Translations, Scaling, Rotations for 2D images using Matrix operations

- **By Gattadi Vivek (2019MCB1217)**
- **2nd year, BTech Mathematics and Computing, IIT Ropar**

CONTENTS:

INTRODUCTION:

1. What is an Image in the Digital world and how it is formed?

Digital Images are made of picture elements known as pixels. Pixels are tiny squares that are colored which collectively form an image.

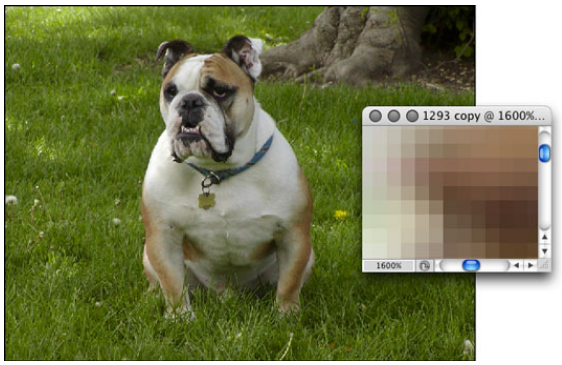


Figure 1: It's not a Bulldog! It's a collection of tiny colored squares!

As you can see, if you zoom in on a particular area, you can see tiny colored squares which are known as pixels.

2. Representation of a Pixel:

- An image is broken down into the smallest piece of information i.e. pixel, which has a single color.
- Each color can be represented as a number such as Hex, Decimal, Binary, etc.
- Pixels are identified by their location within the coordinate grid.

- Color Encoding System:

- By using a prism, Newton proved that white light consists of pure, continuous colors.
- He identified these colors as red, orange, yellow, green, blue, indigo, and violet. You see these colors naturally in Rainbows.

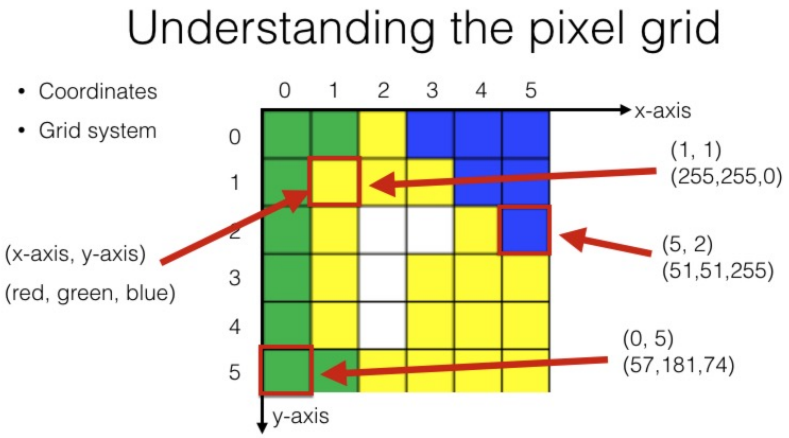
The fundamental colors are Red, Green, Blue known as RGB colors. We need a way to encode the colors, i.e. a color encoding system. If you combine red, blue, and green, you can represent a wide range of colors. This is known as the “RGB Color Scheme”.

The following are the types of representations:

	BASE	RED	GREEN	BLUE
1.	Decimal	255	0	0
2.	Octal	37	0	0
3.	Hex	FF	00	00
4.	Binary	1111 1111	0000 0000	0000 0000

- **Pixel Grid:**

  Pixels are assembled in a grid system. Each one has coordinates. Each pixel is specified by its position within the grid system as identified by (x-axis, y-axis).



- **Resolution:**

  Resolution is determined by the width and the height of the image. For example, for a 400px x 400px image, the resolution is 400 x 400 = 1,60,000. This can also be expressed as 0.16 megapixels.

THEORY:

- AFFINE TRANSFORMATION:

- An Affine Transformation is any transformation that preserves collinearity and ratios of distances.
- Geometric Contraction, Expansion, Dilation, Reflection, Shear, **Rotation**, Similarity transformations, Spiral similarities & **Translations** are all examples of Affine Transformations.
- In general affine transformation is a composition of Rotation, Translations, Dilations, and Shears.
- In general, affine is a generalization of congruent and similar.

- **General Form**

In a 2D plane, if the point (x,y) is transformed to another point say (x', y'), then the new point can be represented as below:

x' = ax + b = ax + 0y +b
y' = cy + d = 0x + cy + d

If we want to represent the above equation in a matrix form, then we can augment one more equation, i.e.

1 = 0x + 0y + 1

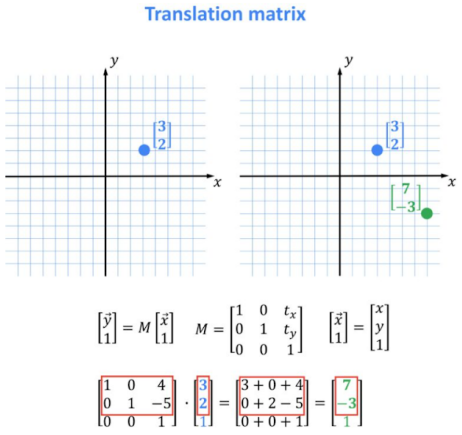
To get the following general transformation matrix in a 2D plane:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & 0 & b \\ 0 & c & d \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

1. **TRANSLATION:**

- In simple words, Translation is nothing but a moment in a specific direction.
- In an image, when we translate each pixel to 'h' units right(or left depending on the sign of h) and 'k' units up(or down depending on the sign of k), the image moves 'h' units horizontally and 'k' units vertically.

- **Translation of a point in a 2D plane:**



Here, M is our transformation matrix.

We can see that pixel [3 2] now have coordinates [7 −3]. So, a pixel is shifted 4 pixels to the right and 5 pixels down. It is good to remember that the positive values of t_x will shift the point to the right and negative to the left. Similarly, positive values of t_y will shift the image up and negative down.

- **ALGORITHM:**

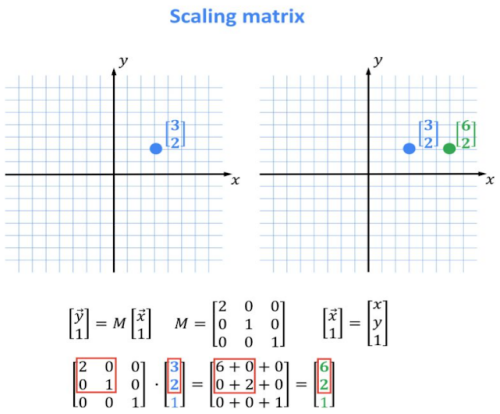
  Generally, our original image has its left bottommost pixel at the origin wrt our image. But when we move each pixel horizontally by 'h' units and 'k' units vertically, then the entire image moves 'h' units right in the same direction and 'k' units vertically in the same direction.

  - If any pixel has become a void spot after the translation of the original pixel then the void pixel is mapped to black color.
  - If 'h' or 'k' is not an integer, then take the integer part of it and assign the position coordinate to the pixels.

2. **SCALING:**

   - The Scaling operator performs a geometric transformation which can be used to shrink or zoom the size of an image.

- **Scaling of a point in a 2D plane:**



Here, M is our transformation matrix.

We can see that pixel [3 2] now have coordinates [6 2]. So, here the x-coordinate is multiplied by 2 and the y-coordinate has remained the same as it is multiplied by 1.

- **ALGORITHM:**

There are many algorithms proposed for Scaling. Here our main focus is on the **Nearest Neighbour Algorithm**.

- **Nearest Neighbour Algorithm:**

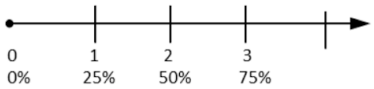
Let us start with an example to get the intuition. Consider the following 4x4 pixel grid of an image.

	0	1	2	3
0	255,255,255	206,0,0	10,0,206	39,206,0
1	249,222,6	2,94,13	254,155,38	140,9,9
2	171,6,248	172,250,6	6,240,249	236,6,250
3	192,192,192	128,128,128	64,64,64	0,0,0

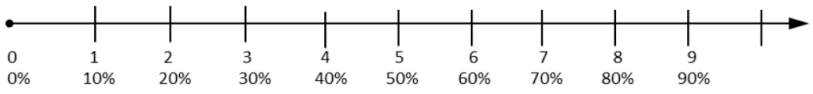
Now we wish to scale it to a 10x10 image. To do this we would obviously need to reuse the pixels of the original image more than once. So the question is which pixel to replicate/ reuse and in which location.

- **Linear Interpolation:**

The nearest neighbour algorithm is based upon linear interpolation. Consider the 1st row of the above as a single line. Each point can be treated as a percentage of the distance of the line length i.e. divide each point by the length of the line which is the width of the image which is equal to 4.



The 1st row of the 10x10 scaled image to be created can be considered in the same manner.



To determine which value from the 1st line to copy when forming the 2nd line, we use the nearest corresponding percentage value from the 1st line.

- If we want to know the 1st value of line-2, we first divide its value by the length of the line = 0.10 which is the percentage of its distance along the line.
- Now we multiply this percentage by the length of the 1st line: 0.10 x 4 = 0.40.
- Since pixel coordinates are integers we round off the number to its nearest integer. Therefore 0.4 rounds to 0.

	0	1	2	3
0	255,255,255	206,0,0	10,0,206	39,206,0
1	249,222,6	2,94,13	254,155,38	140,9,9
2	171,6,248	172,250,6	6,240,249	236,6,250
3	192,192,192	128,128,128	64,64,64	0,0,0

Figure 2a: 4x4 size original image

x (10x10)	x interpolated	x rounded	nearest pixel
0	0.0	0	255,255,255
1	0.4	0	255,255,255
2	0.8	1	206,0,0
3	1.2	1	206,0,0
4	1.6	2	10,0,206
5	2.0	2	10,0,206
6	2.4	2	10,0,206
7	2.8	3	0,0,0
8	3.2	3	0,0,0
9	3.6	4	0,0,0

Figure 2b: 4x4 image scaled to 10x10

Note: In figure 2b the last point(3.6) should be rounded off to 4. However, no pixel exits for value 4. Hence we use the previous value for it.

In short, the algorithm follows below:

- If the matrix A of dimension m1*n1 is scaled to m2*n2 then:
  1. First, divide the x-axis into m1 equal parts. Associate each point with its percentage of the distance of the line length.
  2. Now divide x-axis by m2 equal parts. Associate each point with its percentage of the distance of the line length.
  3. Now multiply each point in the 2nd line by m1.
  4. Now round off each part to its nearest integer. Care must be taken near the endpoints as mentioned above.
  5. Color all the pixels with the same color for the same integer value.
  6. Repeat the same process for the y-axis.

Note: The above algorithm works only for **upscaling** the image size.

- For **downscaling** of the image, we use an algorithm called **Box Filter Algorithm.**

  - **Box Filter Algorithm:**

    Here also, let us start with an example to understand the algorithm. Before getting into the algorithm let's go through a few terminologies.

    - A Source image is an original image before making a scaling operation on it.
    - Target image refers to the scaled image.
    - The region of the target pixel in the source that is being calculated currently is called a filter window.

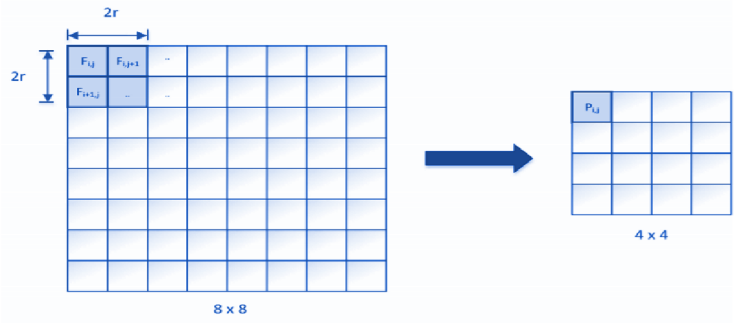


Figure 3: Downscaling of an image.

In this algorithm, each pixel that is to be calculated is the average of pixel values in the filtered window. Hence all the pixels in the target image are filled in this fashion.

In general, the algorithm is as follows:

- If the matrix A of dimension m1*n1 is downscaled to m2*n2 then:
    1. Let **a** = int(m1/m2), **b** = int(n1/n2).
    2. Now to fill the first pixel in the targeted image. Take '**a**' pixels horizontally and '**b**' pixels vertically from the top left corner. Take the average pixel value of R, G, B for all 'a x b' pixels and fill it in the top left corner of the targeted image.
    3. Now move '**a**' units right in the targeted pixel and take next '**a**' units right and '**b**' units down and take the average pixel value and fill it right to the pixel which was calculated before.
    4. Similarly, fill it till the end. After completing the row, move b units down and repeat the same process.

- **Note:** Since m1/m2, n1/n2 need not be integers, so due to this, there may be few pixels left out in the right end and bottom end which are not used in the calculation. But this doesn't make any to the image quality.

3. SHEAR:

In the 2D plane, a horizontal shear is a function that takes a point (x,y) to (x + my, y); where m is a fixed parameter called the **shear factor.** So every point is moved horizontally by an amount proportional to the y-coordinate. If a point is above the x-axis, then it is displaced to the right if m > 0, left if m < 0. If it is below the x-axis, then it is displaced to the right if m < 0, left if m > 0. Due to this transformation, the straight lines parallel to the x-axis remain where they are, while all other lines are turned about a point where they cross the x-axis. Vertical lines, in particular, become oblique lines with slope = 1/m.

For vertical shear (x,y) is transformed to (x, y + mx). And the same analogy carries here also.

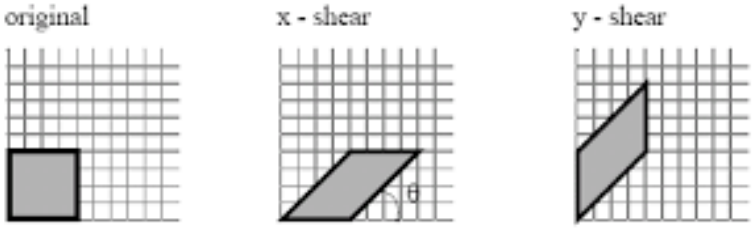


Figure 4: Horizontal and vertical shears.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Figure 5a **Figure 5b**

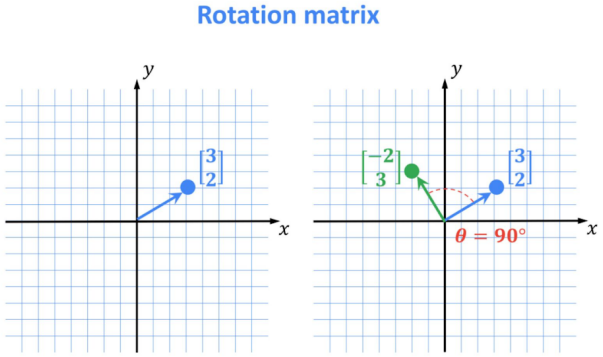
Figure 5: Transformation matrix for horizontal shear (5a) and vertical shear (5b).

- **ALGORITHM:**
  - **Case 1: Shear factor is a number greater than or equal to 1:**
    - Use the Linear interpolation technique along the shear direction for all the x-values in the original image.
  - **Case 2: Shear factor is a number less than 1:**
    - Use the Box Filter technique along the shear direction for all the x-values in the original image.

4. ROTATION:

- The Rotation operator performs a geometric transformation that can rotate an image at a specified angle.

- **Rotation of a point in 2D plane:**



Here M is our Rotation matrix.

$$\begin{bmatrix}\vec{y}\\1\end{bmatrix} = M\begin{bmatrix}\vec{x}\\1\end{bmatrix} \quad M = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \quad \begin{bmatrix}\vec{x}\\1\end{bmatrix} = \begin{bmatrix}x\\y\\1\end{bmatrix}$$

$$\begin{bmatrix}\cos 90 & -\sin 90 & 0\\ \sin 90 & \cos 90 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix} \cdot \begin{bmatrix}3\\2\\1\end{bmatrix} = \begin{bmatrix}0-2+0\\ 3+0+0\\ 0+0+1\end{bmatrix} = \begin{bmatrix}-2\\3\\1\end{bmatrix}$$

Hence the point [3 2] is transformed to [-2 3] when rotated by 90 degrees anti clockwise.

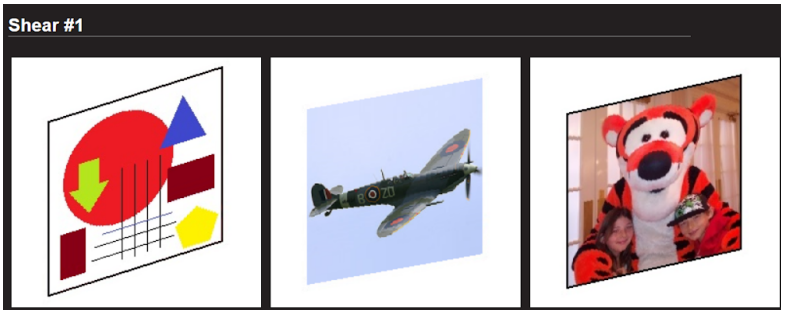
- **ALGORITHM:**
    - **Three Shears Algorithm:**

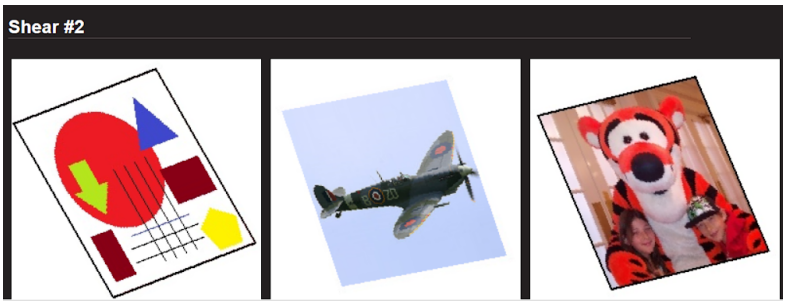
The above transformation matrix M can be decomposed as follows:

$$\begin{bmatrix}x'\\y'\end{bmatrix} = \begin{bmatrix}1 & \tan\theta/2\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ -\sin\theta & 1\end{bmatrix}\begin{bmatrix}1 & tan\theta/2\\ 0 & 1\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}$$

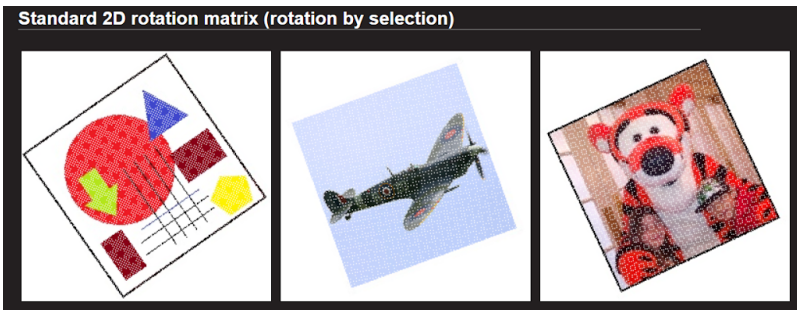
There are some very interesting properties of these three matrices:

- All three matrices are shear matrices.
- The first and last matrices are the same.
- The determinant of each matrix is 1 (each stage is conformal and keeps the area the same).
- As the shear happens in one plane at a time, and each stage is conformal in area, no aliasing gaps appear in any stage.





Now you can see the advantages of this algorithm over another general algorithm in the last shear phase.



- **Python Code using OpenCV library:**

```
import numpy as np
import cv2


def translation(img):
    height, width = img.shape[:2]
    x_shift_pos = 0
    y_shift_neg = height/3
    T = np.float32([[1, 0, x_shift_pos], [0, 1, y_shift_neg]])
    img_translation = cv2.warpAffine(img, T, (width, height))
    '''cv2.imshow('original image',img)'''
    cv2.imshow('translated image', img_translation)
    cv2.waitKey(0)


def scaling(img):
    height, width = img.shape[:2]
    scaled_img = cv2.resize(img, (int(width/2), int(height/2)), interpolation=cv2.INTER_CUBIC)
    cv2.imshow('scaled', scaled_img)
    cv2.waitKey(0)


def rotation(img):
    rows, cols = img.shape[:2]
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1)
    rotated_img = cv2.warpAffine(img, M, (cols, rows))
    cv2.imshow('rotated', rotated_img)
    cv2.waitKey(0)


img = cv2.imread(r'C:\Users\Gattadi Vivek\Desktop\swan.jpg', 0)
rotation(img)
scaling(img)
translation(img)
```

USES:

1. Translation is used for cropping the image.
2. Scaling is used for zooming in and out of the image to get better insights of the image.
3. Rotation for rotating the image.
4. These three combinely is used to correct geometric distortions that occur in real world.

REFERENCES:

1. Wikipedia (https://en.wikipedia.org/wiki/2D_computer_graphics)
2. Wolfram mathematics (https://mathworld.wolfram.com/Translation.html), (https://mathworld.wolfram.com/AffineTransformation.html)
3. Threeshear (https://www.google.com/search?q=threeshear+algorithm+in+image+processing&oq=threeshear+algorithm+in+image+processing&aqs=chrome.0.69i59j0i333l4.2524j0j9&sourceid=chrome&ie=UTF-8)
4. GeeksforGeeks (https://www.geeksforgeeks.org/python-opencv-affine-transformation/?ref=lbp)
5. Images from Google

- **It was a wonderful experience from this course and I have really enjoyed and learnt alot from this course...**

  Thank you Manju Mam, Sairam Sir, Arun Sir

---------- **Thank You** ----------