Multi-Threaded Image Rotation

Rohith Jayarajan
rohith23@terpmail.umd.edu

***Abstract*—The work involves implementing a library for effectively rotating two dimensional image while preserving the image quality. To speed up the image rotation process, the multi-threading paradigm is used. The library currently support Nearest Neighbor interpolation technique for matching pixels from input image to output image.**

I. Introduction

The problem at hand was the rotation of 2-Dimensional images in the most effective manner that would use the full power of the machine without compromising the image quality. So the best way to do this was using the multi-threading paradigm which allocates each thread a part of the task to effectively speed up the computation.

Currently the library supports the Nearest Neighbor interpolation technique for matching pixels from input image to output image. But the library was developed keeping in mind that it is extendable and should be generic to other higher quality interpolation techniques like Linear Interpolation and Cubic Interpolation. Hence, the requirement to the library to be generic is also kept in mind designing this project.

II. Implementation

The project was implemented in C++ following C++11 and above standards using its STL library and CMake. The program accepts the following parameters:

1) Path to input image.
2) Path to output image.
3) Angle of rotation in radians (where positive value denotes counter clock wise rotation and negative value denotes clock wise rotation).

The center of rotation x_c, y_c is $int(img_{height}/2 + 1), int(img_{width}/2 + 1)$.

If x, y denotes the current value of pixel coordinate and x', y' denotes the new pixel coordinate after rotation about the center of rotation, the relation between the two is given by equation 1.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{pmatrix} * \begin{pmatrix} x - x_c \\ y - y_c \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \end{pmatrix} \tag{1}$$

The output image will have the same size and file type of that of the input image. The presented solution works not only with JPG files but also PNG files.

To ensure the entire power of the machine is used, the total number of threads in the machine are calculated and the work of image rotation is distributed between these multiple threads.

Unit tests were written to ensure the correctness of the code.

III. Results

Image rotation using Nearest Neighbor interpolation technique was successfully implemented on both RGB and monochrome images. The results can be seen by rotation performed on the image in figure 1.

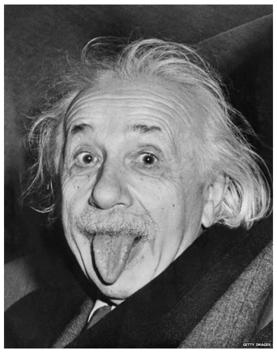
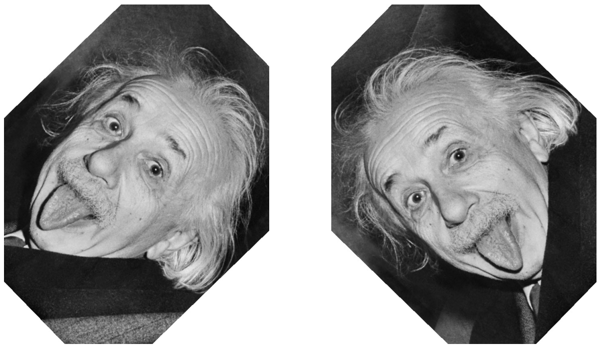


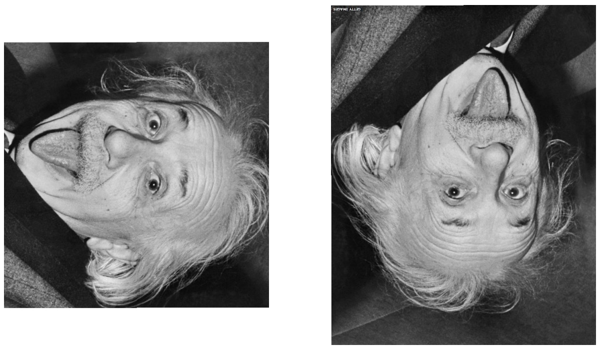
Fig. 1. Original Image

The rotated versions of image in 1 by -0.8, 0.8, -1.57, and -3.14 radians are shown in images 2 and 3.



(a) a (b) b

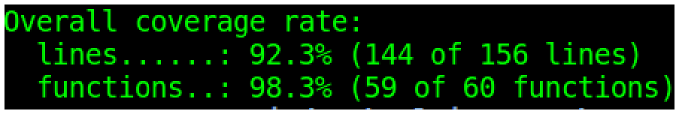
Fig. 2. (a) Image rotated by -0.8 radians. (b) Image rotated by 0.8 radians.



(a) a (b) b

Fig. 3. (a) Image rotated by -1.57 radians. (b) Image rotated by -3.14 radians.

Code coverage information is shown in figure 4



```
Overall coverage rate:
  lines......: 92.3% (144 of 156 lines)
  functions..: 98.3% (59 of 60 functions)
```

Fig. 4. Code coverage

IV. Conclusion

Hence, 2-D image rotation by Nearest Neighbor interpolation technique was successfully implemented and tested with a 98% function code coverage rate.