INDIAN INSTITUTE OF TECHNOLOGY

KHARAGPUR

DIGITAL IMAGE PROCESSING LABORATORY

A REPORT ON

EXPERIMENT 01

# Geometric Transformations

25.01.2021

Group No. 16


| | | |
|---|---|---|
| Name: | Sirish Gambhira | Shubham Sahoo |
| Roll No: | 17EC35009 | 17EC35023 |

**DEPT OF ELECTRONICS AND ELECTRICAL COMMUNICATION ENGINEERING**

**VISUAL INFORMATION AND EMBEDDED SYSTEMS**

# Table of Contents

# Introduction

The BMP file format also known as Bitmap image is used to store bitmap digital images. It is capable of storing two-dimensional images in both monochrome and color, in various color depths, with data compression, alpha channels and color profiles.

**File Structure:**

The bitmap image file consists of a fixed-size header structure followed by a color table and pixel array. The file format is as follows:

1. Header: contains information about the type, size, and layout of a device-independent bitmap file
2. Info Header: specifies the dimensions, compression type, and color format for the bitmap
3. Color Table: contains as many elements as there are colors in the bitmap, but is not present for bitmaps with 24 color bits because each pixel is represented by 24-bit red-green-blue (RGB) values in the actual bitmap data area
4. Pixel Data: an array of bytes that defines the bitmap bits. These are the actual image data, represented by consecutive rows, or scan lines, of the bitmap. Each scan line consists of consecutive bytes representing the pixels in the scan line, in left-to-right order. The system maps pixels beginning with the bottom scan line of the rectangular region and ending with the top scan line.

## Read BMP

The following is the procedure to read a BMP image (8-bit and 24-bit). The function takes the filename as a parameter.

The header of BMP image is of fixed size - 14 bytes, followed by a information header of fixed size - 40 bytes. We read the file using fread command and stored the header information in a 1D array of size 54 bytes.

Image attributes like Signature, file size, reserved, data offset, header size, width, height, planes, bits per pixel, compression, image size, XpixelsPerM, YpixelsPerM, Colors Used, Important colors are extracted from the array at appropriate locations.

The following are important attributes: height - represents the height of BMP image, width - represents the width of BMP image, bit-width - represents the number of bits required to represent the image (8-bit implies grey scale and 24 bit implies color image), data offset - implies the location from where pixel array is stored in the BMP file.

Color table of size [data offset - header] is read from the subsequent position.

BMP image expects its width to be multiple of 4 bytes. This is always true for grey-scale (8-bit) images, whereas for 24 bit images, if width is not a multiple of 4, then it is padded accordingly.

Hence, we took care of padding by calculating the padded_row_size and unpadded_row_size.

Algorithm:

```
int unpadded_row_size = width * colors;
int padded_row_size = (int)(4 * ceil((float)(A.width*colors/4.0f)));
int totalsize = unpadded_row_size * A.height;
imarray = new uint8_t[totalsize];
int i = 0;
uint8_t* currentpointer = imarray +  (A.height - 1)*unpadded_row_size;
for(int i=0;i<A.height;i++)
{
        fseek(ifile, A.offset + (i * padded_row_size), SEEK_SET);
        fread(currentpointer, sizeof(unsigned char), unpadded_row_size, ifile);
```

```
        currentpointer -= unpadded_row_size;
}
```

Since, the image is stored in bottom-up fashion, while reading the file, we filled the pixel array in bottom-up fashion.

`fseek(ifile, A.offset + (i * padded_row_size), SEEK_SET)`, here we multiply it by padded_row_size as the BMP image may contain padding. SEEK_SET is used to start from the top of the file.

`fread(currentpointer, sizeof(unsigned char), unpadded_row_size, ifile)`, here only unpadded_row_size amount of data is read since it is of our interest and thus, the corresponding array pointer is updated accordingly `currentpointer -= unpadded_row_size`

## Write BMP

Similar procedure is followed inorder to write a BMP image. The function takes BITMAP header structure, pixel array and filename as parameters.

The bitmap header attributes like signature, filesize, color table, etc.., are read from the BITMAP header structure.

Algorithm:

```
for(int i=0;i<height;i++)
{
        int pixeloffset = ((height - i - 1) * unpadded_row_size);
        fwrite(&pixels[pixeloffset], 1, padded_row_size, outfile);
}
```

Since the BITMAP images are stored in bottom-up fashion, the last row of the pixel array is selected first using the pixeloffset variable. During writing, padded_row_size amount of data is written from the position. If the available data size is less than padded_row_size, then automatically zeros are written onto the file.

## Geometric Transformations

Geometric transformations are defined as a set of transforms where the geometry of the image is changed without altering the pixel values. It involves two basic steps of transformations[1]:

1. Spatial transformation
2. Interpolation

### Spatial Transformation

Pixel coordinates (x, y) undergo geometric distortion to produce an image with coordinates (x', y')

$$x' = r(x, y)$$
$$y' = s(x, y),$$

There are different types of transformations available:

1. Translation
2. Scaling
3. Shear
4. Rotation

### Interpolation

It works on the principle of using known data to estimate the values at unknown points. There are many interpolation methods available:

1. Nearest Neighbour
2. Bilinear interpolation
3. Bicubic Interpolation

[1] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT5/node5.html

**Grey-scale conversion**

The 24-bit color images are converted into grey-scale images using the following formula:

intensity[2] = 0.3 * red + 0.59 * green + 0.11 * blue.

We used a 1D pixel array inorder to read BMP images. Hence the red, green, blue intensities of (i, j) pixel can be obtained by the data [i * width * colors + j * colors + 2], data[i *width*colors + j*colors + 1], data[i * width * colors + j * colors + 0]. Here width represents width of image, colors = 3 (since RGB).

Correspondingly, the image offset and bits per pixel are modified to write the resulting pixel array as the grey-scale image.

**Diagonally Flipped pixel array**

In order to diagonally flip a pixel array, a pixel at (i, j) location must be shifted to (j, i) in the resulting image.

Let height, width represent height and width of the image. Since a 1D array is used, (i, j) in original image i.e original[i * width + j] must be moved to transpose[j*height + i]. This can be extended to color images by copying the color values similarly.

**Rotating pixel array**

We wrote a custom function inorder to rotate the image by any angle. The following are the steps involved:

1. Convert the 1D array into 3D array (for color images)
2. Create new array target where the rotated image is to be written
3. We know that the centre of image will not undergo any rotation, hence all the while iterating through pixels, the distance from centre should be considered, instead of pixels directly.

[2] https://en.wikipedia.org/wiki/Grayscale#Converting_color_to_grayscale

4. For every pixel in the target image, we find the corresponding pixel in original image by following formula:

   $$[\text{old coordinates}] = [R]^T[\text{new coordinates}]$$

5. Here R represents the rotation matrix i.e

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

6. Since, we rotated with respect to centre, the centre coordinates are added to get back the original coordinates.
7. Since the coordinates can be floating point, we used interpolation methods (discussed below) to get the required pixel.

**Scaling**

We wrote a custom function to scale the pixel array by any factor. Currently, only till pixel array size of 512 is supported.

The procedure is as follows:

1. Inorder to scale the pixel array by n times, the height and width are to be scaled by sqrt(n) times.
2. For easy implementation, we converted the 1D arrays to 3D.
3. fac_x, fac_y represent the amount of times, x coordinates and y coordinates to be scaled.
4. for each pixel position (i, j) in the original image, the scaled position (x, y) is calculated accordingly and filled.
5. Inorder to interpolate the missing pixels, we implemented two algorithms.

**Interpolation**

The coordinates obtained in the original image are converted from float to integer by taking floor operator.

1. **Nearest Neighbor interpolation**
    a. Inorder to interpolate a pixel position (x, y), we considered a square grid centred around the pixel of size = 1 (parameter)
    b. For each color in the image, the intensity values in this grid are averaged and assigned to the required pixel.

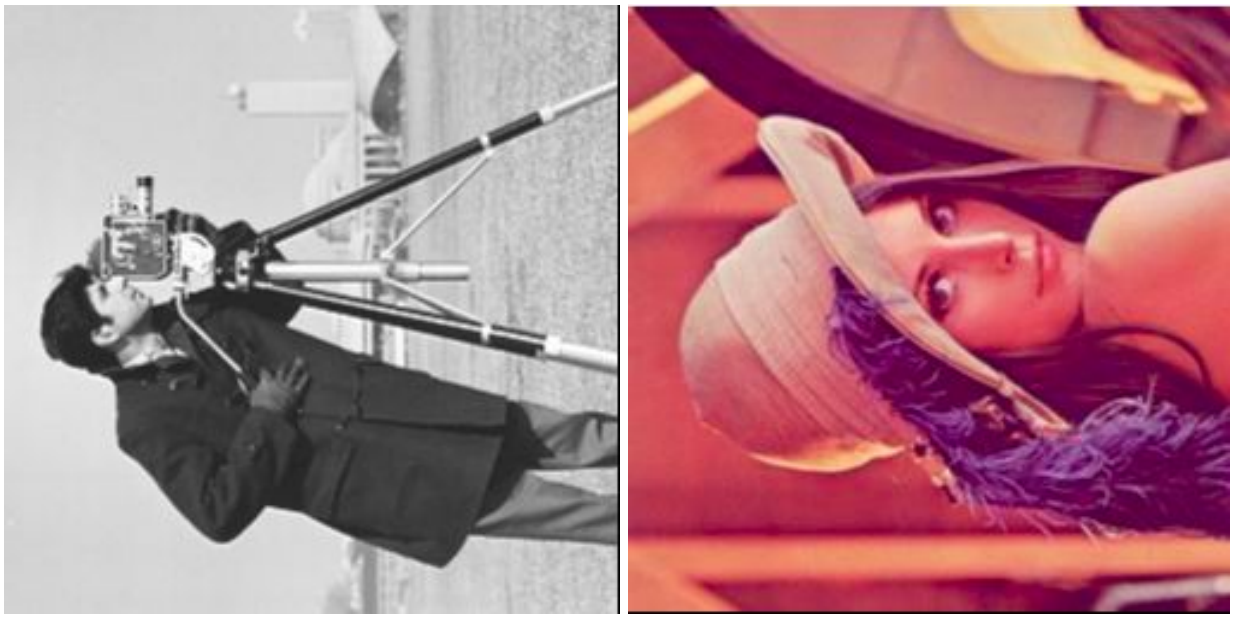
2. **Bilinear Interpolation**
    a. Inorder to interpolate a pixel position (x, y), we considered four neighbors i.e A(x - 1, y - 1), B(x - 1, y + 1), C(x + 1, y + 1), D(x + 1, y - 1).
    b. If x,y is equal to any of these neighbors, we returned the intensity value directly.
    c. Let q1, q2, q3, q4 represent the intensity values at these locations.
    d. The bilinear interpolation is implemented in two stages, i.e first we perform linear interpolation using A and D, to obtain a point E.
    e. Similarly, we interpolate linearly B and C, to obtain a point F.
    f. The required value is obtained by interpolating using E and F as boundaries.
    g. This can be extended to color images by iterating over channels.

# Results

## Input Images :

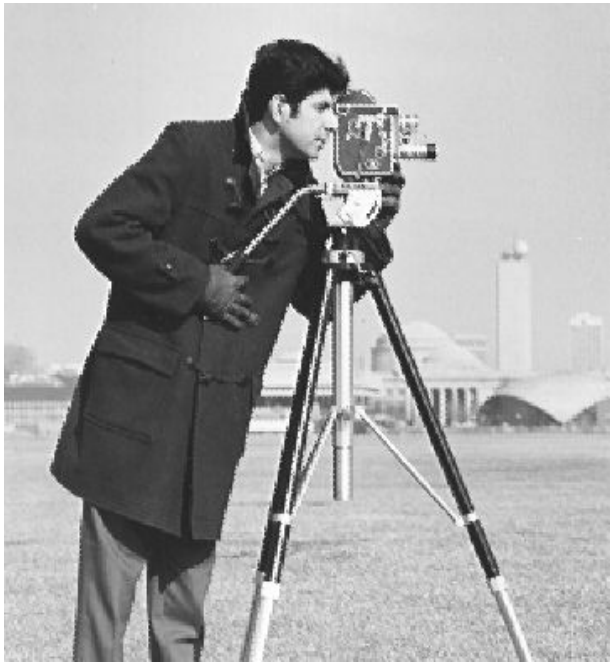

## Gray scale Images:


**$90^0$ degree rotated :**



**$45^0$ degree rotated :**

**Scaled :**



**Transposed :**

## Analysis

- Conversion of BMP formatted image data to multidimensional arrays helps in easier operations than the original format.
- Nearest neighbor interpolation gave better results than bilinear interpolation method for rotation and scaling.
- We can use different color tables in grayscale images also known as color palettes.
- The size parameter in the nearest neighbor interpolation can be changed according to the scaling factor(which is a fixed no. 2 for this experiment).
- Rotation and Scaling are written in generalised form which is applicable for any parameters.

### References

1. www.ece.ualberta.ca/~elliott/ee552/studentAppNotes/2003_w/misc/bmp_file_format/bmp_file_format.htm
2. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT5/node5.html
3. https://en.wikipedia.org/wiki/Grayscale#Converting_color_to_grayscale