist)	To store detailed information about the bitmap image and define the pixel format	Immediately follows the Bitmap file header
Color table	Semi-optional	Variable size	To define colors used by the bitmap image data (Pixel array)	Mandatory for color depths ≤ 8 bits
Pixel array	No	Variable size	To define the actual values of the pixels	The pixel format is defined by the DIB header or Extra bit masks. Each row in the Pixel array is padded to a multiple of 4 bytes in size

Most headers begin with a signature or magic number - a short sequence of bytes for identifying the file format.



Bitmap File Header:

Offset hex	Offset dec	Size	Purpose
00	0	2 bytes	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 bytes	The size of the BMP file in bytes
06	6	2 bytes	Reserved; actual value depends on the

			application that creates the image, if created manually can be 0
08	8	2 bytes	Reserved; actual value depends on the application that creates the image, if created manually can be 0
0A	10	4 bytes	The offset, i.e. starting address, of the byte where the bitmap image data (pixel array) can be found.

Bitmap Info Header:

Offset (hex)	Offset (dec)	Size (bytes)	Windows `BITMAPINFOHEADER`[2]
0E	14	4	the size of this header, in bytes (40)
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 metre, signed integer)
2A	42	4	the vertical resolution of the image. (pixel per metre, signed integer)
2E	46	4	the number of colors in the color palette, or 0 to default to 2^n
32	50	4	the number of important colors used, or 0 when every color is important; generally ignored

Color Table: The color table (palette) occurs in the BMP image file directly after the BMP file header, the DIB header. Therefore, its offset is the size of the BITMAPFILEHEADER plus the size of the DIB header (plus optional 12 bytes for the three bit masks).

Pixel Table:

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 above 1, multiple padded rows are stored consecutively, forming a Pixel Array.

Formula for calculating Row Size:

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

The formula for calculating grayscale intensities of a RGB image:

$$\textit{Grayscale} = R * 0.30 + G * 0.59 + B * 0.11$$

Algorithm

ReadBMP:

Reading the header and image data. Read and return header data. Stores the image data into memory allocated dynamically.

1. Information retrieved from the header is stored and displayed.
2. Using the dimensions a new pixel array is created and intensities are allocated from the image data
3. Unknown data is also read to avoid any errors while writing the output image.

TransformBMP:

1. **Gray Scale:** Intensity was calculated using the conversion formula if the image is RGB. After all other conversions, the same intensity values were assigned to all channels
2. **Flip Diagonally:** New array created with required dimensions. Pixels assigned with intensities flipped along the diagonal.
3. **Rotate 90 degrees:** New pixel array with Width = Header-> Height and Height = Header -> Width was created. Sequentially pixels were assigned from top left to bottom right.
4. **Rotate 45 degrees:** New pixel array with (Width = Height = Header->Height + Header Width - 1) was created. Sequentially pixels were assigned from top left to bottom right in a diagonal manner. The gaps were filled with nearest neighbor interpolation.
5. **Scale Twice:** New pixel array with 2*Width and 2*Height was created. Each pixel intensity value was assigned to 4 pixels in the new array.

WriteBMP:

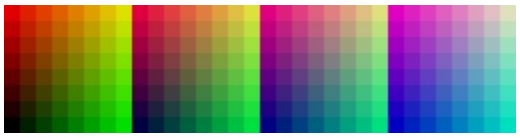
The header and unknown data from the original image is used along with the transformed pixel array data. Any dimensional changes were taken care of in TransformBMP.

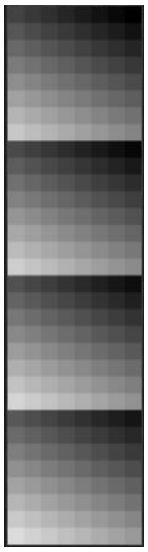
1. Output file is created in writing mode.
2. Header data was written to the output file.
3. Transformed pixel and any extra data read was written to the file.

Results

Gray Scaling

Flipping Diagonally:





90-degree rotation anticlockwise:

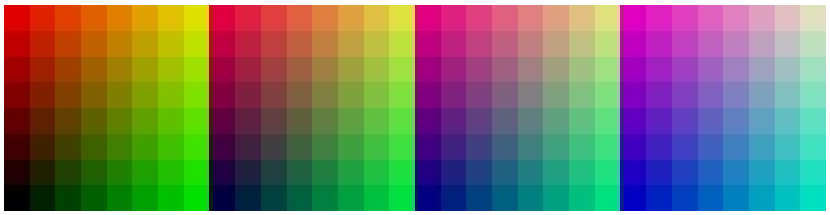
45-degree rotation anticlockwise:

Original Image

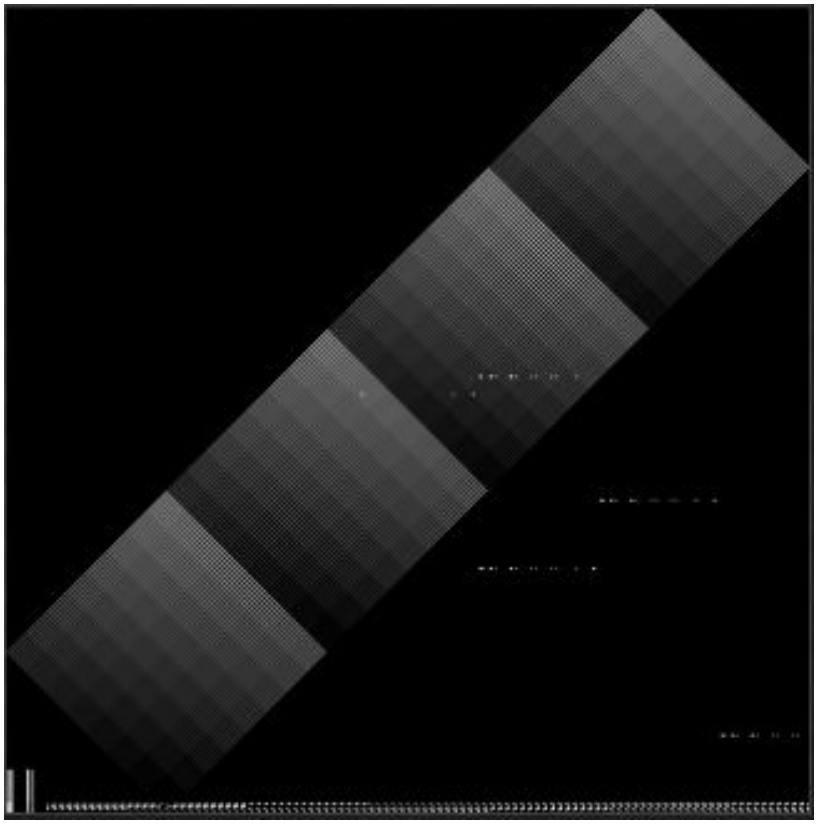
After 45 degree rotation

After Interpolation

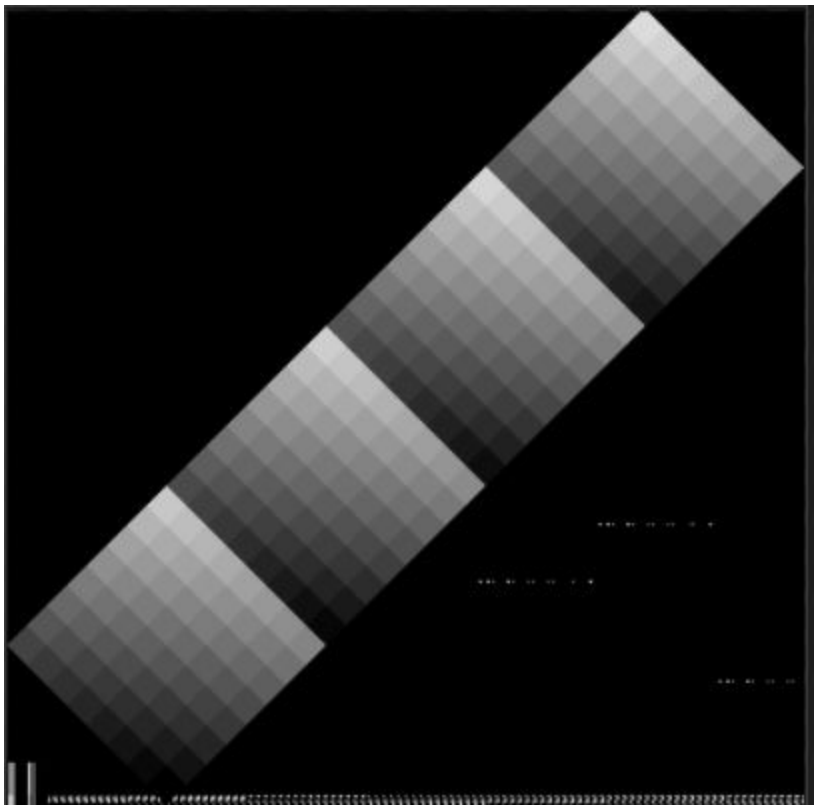
Original Image



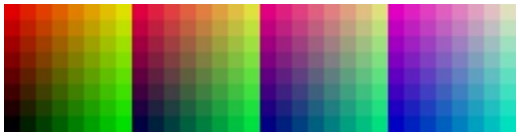
After 45 degree rotation:

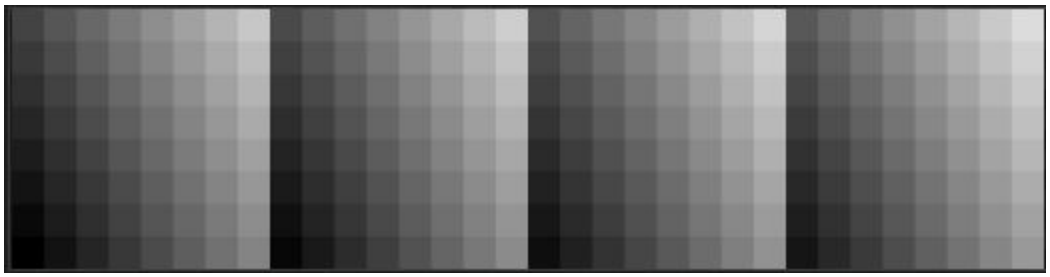


After Interpolation:



Scaled Twice:





Analysis

1. Bitmap file format or Bitmap Image File or Device Independent Bitmap (DIB) File Format is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter), especially on Microsoft Windows and OS/2 operating systems.
2. The File Format is capable of storing both monochrome and colored images in various depths.
3. The bitmap image file consists of fixed-size structures (headers) as well as variable-sized structures appearing in a predetermined sequence.
4. The BMP file format consists of Bitmap File Header, DIB Header and a Pixel Array compulsorily. It might also contain optional components like Extra Bit Masks and Gaps.
5. 45 degree rotation of the image creates pixel gaps that are required to be filled using nearest neighbor interpolation. One of the 4 filled neighbors of the empty pixel were chosen for filling the gap.
6. 45 degree rotation also required creating a new pixel array with new dimensions.
7. Scaling the image twice required each pixel to be copied into 4 pixel positions for a perfectly filled image.
8. The pixel intensity values were required to be copied to the other two channels in case of RGB image due to the presence of 3 channels.