**TANG’s REPORT**

**Overview:**

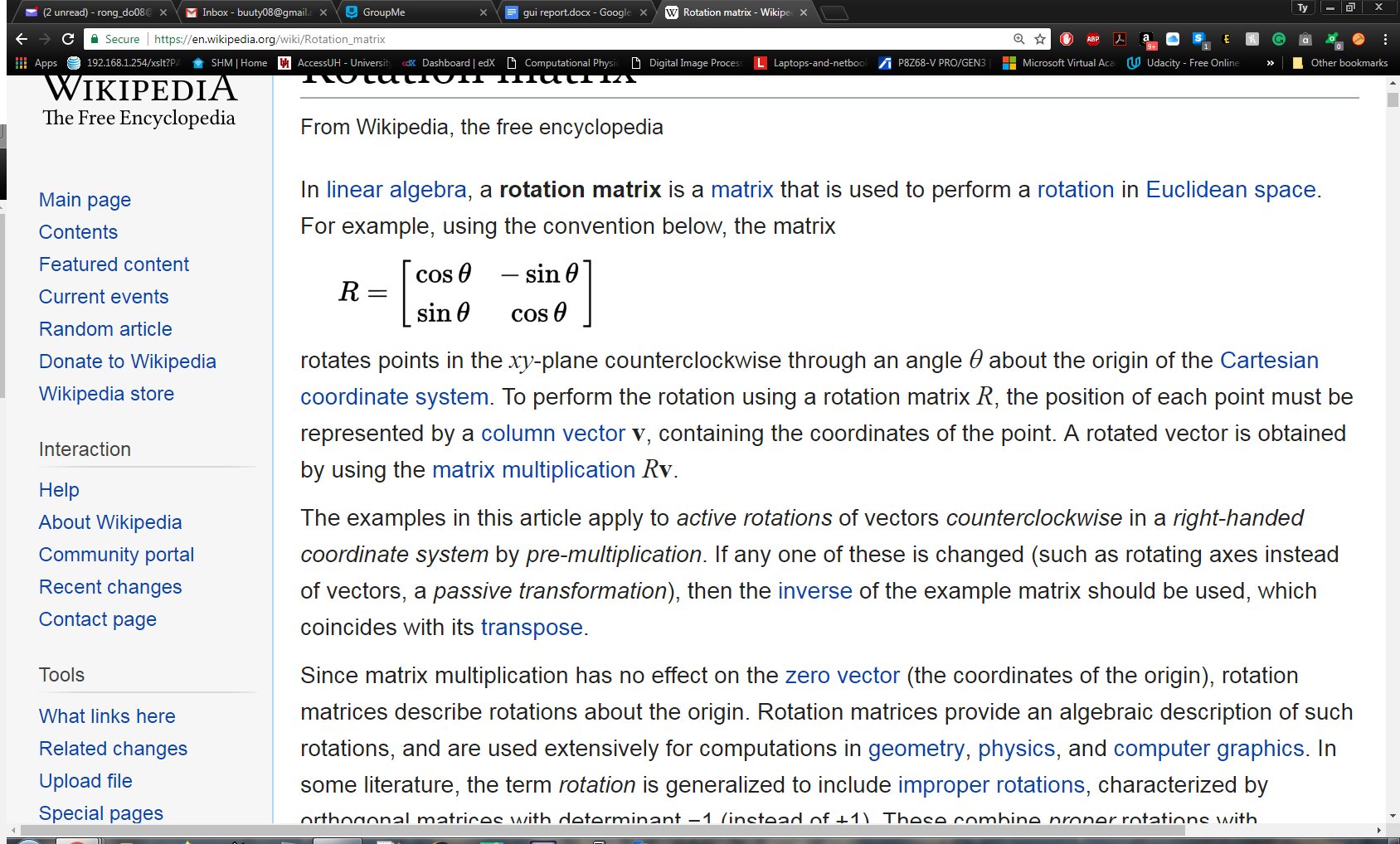
Geometric operations change an image’s geometry via translation, rotation, scaling, etc. Our objective for this project was to create a GUI that would perform specific global transformations of an image efficiently through rotation, translation, shearing, and scaling. In our project, **we were able to implement the following functions:** Rotation, Translation, Shearing(Vertical shearing, Horizontal Shearing), Scaling(Nearest Neighbor,Bilinear,cubic,Lanczos4 ).

**Images we used:**

**Rotation:**

Image rotation is performed by computing the inverse transformation for every destination pixel. Output pixels are computed using bilinear interpolation. The locations of pixels in an image are rotated around a certain origin by a determined rotation angle. Usually the center of the image is selected as origin, and the image is rotated respectively. When performing rotation, the mage maps the center point of the source image to the center point of the destination image. No scaling is done. Depending on the relative sizes of the source image and destination image, parts of the source image may be clipped, and areas outside the source image may appear in the destination image.

The implementation we used for this rotating function didn’t work as we expected. We used the rotation matrix: 

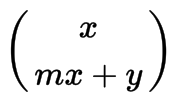
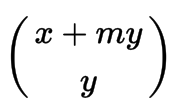
We applied this matrix to every pixels of the input image to calculate the new destination for each pixel to get the rotated image. However, the result wasn’t like what we expected. There were speckled black pixels dotted all over the rotated image. This is because some of the destination pixels were unassigned. Also, the specks formed patterns due to the sine and cosine functions and the regularity of pixel width and height.



**Shearing:**

In a simple vertical shearing operation (no scaling or translation), a rectangular region of interest is mapped into a parallelogram, whose left and right edges are parallel to the original rectangle, and whose top and bottom edges are slanted at a particular slope. The width of the parallelogram is the same as the width of the original rectangle, and the height of the parallelogram (the height of any column) is the same as the height of the original rectangle.

There is two functions for shearing. One for vertical and the other for horizontal shearing. I use the following simple formulas to shear the images.

They are for vertical and horizontal shearing respectively. However, because the images in the program are numpy arrays, after applying the formulas above, the pixels will sometimes map to a negative value. In order to solve this, I had to translate the image to the right or up by the absolute value of the smallest negative number minus the new length or width of the resulting image.

Vertical Shearing: Horizontal Shearing:



**Translation:**

The translation operator performs a geometric transformation which maps the position of each picture element in an input image into a new position in an output image, where the dimensionality of the two images often is, but need not necessarily be, the same. Under translation, an image element located at (x1,y1) in the original is shifted to a new position (x2,y2)in the corresponding output image by displacing it through a user-specified translation (Bx,By). The treatment of elements near image edges varies with implementation. Translation is used to improve visualization of an image, but also has a role as a preprocessor in applications where registration of two or more images is required.

The implementation we used for this project work as we expected. We used a nested for-loop to translate pixel by pixel to the new location. This is probably the easiest way to do it but, it might not be efficient if the input image is large.



**Scaling**:

Zoom algorithms may also be used to perform image scaling, which refers to a reduction in size of the displayed digital image, and often results in a loss of image resolution. Scaling is a useful operation for formatting a digital image to fit within the bounds of a display medium, as in the case of displaying a collection of thumbnail images. The effects of scaling an image vary to a significant degree, depending on a wide spectrum of image parameters. Interpolation or averaging methods cause fine details to disappear, and sampling (showing every nth pixel) often produces aliasing artifacts.

The nearest neighbor algorithm produces a very pixelated replica of the original image. This algorithm uses only a four pixel neighborhood where the new pixel takes the intensity value of the nearest pixel. Bilinear interpolation produces a smoother image, because it uses a 4 pixel neighborhood in a much better way. It calculates the intensity of the new pixel using the distance to all 4 points as a ratio of how the intensity would be at the new pixel’s location. On the other hand, cubic or bicubic, interpolation uses a third degree polynomial to find the intensity of the new pixel. To achieve this, bicubic uses a 16 pixel neighborhood, and the result is a smooth, but sharper image compared to that of bilinear interpolation. Finally, Lanczos4 is a much larger behemoth. It requires a 64 point point neighborhood which increases the quality of the resizing by a large margin. The resized images are almost identical to the original image. That being said, my implementation of the algorithm is not perfect, but it is close enough. Overall, Lanczos4 is the superior interpolation method.

**Nearest Neighbor and Bilinear:**

**Cubic and Lanczos4:**

**GUI:**

The interface we chose to use was implemented using an inbuilt library called Tkinter.

We were able to utilize widgets that allow us to create display windows known as labels inside a larger widget. Once this was done our images were converted to TkImage objects that can be attached to and displayed on these labels.

The user may select an image from the hard disk that they wish to perform some Geometric transformation on. For rotations they are expected to enter a positive value to rotate the image clockwise, and negative value for counterclockwise rotation. Translations must be done by entering values of 0 or greater for the horizontal (x) and vertical (y) directions that they wish to move the image. All of the four interpolation methods expect x and y values greater than 0 to be entered to resize the image. Finally horizontal and vertical shearing require the input of a coefficient value of 0 or greater and the direction in which you wish to shear the image.