# Topic 1: 2D Transformation

Each image uses 24 bits to represent the color of each pixel. Eight bits are used to store the intensity of the red part of a pixel ,eight bits - the green component, and eight bits are for the blue component. Each pixel has a coordinate pair *(x,y),* which describes its position on two orthogonal axes from defined origin *O*.
To perform a succession of transformation we need to follow a sequential process −

- Translate the coordinates
- Rotate the translated coordinates
- Scale the rotated coordinates to complete the composite transformation

*We have to use 3×3 transformation matrix instead of 2×2 transformation matrix. To convert a 2×2 matrix to 3×3 matrix, we have to add an extra coordinate "u". In this way, we can represent the point by 3 numbers instead of 2 numbers, which is called **Homogenous Coordinate** system. Any Cartesian point P(x,y) can be converted to homogenous coordinates by P' (x,y,u).We will not use of homogeneous coordinates in their full generality. Here the "u"coordinate always will be 1.

$$[x, y] \quad \rightarrow \quad [x, y, 1]$$

What we need to do is take the RGB values at every *(x,y)* location, rotate it as we need, and then write these values in the new location. There is a way to get this new coordinates for each of the processes:

**Scaling**: We get scaled picture by multiplying the original coordinates of the object (x,y) with the scaling factor. For new coordinates(x', y') we multiply original (x,y) coordinates to diag matrix with entries equal to scaling factor.

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} X \\ Y \end{pmatrix} \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

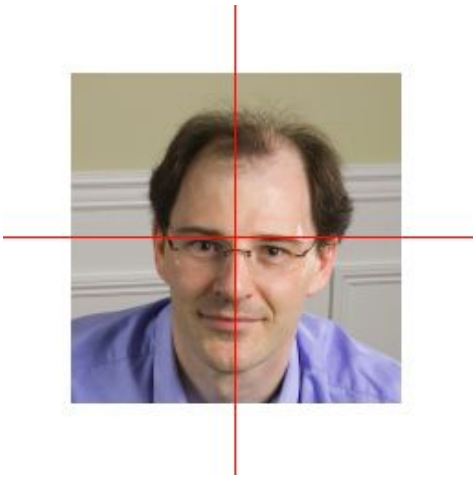
*If scaling coefficient is smaller than 1 we compress the dimensions of the object, if bigger - expand.*

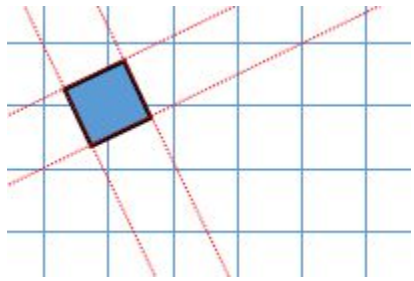
**Rotation:**.The *(x,y)* coordinates relative to the axes are the same; what we have done is rotate the coordinate frame of reference by an angle *θ(θ - angle at which we rotate(transform) our image)*. For new coordinates(x*, y*) of values at (x, y) we used this formula:

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

But if we use only this formula, we will get some white dots(or holes). This is a problem called aliasing. Multiplying by *sines* and *cosines* on the integer coordinates of the image gives real number results, and these have to be rounded back to integers again to be

plotted. Sometimes this number rounding means the same destination location is addressed more than once, and sometimes certain pixels are missed completely. When the pixels are missed, the background shows through. This is why there are holes.



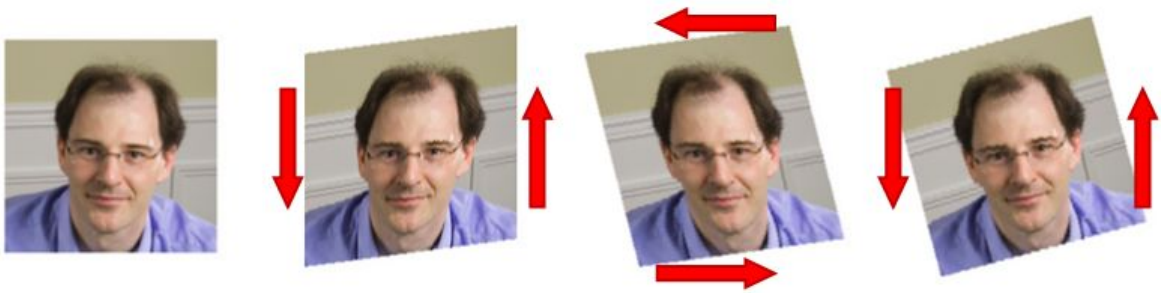


The way to fix this problem is the expansion of the single 2D rotation matrix into a three different matrices

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin\theta & 1 \end{bmatrix} \begin{bmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

- The three matrices are all shear matrices.
- The first and the last matrices are the same.
- The *determinant* of each matrix is 1.0 (each stage is conformal and keeps the area the same).
- As the shear happens in just one plane at time, and each stage is conformal in area, no aliasing gaps appear in any stage.



Source Shear #1 Shear #2 Shear #3

**Reflection** - it's a mirroring of original object. We can say that it is a rotation operation with 180°(or another way to say - 90° multiplied by integer number(2,3 or 4)). In reflection transformation, the size of the object does not change.
For this operation we use the same formula, as for rotation. The angle $\theta$ is 90°, 180° or 270°.

**We used built-in Python fuctions:**

- library SciPy, scipy.ndimage.interpolation.rotate (function rotate)
- mathplotlib.pyplot(for representing new matrix)
- matplotlib.image(for reading image)
- scipy.misc.imresize(for scaling)

***Input:** image, rotation angle/ scaling cooeficient
**Output:** transformed image