INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Department of Electronics & Eletrical Communication Engineering

Vision and Intelligent Systems

EC69211 – Image and Video Processing Laboratory

Experiment – 2

Scaling and rotating BMP image

Submitted by:

Bbiswabasu Roy (19EC39007)

Jothi Prakash (19EC39023)

CONTENTS

Objective:

1. Scaling a BMP image with different scaling factors along x and y directions where scaling factors can be <1 or >1
2. Rotating a BMP image by 45^o and 90^o using nearest neighbour and bilinear interpolation

Theory:

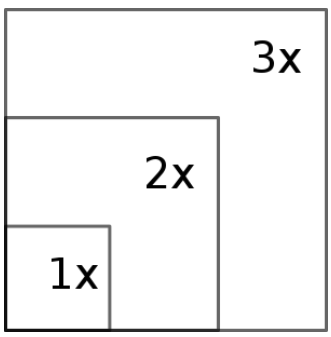
- Image Scaling

  We will assume we have an image with a resolution of width × height that we want to resize to new_width × new_height. First, we will introduce the scaling factors scale_x and scale_y defined as follows:

  scale_x = new_width / width

  scale_y = new_height / height

  A scale factor <1 indicates shrinking while a scale factor >1 indicates stretching.

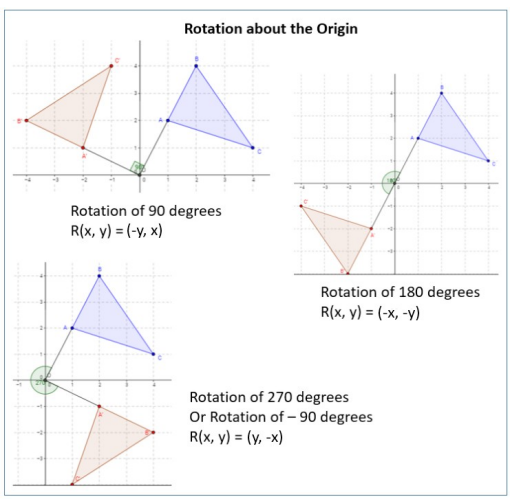


- Image Rotation

  We assume the x and y coordinates of the image and then perform the rotation transformation as given below.

$$x' \rightarrow x\cos(\theta) - y\sin(\theta)$$
$$y' \rightarrow y\cos(\theta) + x\sin(\theta)$$

  This transformation gives us the final coordinates of each pixel in the new rotated matrix of the image.

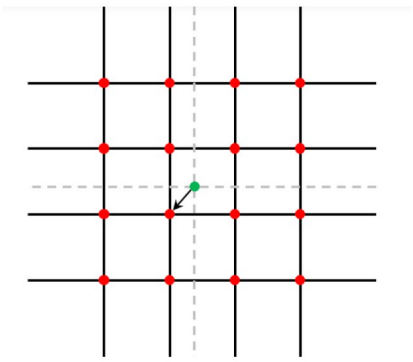


- Nearest Neighbour Interpolation

  This is probably the simplest image scaling method. Every output pixel is replaced by its nearest pixel in the input.

  In 1 dimension, the value at any points x is the value of its nearest point to immediate left or right.

  In 2D, a pixel with coordinates (x, y) in the output image has coordinate (x / scale_x, y / scale_y) in the input image. Since these coordinates don't always exist (if they have a decimal part), we will round the coordinates to the nearest integer, thus rounding to the nearest neighbour.

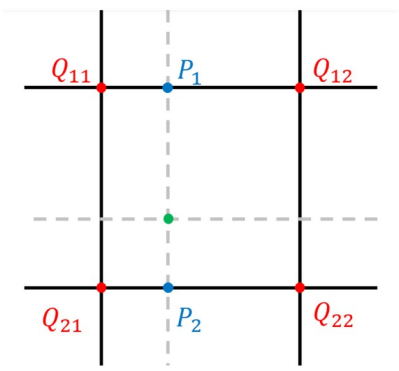


Nearest neighbour in 2-D

- Bilinear Interpolation

  Linear interpolation is equivalent to drawing a line between every two consecutive points. This can also be thought of as taking the average of the neighbouring points weighted by their distance.

  In 2 dimensions, this interpolation involves taking wighted sum from the 4 neighbouring points. The sum can be computed by first interpolating along rows and then interpolating along columns or the other way. This results in smoother transition of intensity values at different points as compared to nearest neighbour interpolation.



Bilinear interpolation in 2-D

Algorithms:

Following algorithm was used to read the BMP file header and pixel array:

1. Open the required file in read binary format
2. Keep reading appropriate number of bytes from the file as per the header format, store it in memory as *unsigned char[]* and then convert the stored bytes to *unsigned int*
3. If number of colors mentioned in header is non-zero, then read and store the color table data
4. Declare a pixel_array of dimension height x width x 3
5. For each pixel (i, j) in the image, do
6. Read bits_per_pixel/8 number of bytes from current file pointer.
7. Store the intensity levels in pixel_array at index (height – i, j) as RGB (though they are read as BGR from file). Since we only handle 8-bit grayscale and 24-bit colored images, we don't need to shift file pointer to color table

Following algorithm was used to write the BMP file header and pixel array:

1. Open the required file in write binary format
2. Convert header fields form *unsigned int* to *unsigned char[]* and keep writing appropriate number of bytes in the output file
3. If the image has color table, write the colors with same indexing as stored in the bmp_header
4. For each pixel (height-i, j) in pixel_array, do
5. Get RGB components of the image from the pixel_array[height-i][j] and convert it to *unsigned char[]* of size bits_per_pixel/8 storing data in BGR order
6. Write the obtained bytes in the file

Following algorithm was used to scale the image:

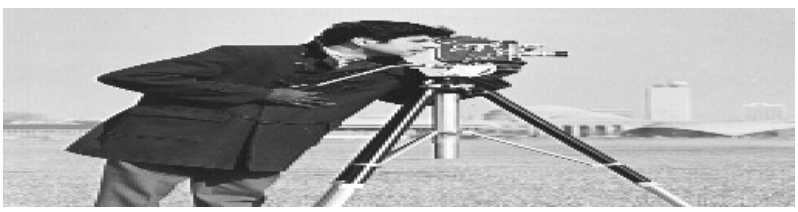
1. Take the scaling factor as input for the x and y direction
2. Read the Image data in bmp_data and pass it to the scaleImage() function to scale the image
3. Create a new_pixel_array with new scaled dimensions to accommodate the resized image
4. For each pixel in new_pixel_array, find its position in original pixel_array and perform the Nearest Neighbour Interpolation to fill up the matrix values of the scaled image from the initial image
5. Assign original pixel_array = new_pixel_array and update header fields in bmp_data

Following algorithm was used to rotate the image:

1. Take the rotation angle and the type of interpolation as input.
2. Compute new_height and new_width of image after rotation and create a new pixel_array of dimensions new_height x new_width x 3
3. Convert row and column indices to cartesian coordinates with +ve x-axis along column and -ve y-axis along rows
4. Perform inverse rotation of the final image points with respect to final image center and find its corresponding points in original image
5. From the points obtained on original image, perform Nearest Neighbour or Bilinear Interpolation to get interpolated intensity values at those points
6. Assign original pixel_array = new_pixel_array and update header fields in bmp_data

Results:

The following output images were generated when height was scaled by f_x and width was scaled by f_y using nearest neighbour interpolation:



cameraman.bmp with $f_x = 0.5$ and $f_y = 2$

lena_colored_256.bmp with $f_x = 0.75$ and $f_y = 3$

cameraman.bmp with $f_x = 2$ and $f_y = 0.5$

lena_colored_256.bmp with $f_x = 2$ and $f_y = 1$

Rotation was applied to the given images with nearest neighbour and bilinear interpolations. It was found that when nearest neighbour interpolation was applied, the image had some sharp transitions or edges while the image was much smoother when bilinear interpolation was applied to it. However, no appreciable difference was found when the angle of rotation was multiple of 90^o because in those cases, there were no extra pixels and hence the effect of interpolation was prominent. The following output images were generated on applying rotation:

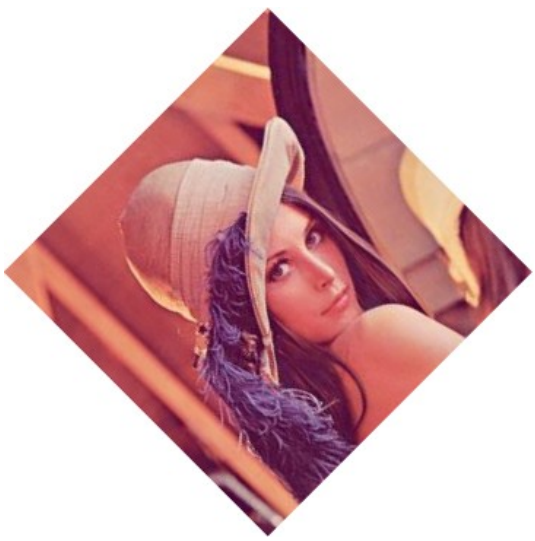
cameraman.bmp rotated 45^o with nearest neighbour interpolation

cameraman.bmp rotated 45^o with bilinear interpolation

cameraman.bmp rotated 90º with nearest neighbour interpolation

cameraman.bmp rotated 90º with bilinear interpolation

lena_colored_256.bmp rotated 45º with nearest neighbour interpolation



lena_colored_256.bmp rotated 45º with bilinear interpolation

lena_colored_256.bmp rotated 90º with nearest neighbour interpolation

lena_colored_256.bmp rotated 90º with bilinear interpolation

Discussion:

- After scaling the image, the size of data in image file will change but the values stored in header fields like “File size”, “Image height”, “Image width” remains same as before. Hence, those fields must be explictly modified to make the data consistent. It was also observed that even if those fields were not modified, the images still opened perfectly without any error.
- It was important to note that each row in the pixel data of BMP file is padded so as to make it multiple of 4 bytes. While scaling the image, if the scaling factors were chosen such that width of row does not become multiple of 4 bytes, then we must pad that row to nearest higher mulitple of 4 bytes. When this padding was not done, it was found that the displayed image was tilted and many pixels got shifted from their expected location.
- While rotating the image, we perform backward transformation, i.e, start with final image and try to interpolate points from initial image. Due to this, if we want to rotate original image by θ, then for each pixel in final image we must find its new coordinate after rotating it by $-\theta$ about the image center.
- While applying rotation to the image, if we do not change the image (background) height and width then some part of the image might get cropped out. Hence, we must compute the new height and width of the image (which in general will be greater than original values) and then apply the rotation operation.
- If we use bilinear (or higher order) interpolation in an 8-bit color indexed image, then we may need to insert additional colors in the color table since the intensity levels obtained after interpolation might not be present in the original image. This might also lead to increase of the size of color table. These problems will not occur for nearest neighbour interpolation as for each pixel in final image, the intensity levels will be some color from the original image.