Image Processing Laboratory

Experiment 1

Report

Submitted by

Arnab Biswas (21EC65R01)

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, rotate the image by 45 degree and 90 degree and save it in the disk, scaling the image and set each channel of the 'corn.bmp' to zero at an instance. This experiment was done without the help of OpenCV function, 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:

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

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 Reserved; actual value depends on the application that creates the image
06	6	2	Reserved; actual value depends on the application that creates the image
08	8	2	The offset, i.e. starting address, of the byte where the bitmap image data (pixel array) can be found.
0A	10	4	

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

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.

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

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

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

1. Here, we have done the rotation using the matrix formula that

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

From the matrix formula, we can write that

$x = u\cos\theta - v\sin\theta$ and $y = u\sin\theta + v\cos\theta$

For getting the pixel array and set each channel of the 'corn.bmp' to zero at an instance, I have changed the pixel value of red, blue and green one by one and save the image into three files named CornNotBlue.bmp, CornNotRed.bmp, CornNotGreen.bmp.

Scaled image is generated using the formula:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Here, S_x and S_y are the scale factor of this image.

Algorithm

There are three main operations int the program:

- **openbmpfile:** Function to read the BMP file. 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: Read the data that may appear as unknown, so that during writing, we can use them again to avoid error in writing image.

```
void openbmpfile(){
                                                                    //function to
invoke all operations
        struct bmpheader header;
                                                                //declare a variable to
store the bitmap header
        struct infohead dibheader;
                                                                //deaclre a variable to
store the dib header
        //printf("%d\n",sizeof(struct bmpheader));

        FILE *fp=fopen("lena.bmp","rb");
                                                                //lena.bmp is opened
with a file pointer
        if(fp==NULL) return 1;


        fread(&header.name0,1,1,fp);
        fread(&header.name1,1,1,fp);
        fread(&header.size,3*sizeof(int),1,fp);

        //fread(&header,sizeof(struct bmpheader),1,fp);
        printf("\n\nFor The Invoked Image:");
        printf("\nFirst two characters:%c%c\n",header.name0,header.name1);
        if((header.name0!='B')&&(header.name1!='M'))
                                                        //check if it is a bitmap
        {
                printf("\nNot a Bitmap Image:");
                fclose(fp);
                return;
        }

        printf("Size:%d\n",header.size);
        printf("Offset:%d\n",header.image_offset);
        fread(&dibheader.header_size,sizeof(struct infohead),1,fp);
//store the dib header data
        printf("Header size:%d\nWidth:%d\nheight:%d\nColor planes:%d\nBits per
pixel:%d\nCompression:%d\nImage
Size:%d\n",dibheader.header_size,dibheader.width,dibheader.height,dibheader.colorplane
s,dibheader.bitsperpixel,dibheader.compression,dibheader.image_size);


        //displays the header data
        if((dibheader.header_size!=40)||(dibheader.compression!=0)||(dibheader.bitsperp
ixel!=24))          //checks if the image is 24bpp image with standard compression
```

```
	{
		fclose(fp);
		return;
	}

	fseek(fp,header.image_offset,SEEK_SET);
                                                        //fp is pointed to start
of image pixel array
	struct Image image=newimage(fp,dibheader.height,dibheader.width);
                                        //image pixel array is created and stored in
image
	createBWImage(header,dibheader,image);
                                                        //grayscale image is
created

	fseek(fp,header.image_offset,SEEK_SET);

	image=newimage(fp,dibheader.height,dibheader.width);
	flipped(header,dibheader,image);
                                                //flipped image is created

	fseek(fp,header.image_offset,SEEK_SET);
	image=newimage(fp,dibheader.height,dibheader.width);
	rotate90(header,dibheader,image);
                                                //image rotated by 90degree is created

	int scale=2;

	//upscaling image size scaling is declared
	struct infohead dibheader2=dibheader;

	struct bmpheader header2=header;
                                                        //height and width are
multiplied by scaling factor
	dibheader2.height=dibheader.height*scale;
	dibheader2.width=dibheader.width*scale;
	dibheader2.image_size=dibheader2.height*dibheader2.width*3;
                                                //now header and image size sizes are
declared
	header2.size=dibheader2.image_size+header2.image_offset;

	printf("\n\nFor The Image to be Upscaled:");
	printf("\nFirst two characters:%c%c\n",header2.name0,header2.name1);
	printf("Size:%d\n",header2.size);
	printf("Offset:%d\n",header2.image_offset);
	printf("Header size:%d\nWidth:%d\nheight:%d\nColor planes:%d\nBits per
pixel:%d\nCompression:%d\nImage
Size:%d\n",dibheader2.header_size,dibheader2.width,dibheader2.height,dibheader2.colorp
lanes,dibheader2.bitsperpixel,dibheader2.compression,dibheader2.image_size);

                                                        //all header parameters
are shown for the upscaled image

	fseek(fp,header.image_offset,SEEK_SET);
	image=newimage(fp,dibheader.height,dibheader.width);
	struct Image image2;

	image2=scaledimage(dibheader2.height,dibheader2.width,image,scale); //partially
upscaled image array is created
	scaled(header2,dibheader2,image2,scale);
//interpolated image is created and stored
```

```
	dibheader2.height=dibheader.height*2;
	dibheader2.width=dibheader.width*2;
	dibheader2.image_size=dibheader2.height*dibheader2.width*3;
                                                //now header and image size sizes are
declared
	header2.size=dibheader2.image_size+header2.image_offset;

	fseek(fp,header.image_offset,SEEK_SET);
	image=newimage(fp,dibheader.height,dibheader.width);

	image2=createimage(dibheader2.height,dibheader2.width); //partially upscaled
blank image array is created
	fseek(fp,header.image_offset,SEEK_SET);
	image=newimage(fp,dibheader.height,dibheader.width);
	rotate45(header2,dibheader2,image2,image);
	fclose(fp);
	freeImage(image);
	freeImage(image2);
                                                //file pointer is closed
	return;
}
```

rotate90():

This function is used to rotate the given image by 90 degree and get the output. Here, we take the input image by a file pointer and put a check if it is NULL or not if this is not NULL, then pass the image through `rgbrotate90`. This function is used to rotate the image by 90 degree and it is done by flipping the image. The resultant image array is flipped by the middle column of the image pixel array.

```
int rotate90( struct bmpheader header, struct infohead dibheader,struct Image pic){ //
function to rotate an image by 90degree and store it in a file

	int i;
	FILE *fpw=fopen("Rot90.bmp","wb");
	if (fpw==NULL) return 1;

	rgbrotate90(pic);
                                                                        //function to
rotate an image by 90degree is invoked

	fwrite(&header.name0,1,1,fpw);
	fwrite(&header.name1,1,1,fpw);
	fwrite(&header.size,3*sizeof(int),1,fpw);

	fwrite(&dibheader,sizeof(struct infohead),1,fpw);
	fseek(fpw,header.image_offset,SEEK_SET);
	for(i=pic.height-1;i>=0;i--){
		fwrite(pic.rgb[i],pic.width,sizeof(struct RGB),fpw);
	}


	//file writing operation is complete
	fclose(fpw);
	return 0;

}
```

```
void rgbrotate90(struct Image pic){                    //function to rotate an image
by 90degree
       int i,j;
       unsigned char T;
       struct RGB PT;
       RGBFlipImage(pic);                                          //function to
flip an image along its principle diagonal is invoked
       for(i=0;i<pic.height;i++)
              for(j=0;j<pic.width/2;j++)                    //the resultant image
pixel array is flipped by its middle column
              {

                     PT=pic.rgb[i][j];
                     pic.rgb[i][j]=pic.rgb[i][pic.width-j];
                     pic.rgb[i][pic.width-j]=PT;
              }
}
```

Flipped:

This function is used to flipped the given image and it is done by the help of another function named `RGBFlipImage`, which is used nested for loope to convert the given image into its traverse by interchanging the rows and columns of the image pixel array. In the `flipped` function, first I checked if the contain of the given image is empty or not. If the contain is not empty, then have passed the image through the function `RGBFlipImage`.

```
void flipped( struct bmpheader header, struct infohead dibheader,struct Image pic){
              //function to create and store the flipped version of an image fliped
along its principle  diagonal

       int i;
       FILE *fpw=fopen("Flipped.bmp","wb");                          //Opens a
file Flipped.bmp for writing
       if (fpw==NULL) return 1;

       RGBFlipImage(pic);
       //function to flip the image is invoked

       fwrite(&header.name0,1,1,fpw);
       fwrite(&header.name1,1,1,fpw);
       fwrite(&header.size,3*sizeof(int),1,fpw);

       fwrite(&dibheader,sizeof(struct infohead),1,fpw);
       fseek(fpw,header.image_offset,SEEK_SET);
       for(i=pic.height-1;i>=0;i--){
              fwrite(pic.rgb[i],pic.width,sizeof(struct RGB),fpw);
       }
                     //storing operation is completed
       fclose(fpw);

}

void RGBFlipImage(struct Image pic){            //function to flip image along
principle diagonal
       int i,j;
       unsigned char T;
       struct RGB PT;
```

```
	for(i=0;i<pic.height;i++)
		for(j=i+1;j<pic.width;j++)		//the image pixel array is
converted into its transverse
		{

			PT=pic.rgb[i][j];
			pic.rgb[i][j]=pic.rgb[j][i];
			pic.rgb[j][i]=PT;

		}


}
```

createBWImage:

This function is used to create a grayscale image from a colour image. This can be done the formula $Grayscale = R \times 0.30 + G \times 0.59 + B \times 0.11$, which is used in the function `grayscale`. This function is called from the function, `RGBImageToGrayscale`, which is used to change the value of all the pixels of red, blue and green colour to the same value which can be found using the above formula.

```
int createBWImage(   struct bmpheader header, struct infohead dibheader,struct Image
pic){					//function to create and store the grayscaled
image

	int i;
	FILE *fpw=fopen("BMgray.bmp","wb");

		//opens a file BMgray to stored the result
	if (fpw==NULL) return 1;

	RGBImageToGrayscale(pic);

		//function to convert pic into grayscale

	fwrite(&header.name0,1,1,fpw);
	fwrite(&header.name1,1,1,fpw);
	fwrite(&header.size,3*sizeof(int),1,fpw);

	fwrite(&dibheader,sizeof(struct infohead),1,fpw);
	fseek(fpw,header.image_offset,SEEK_SET);

	for(i=pic.height-1;i>=0;i--){
		fwrite(pic.rgb[i],pic.width,sizeof(struct RGB),fpw);
	}

	fclose(fpw);
	return 0;

}

unsigned char grayscale(struct RGB rgb){

	//returns the grayscaled RGB value for a pixel
	return ((0.3*rgb.red)+(0.6*rgb.green)+(0.1*rgb.blue));
}
```

```
void RGBImageToGrayscale(struct Image pic){

	//function to create a grayscaled image sent as argument
	int i,j;


	for(i=0;i<pic.height;i++)
		for(j=0;j<pic.width;j++)

	pic.rgb[i][j].red=pic.rgb[i][j].green=pic.rgb[i][j].blue=grayscale(pic.rgb[i][j
]);					//grayscaled contribution is made of 30% red
60%blue 10%green
}
```

rotate45:

This is used to rotate the given image by 45 degree angle. Here, at first we have create a larger image than the given image to get the output image correctly without cropping. So, we have to translate using the axis formula on rotation of an image.

```
void rotation45(struct Image pic,struct Image image){
					//function to rotate image and put it in
corresponding value of pic
	int x,y,x1,y1;

	unsigned char T;
	struct RGB PT;
	for(y=(-pic.height/4-1)+1;y<(pic.height/4)-1;y++)
		for(x=(-pic.width/4)+1;x<pic.width/4-1-1;x++)
		{
			y1=pic.height/2-1+ (-(x+pic.width/4-pic.height/4+1+y)/sqrt(2.0));
			x1=pic.width/2+ ((pic.width/4+x-y-pic.height/4+1)/sqrt(2.0));


			pic.rgb[y1][x1]=image.rgb[pic.height/4-1-y][pic.width/4+x];

		}
}

int rotate45(struct bmpheader header2, struct infohead dibheader2, struct Image
pic,struct Image image){	//function to create rotated and store partially upscaled
image

	int i;
	FILE *fpw=fopen("Rotated.bmp","wb");

	//file pointer is used to point to file invoked in writing mode in binary
format
	if (fpw==NULL) return 1;

	rotation45(pic,image);

		//function to rotate the image

	fwrite(&header2.name0,1,1,fpw);
	fwrite(&header2.name1,1,1,fpw);
	fwrite(&header2.size,3*sizeof(int),1,fpw);

	//header and dib header data of bitmap is stored in the file sequentially
```

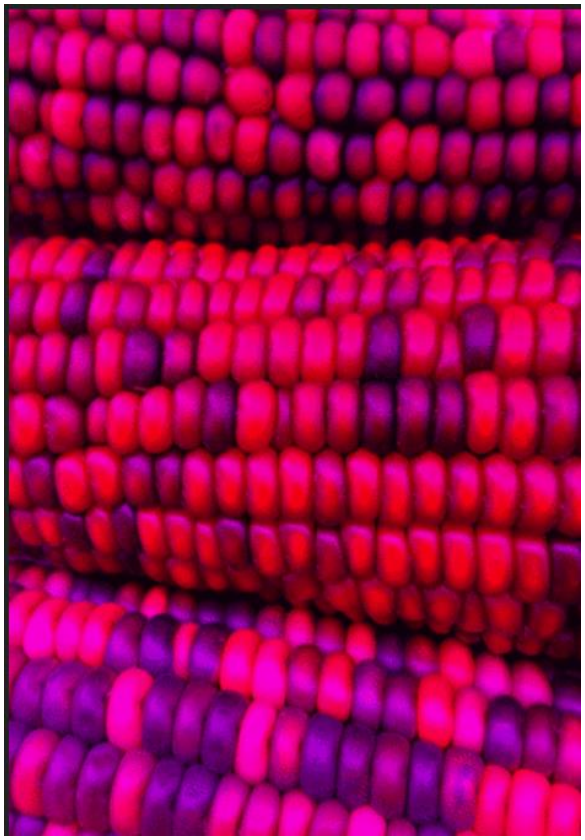
```
	fwrite(&dibheader2,sizeof(struct infohead),1,fpw);
	fseek(fpw,header2.image_offset,SEEK_SET);
	for(i=pic.height-1;i>=0;i--){

		//the image pixel array is stored in linear format in row major wise in
bottom up format
		fwrite(pic.rgb[i],pic.width,sizeof(struct RGB),fpw);
	}

	fclose(fpw);

			//close the file pointer
	return 0;

}
```

I have mentioned the datatype `cv::MatIterator_<cv::Vec3b>` to iterate through the image and change the pixel value of red, blue and green colour.

I have used the image name cornmain.bmp as my input and save the output image in the file named CornNotBlue.bmp, CornNotGreen.bmp, CornNotRed.bmp.

```
Mat img1 = imread("C:/cornmain.bmp", -1);
	for (cv::MatIterator_<cv::Vec3b> it = img1.begin<cv::Vec3b>(); it !=
img1.end<cv::Vec3b>(); ++it)
	{
		(*it)[0] = 0;
	}
	imwrite("C:/opencv/CornNotBlue.bmp", img1);
	Mat img1 = imread("C:/cornmain.bmp", -1);
	for (cv::MatIterator_<cv::Vec3b> it = img1.begin<cv::Vec3b>(); it !=
img1.end<cv::Vec3b>(); ++it)
	{
		(*it)[1] = 0;
	}
	imwrite("C:/opencv/CornNotGreen.bmp", img1);
	Mat img1 = imread("C:/cornmain.bmp", -1);
	for (cv::MatIterator_<cv::Vec3b> it = img1.begin<cv::Vec3b>(); it !=
img1.end<cv::Vec3b>(); ++it)
	{
		(*it)[2] = 0;
	}
	imwrite("C:/opencv/CornNotRed.bmp", img1);
```

Output:



CornNotGreen.bmp

CornNotBlue.bmp

CornNotRed.bmp

Flipped Output:

Flipped1.bmp

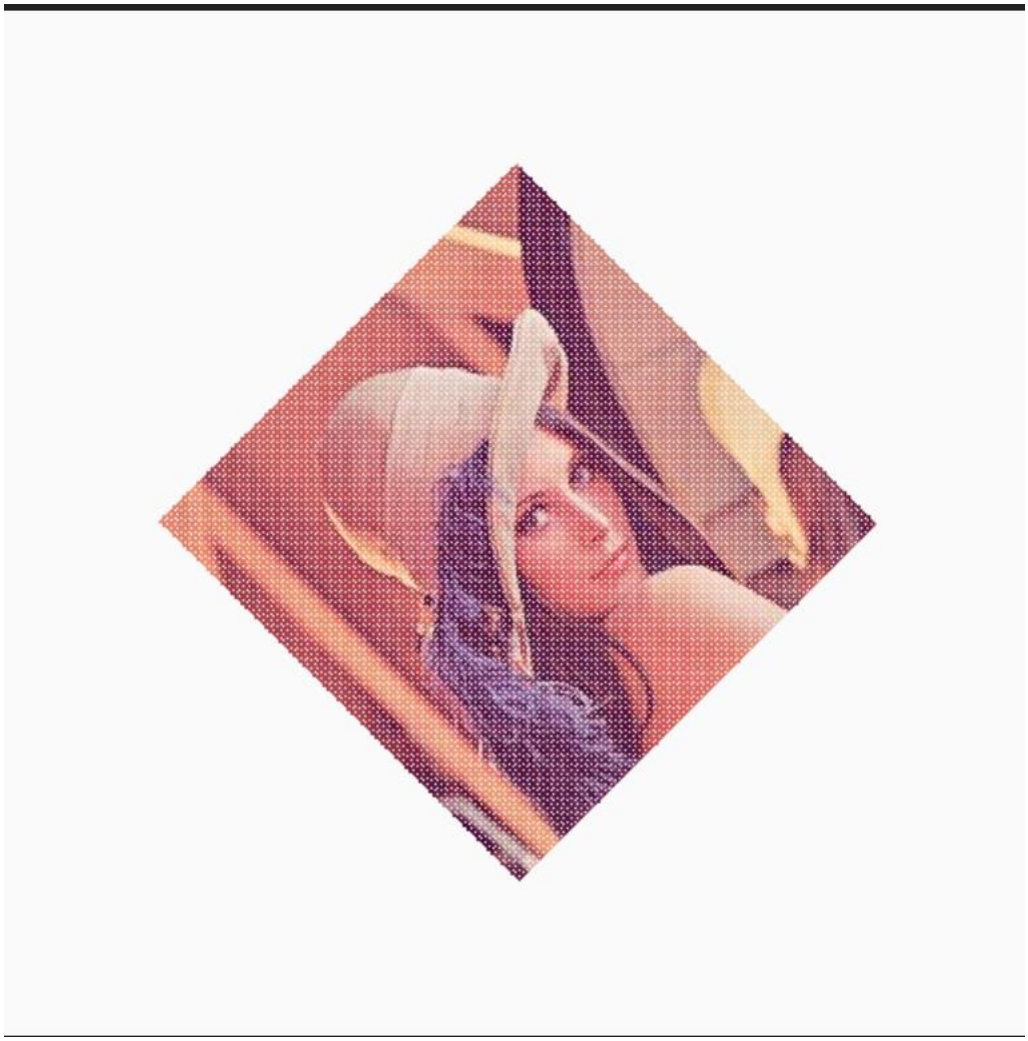
Flipped2.bmp

Output with 90-degree rotation:

Rotate901.bmp

Rotate902.bmp

Output with 45-degree rotation:



Rotated1.bmp

Rotated2.bmp

Scaled Output:

Upscaled1.bmp

Upscaled2.bmp