Graphics Manipulation

Graphics Manipulation

The manipulation of images

We will consider scaling, translation, rotation, etc.

Affine and projective transformation

Can be applied to bitmaps.

Graphic Manipulation

Manipulation achieved as a series of transformations.

These transformation achieved by matrices.

Called transformation matrices.

Complex manipulation is achieved by combination (multiplication) of the basic transformation matrices.

We will see that the order of multiplication is important.

Moving pixels

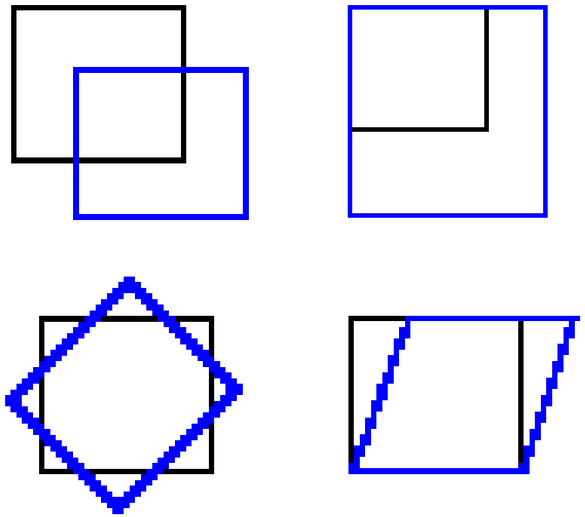
Manipulating a picture using affine transformation

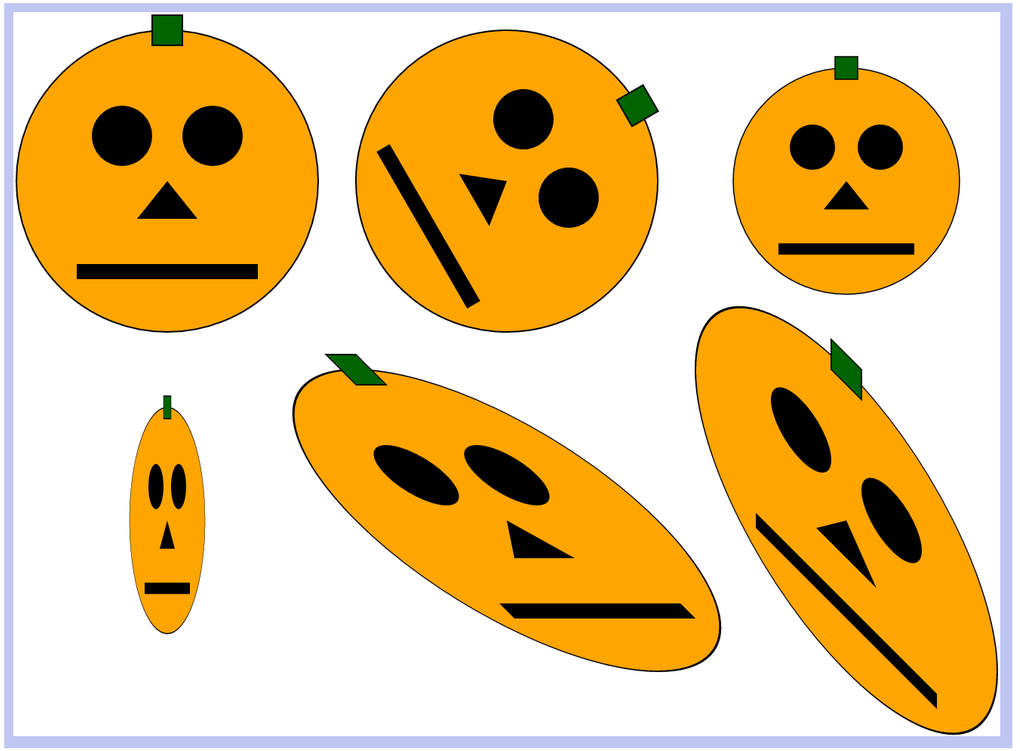
In bitmaps we manipulate each pixel in the image.

Affine transformation is a linear mapping method that preserves points, straight lines, and planes. Sets of parallel lines remain parallel after an affine transformation.

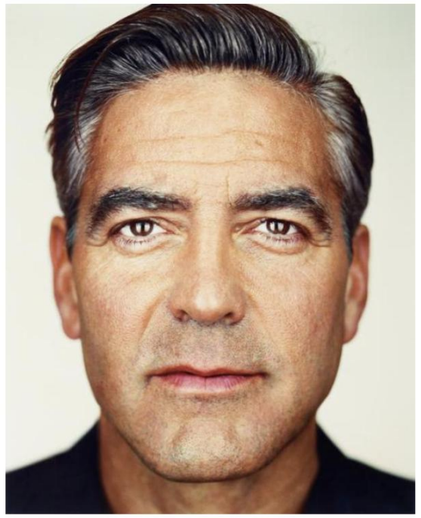
There are three basic operations:

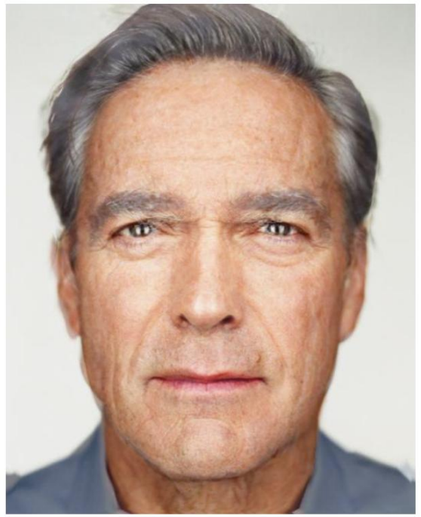
- Translation
- Scaling
- Rotation
- Shearing

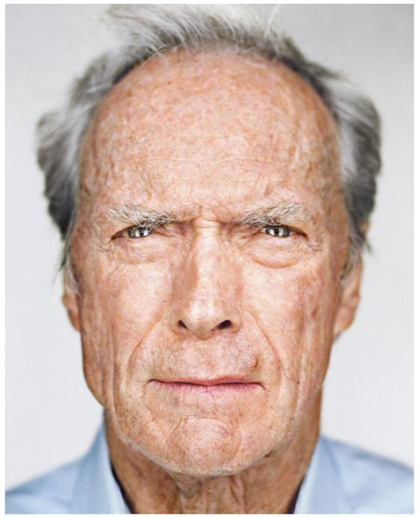


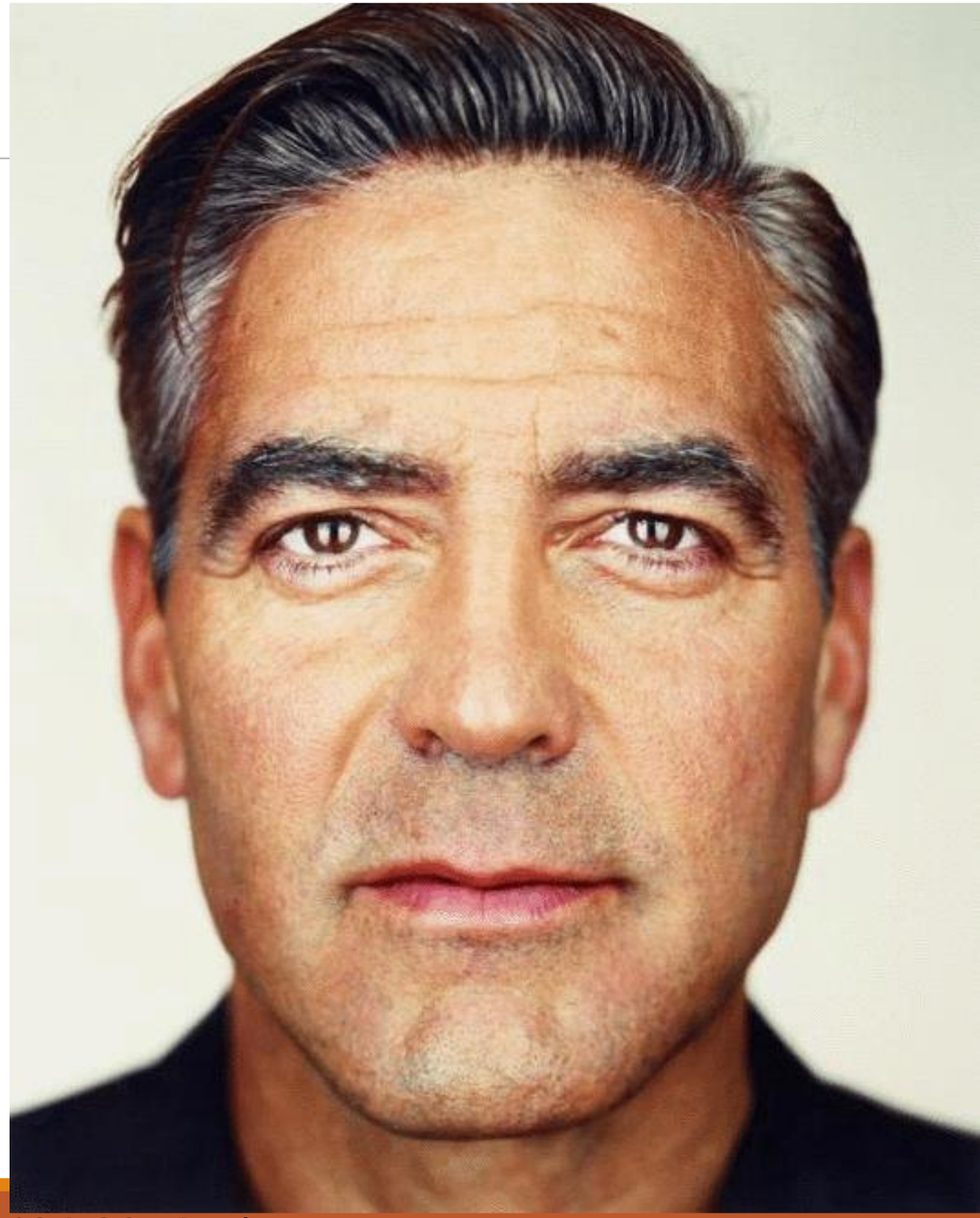


Face Morphing



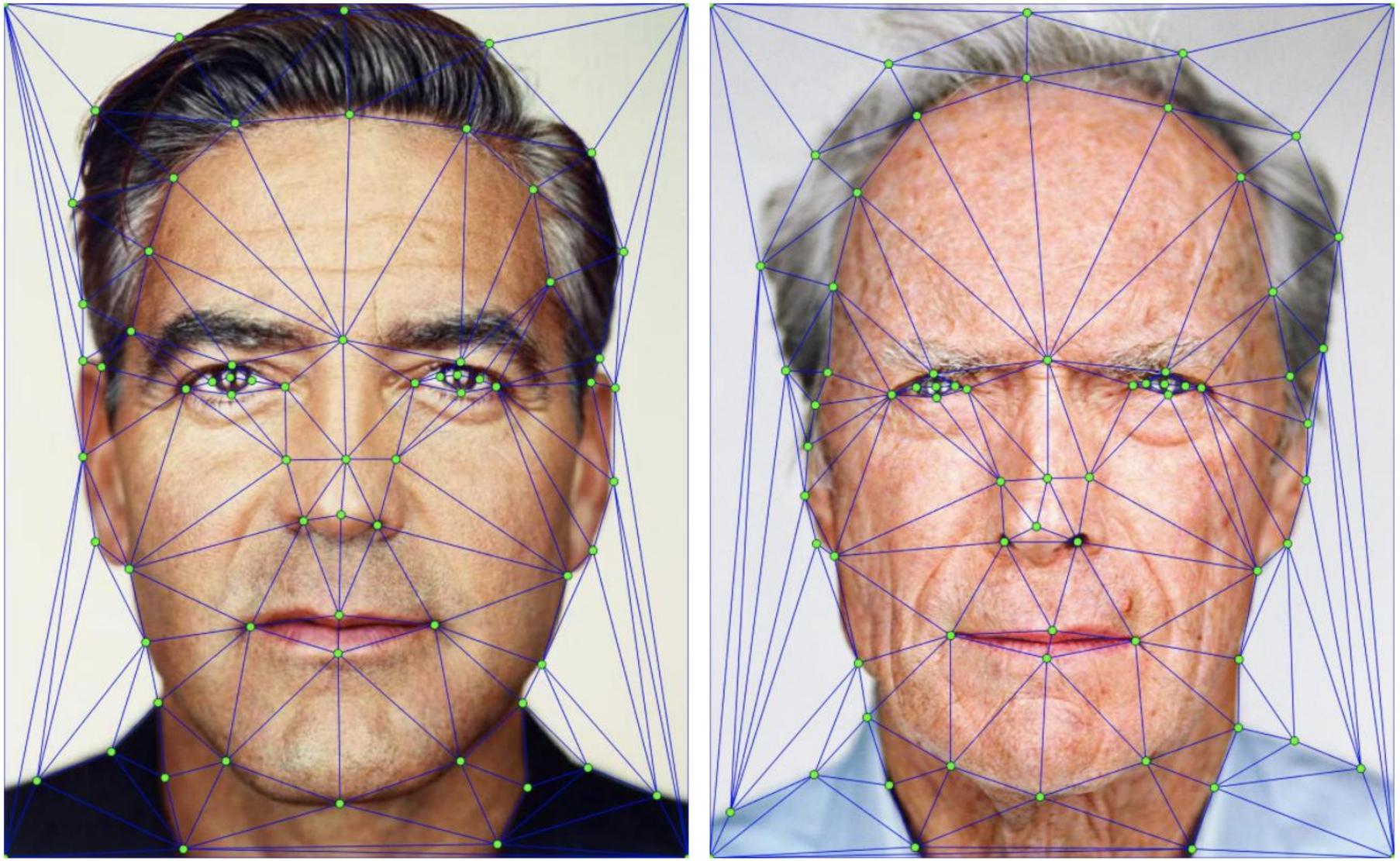






https://inst.eecs.berkeley.edu/~cs194-26/fa17/upload/files/proj4/cs194-26-abw/

Face Morphing



https://inst.eecs.berkeley.edu/~cs194-26/fa17/upload/files/proj4/cs194-26-abw/

The equations of manipulation

Translation

$$\begin{aligned} x' &= x + T_x \\ y' &= y + T_y \end{aligned}$$

Scaling.

$$\begin{aligned} x' &= x \times S_x \\ y' &= y \times S_y \end{aligned}$$

Rotation.

$$\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{aligned}$$

Matrix multiplication

Multiply each element of a row of the first matrix by each element of the corresponding column of the second matrix.

Add the products together.

Put the result in position (row of first matrix, column of second matrix) of a new matrix.

Repeat for all rows of the first matrix.

Means that the number of rows of the first matrix must be the same as the number of columns of the second matrix.

Easy for computers.

Matrix multiplication

- Example:

$$\begin{bmatrix} 5 & 8 \\ 7 & 2 \end{bmatrix} \times \begin{bmatrix} 6 & 1 \\ 4 & 3 \end{bmatrix}$$

- Try it with Matlab.

A=[5 8 ; 7 2]

B=[6 1 ; 4 3]

A*B

$$= \begin{bmatrix} (5 \times 6) + (8 \times 4) & (5 \times 1) + (8 \times 3) \\ (7 \times 6) + (2 \times 4) & (7 \times 1) + (2 \times 3) \end{bmatrix}$$

Try B*A

Not the same.

$$= \begin{bmatrix} 62 & 29 \\ 50 & 13 \end{bmatrix}$$

- Try this tutorial if you don't get it.

https://www.youtube.com/watch?v=kuixY2bCc_0

The equations in a matrix form.

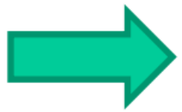
Scaling as an example:

$$\begin{aligned} x' &= x \times S_x \\ y' &= y \times S_y \end{aligned}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ & S_y \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}$$

But…

$$\begin{aligned} x' &= ax + by + t_x \\ y' &= cx + dy + t_y \end{aligned}$$



$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Homogeneous Coordinates

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

“Subsequent” operations are inserted here, by pre-multiplying

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

The equations in a “homogeneous” matrix form.

Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Identity

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Reflection

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear-X

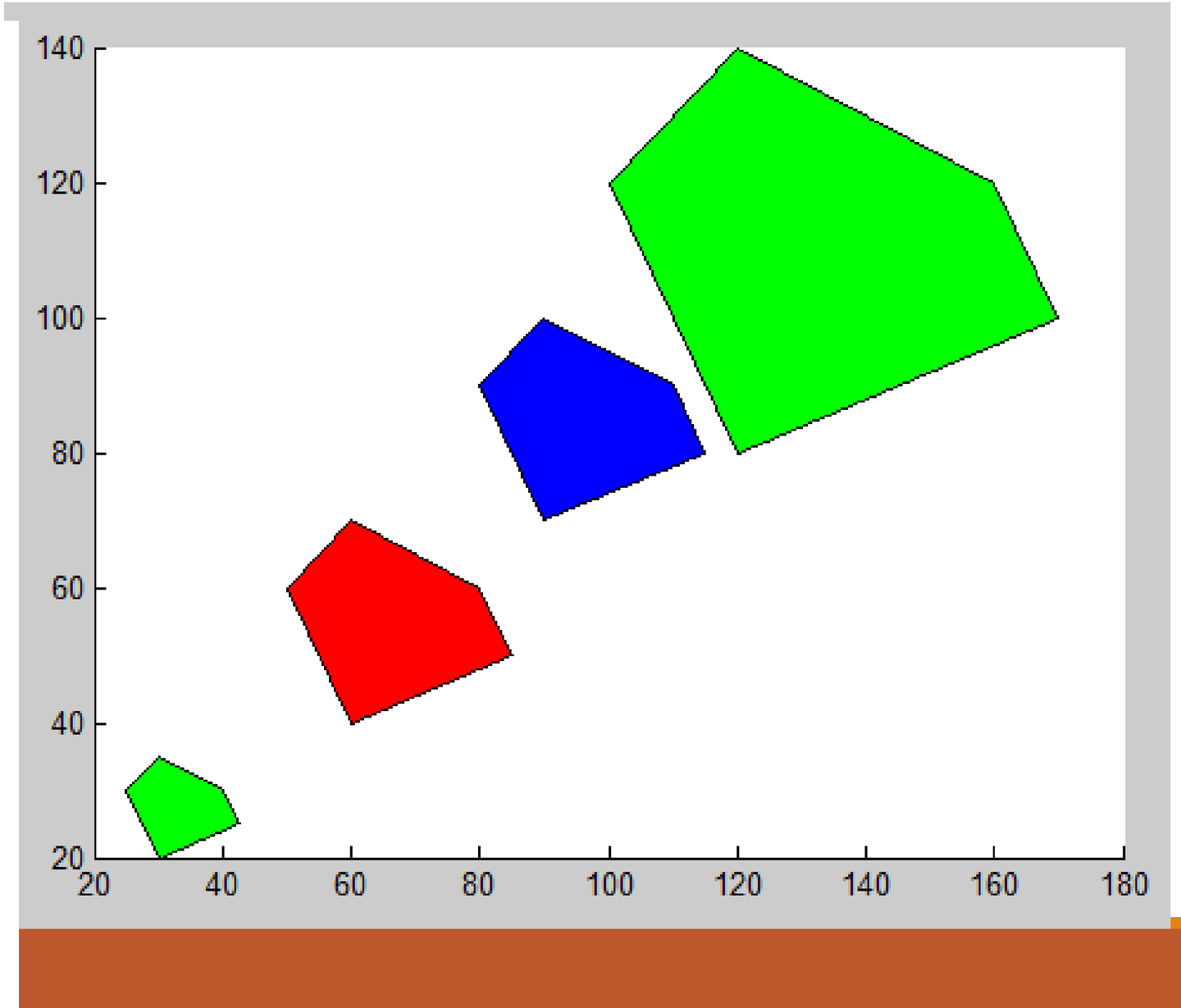
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \lambda_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear-Y

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \lambda_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scaling & transforming in Graphics system

```
%MATLAB CODE
xy=[50 60 80 85 60;  60 70 60 50 40; 1 1 1 1 1];
x=xy(1,1:5);
y=xy(2,1:5);
patch(x, y, 'r');
%translate the shape
RT=[1 0 30; 0 1 30; 0 0 1];
   xyp=RT*xy;
   x=xyp(1,1:5);
   y=xyp(2,1:5);
patch(x, y, 'b');

%Scale up the shape
RT=[2 0 0; 0 2 0; 0 0 1];
   xyp=RT*xy;
   x=xyp(1,1:5);
   y=xyp(2,1:5);
patch(x, y, 'g');

%Scale down the shape
RT=[0.5 0 0; 0 0.5 0; 0 0 1];
   xyp=RT*xy;
   x=xyp(1,1:5);
   y=xyp(2,1:5);
patch(x, y, 'g');
```



Compound Transformations

By multiplying different types of transformation matrix, we can do two (or more) transformations at once. Effectively this transforms the already transformed image.

For example:

Scaling followed by rotation.

$$scale = \begin{bmatrix} 0.5000 & 0 & 0 \\ 0 & 0.2500 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$rotate = \begin{bmatrix} 0.7070 & -0.7070 & 0 \\ 0.7070 & 0.7070 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$compound = rotate \times scale$$

$$= \begin{bmatrix} 0.7070 & -0.7070 & 0 \\ 0.7070 & 0.7070 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0.5000 & 0 & 0 \\ 0 & 0.2500 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.3535 & -0.1767 & 0 \\ 0.3535 & 0.1767 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Compound Transformations

For example:

Scaling followed by translation.

$$scale = \begin{bmatrix} 0.5000 & 0 & 0 \\ 0 & 0.2500 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$translate = \begin{bmatrix} 1 & 0 & 20 \\ 0 & 1 & 50 \\ 0 & 0 & 1 \end{bmatrix}$$

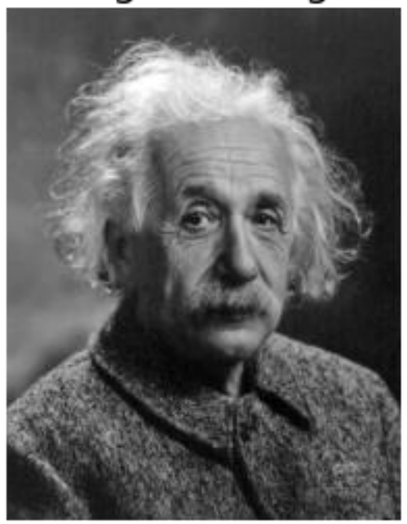
$$compound = translate \times scale$$

$$= \begin{bmatrix} 1 & 0 & 20 \\ 0 & 1 & 50 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0.5000 & 0 & 0 \\ 0 & 0.2500 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

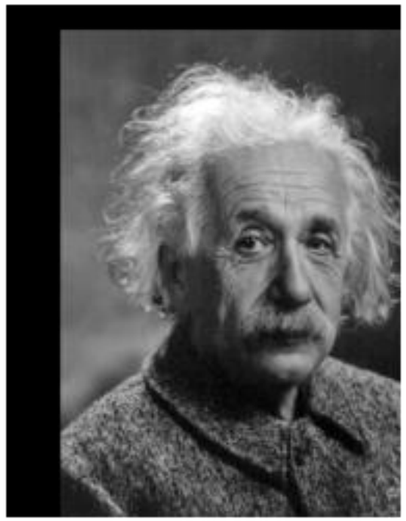
$$= \begin{bmatrix} 0.5000 & 0 & 20 \\ 0 & 0.2500 & 50 \\ 0 & 0 & 1 \end{bmatrix}$$

Affine transformation

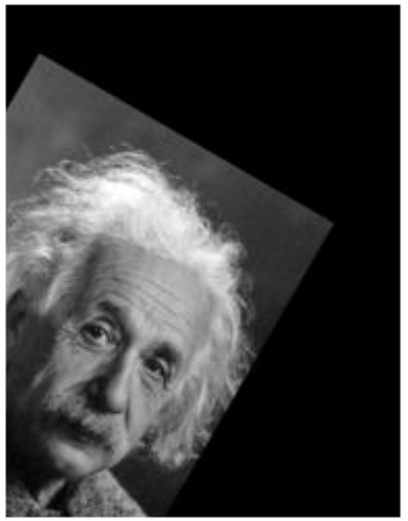
Original image



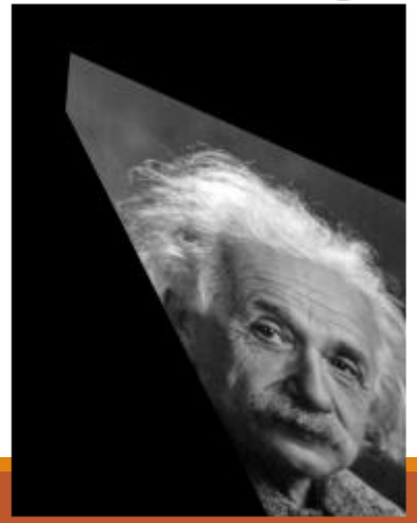
First Translation



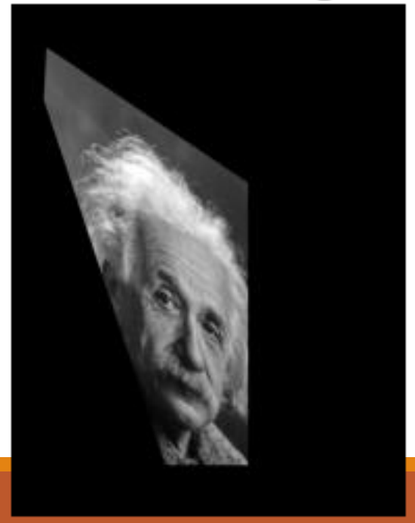
Then rotation



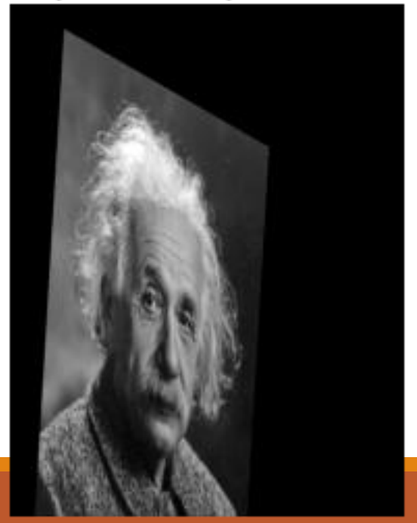
Then shearing



Then scaling



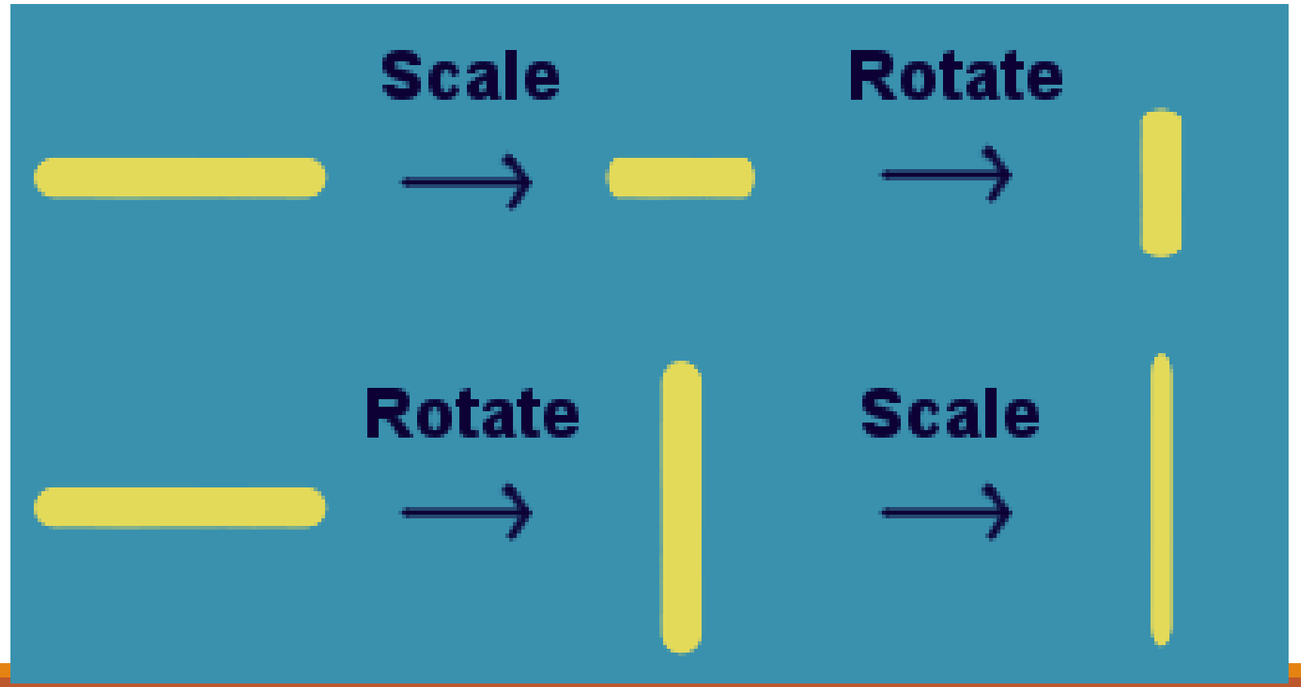
Compound operation ??



Order of transformation matrices

Note that Scaling followed by translation is not the same as translation followed by scaling, because we scale the translation in the second case.

Rotation followed by scaling distorts the transformation matrix and results in “skew”.

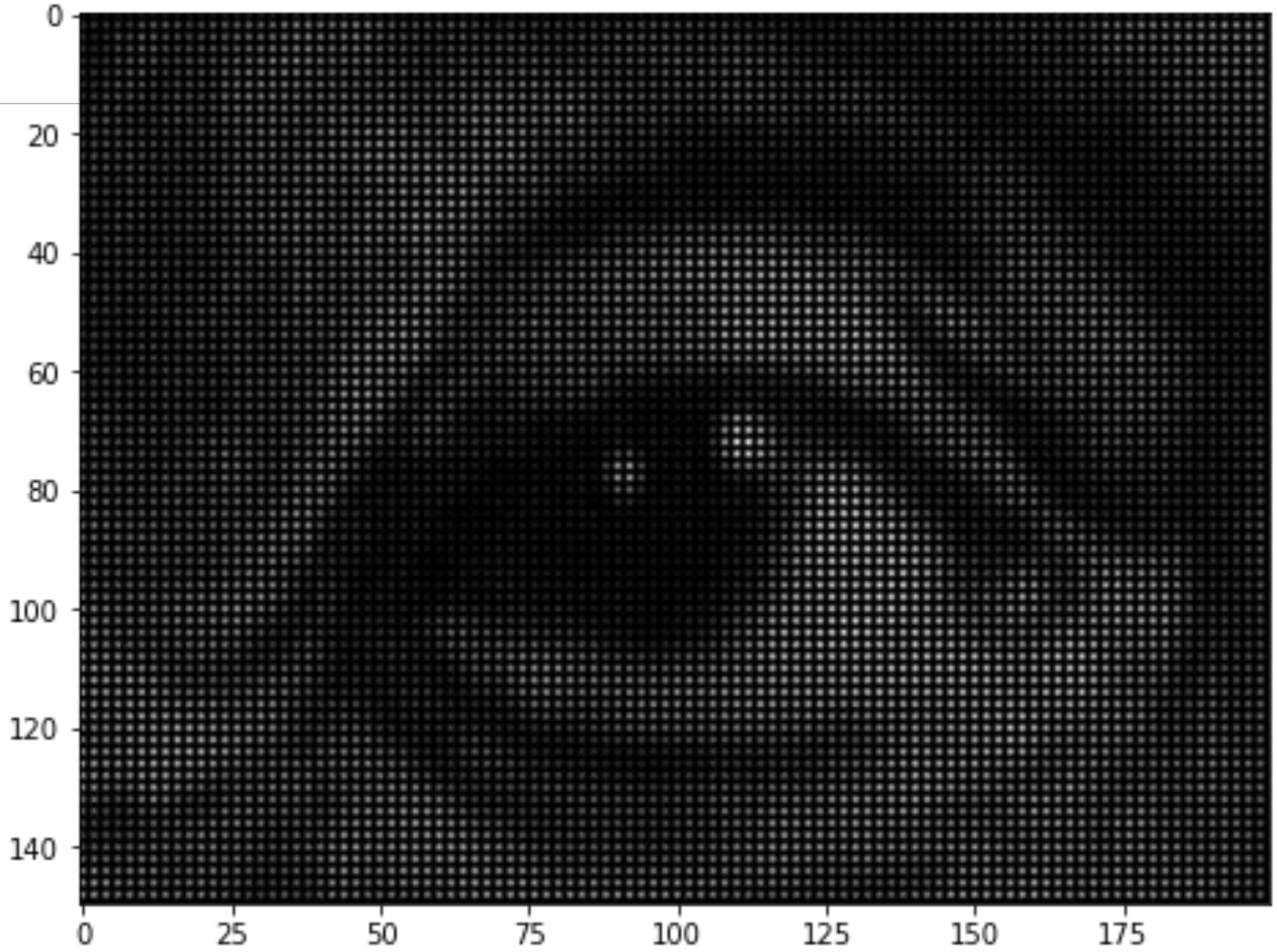


Order of transformation matrices

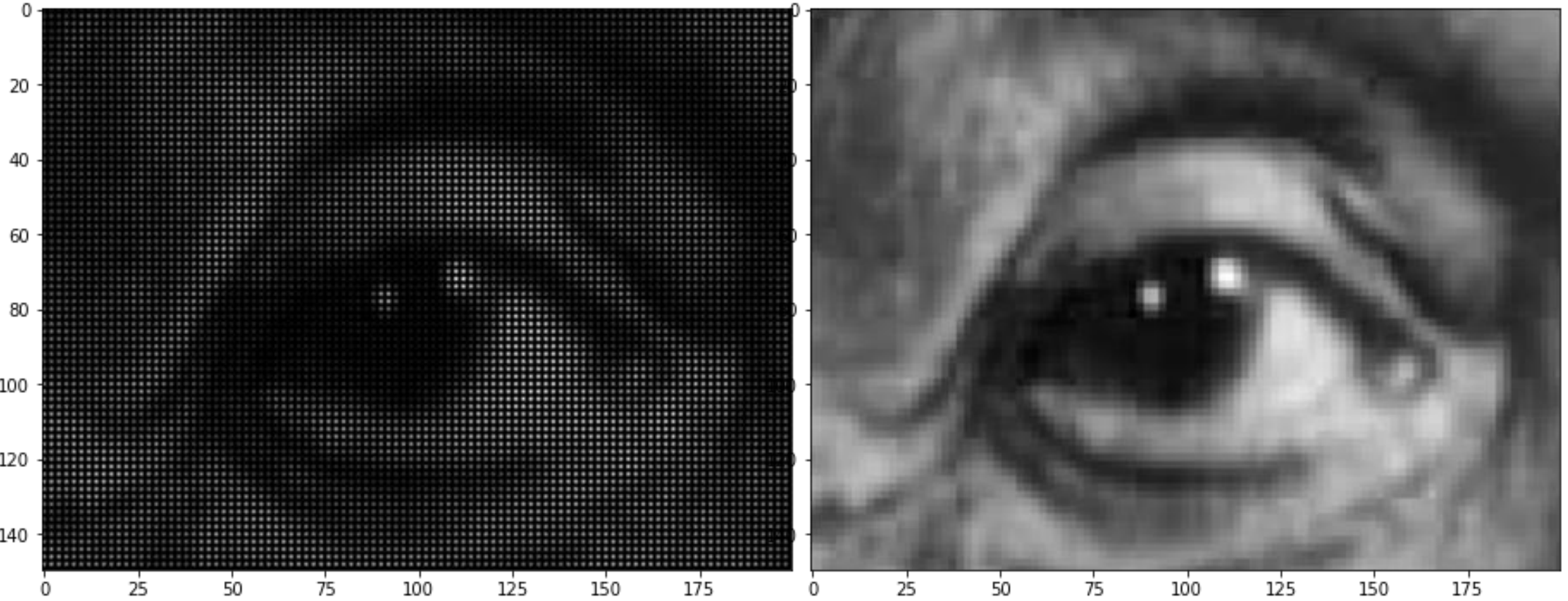
This is true of all compound transformations, so the order is important.

Also, because the transformation matrix is multiplied by the object to be transformed, rather than the other way around, “later” operations are on the left of any expression. This may look like the wrong way round.

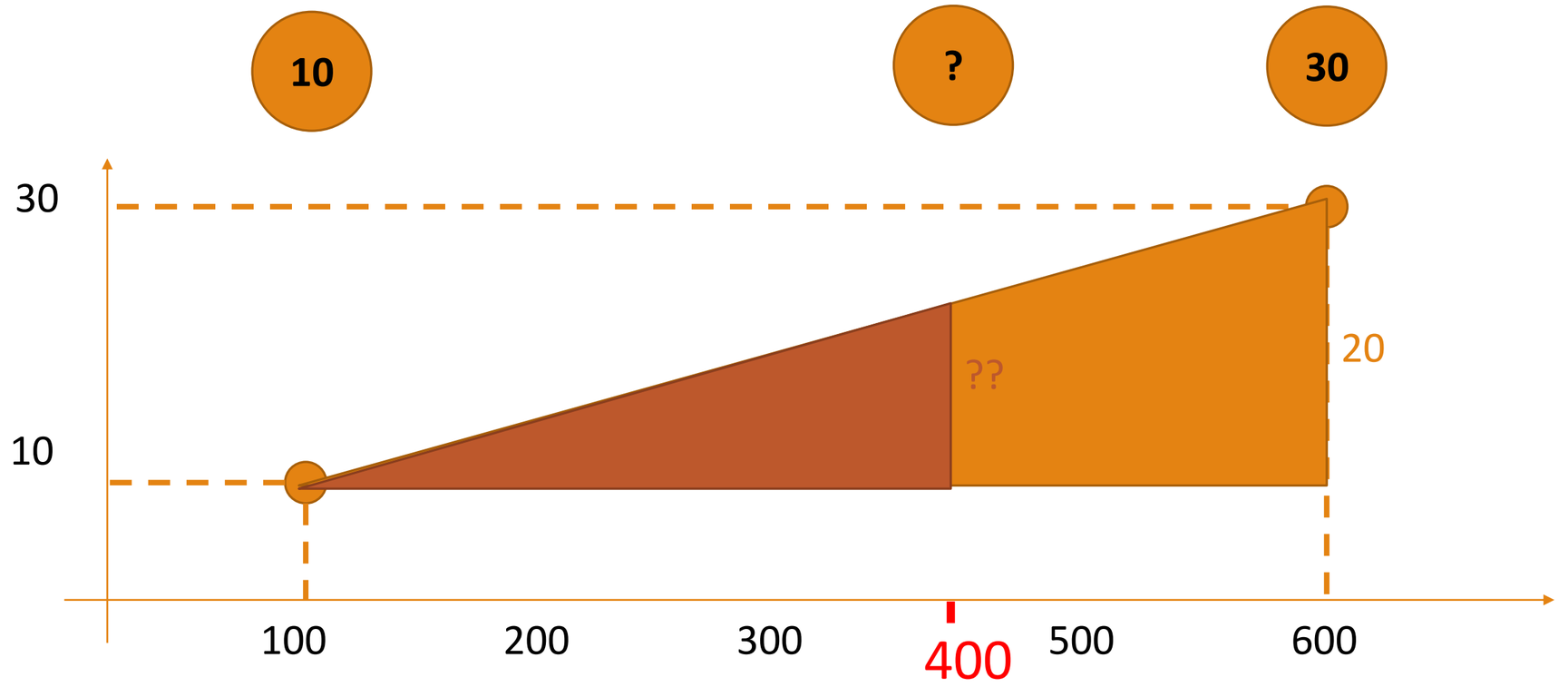
Scaling x2



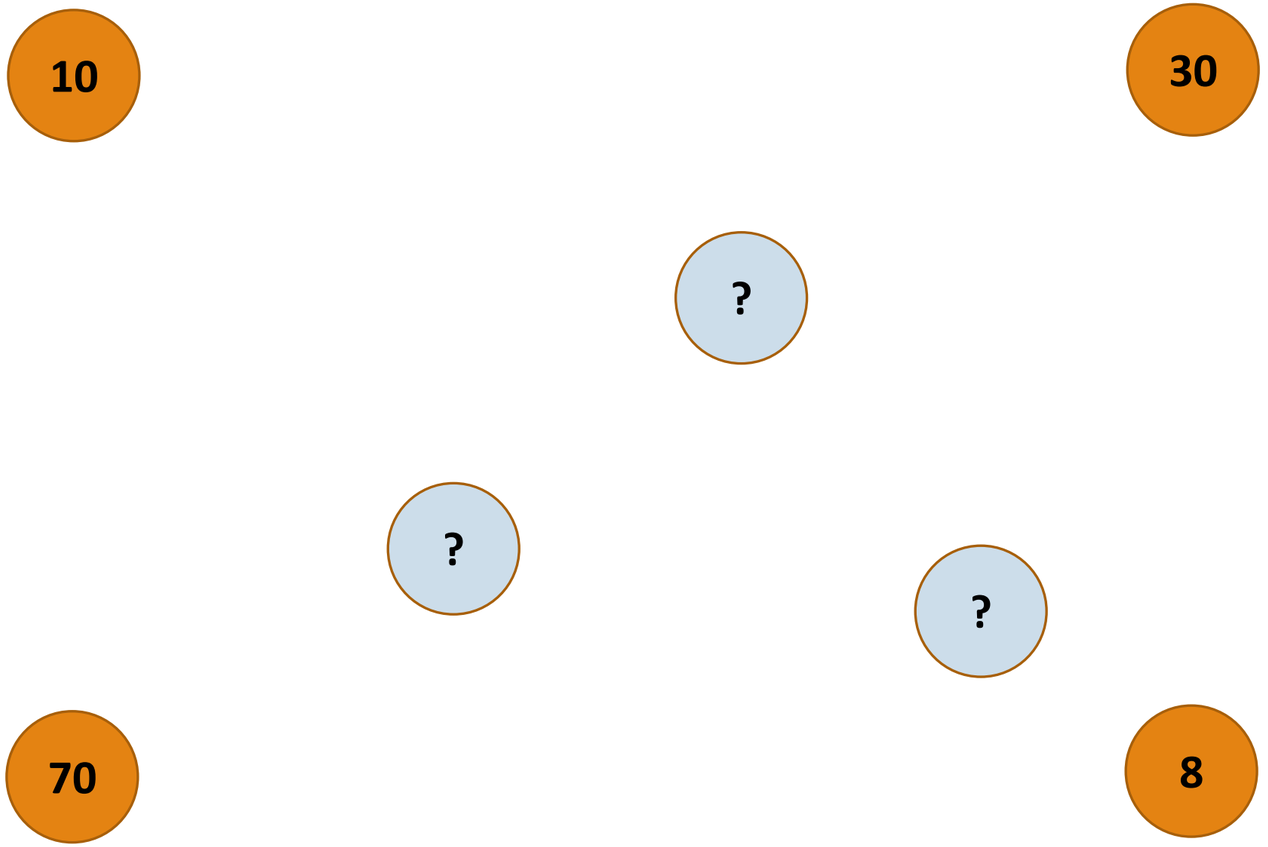
Fill the holes with Nearest neighbours



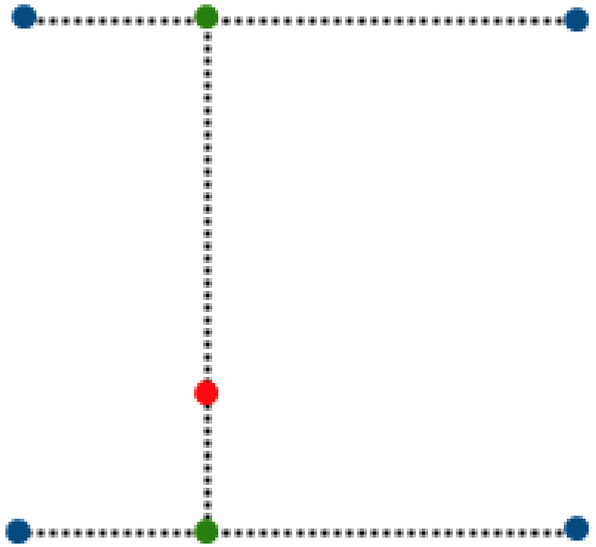
Linear and Bilinear interpolation



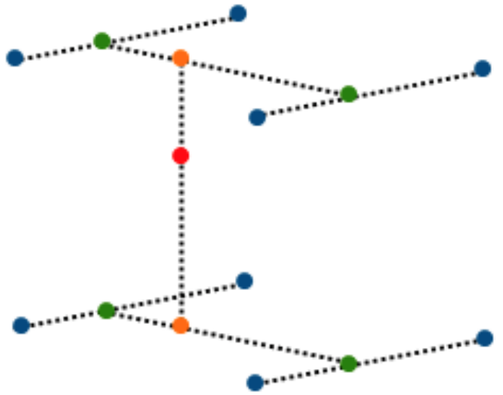
Interpolation is a method of estimating a value from a set of given values.


10
30
?
?
?
70
8

Linear and Bilinear interpolation



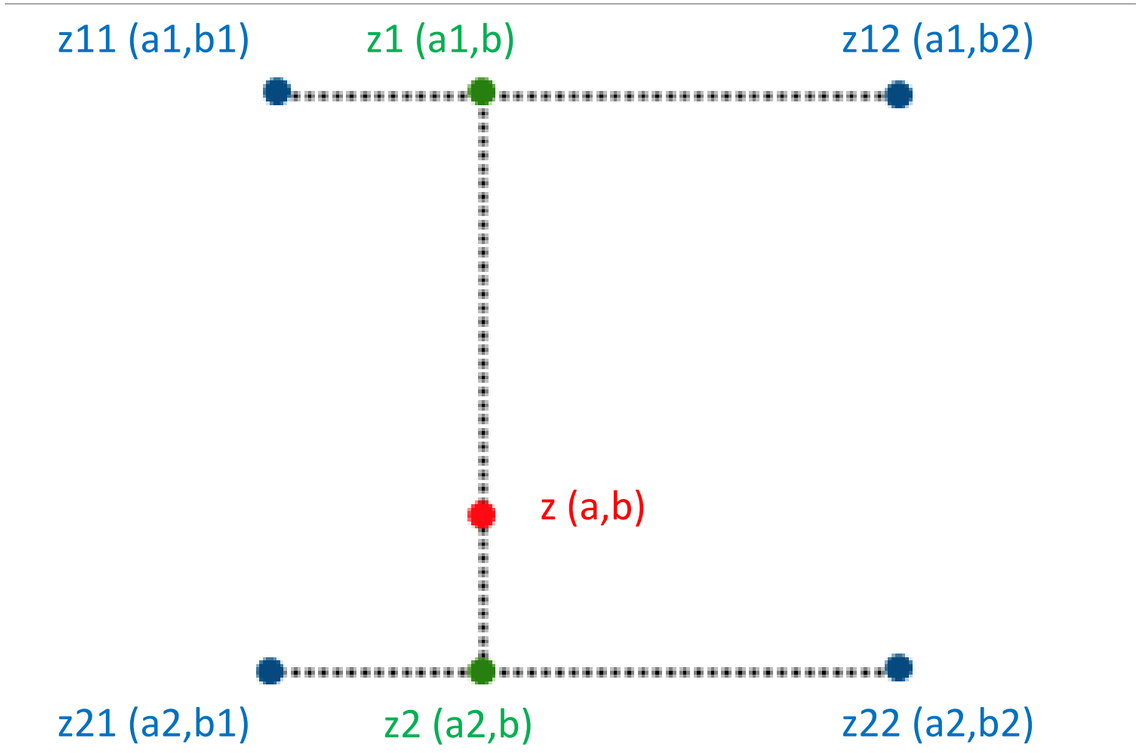
Stage 1 (original values)
Stage 2 horizontal interpolation
Stage 3 vertical interpolation (final)



Total LERPs = 4 + 2 + 1
= 7

“Bilinear” means there are 2 directions to interpolate. In our case, we’re interpolating between 4 pixels. Visualise each pixel as a single point. Linearly interpolate between the top 2 pixels. Linearly interpolate between the bottom 2 pixels. Then linearly interpolate between the calculated results of the previous two. You can expand on this concept to get trilinear interpolaton.

```
for n = 1:numel(t_matrix)

    % current coordinate
    [x, y] = ind2sub([rows cols], n);

    v = [x;y;1];
    v = IVT*v;

    a=v(1);
    b=v(2);

    a1 = floor(a);
    a2 = ceil(a);
    b1 = floor(b);
    b2 = ceil(b);

    if rows>a2 && a1 > 0 && cols>b2 && b1 > 0

    z11=double(input_matrix(a1,b1));
    z12=double(input_matrix(a1,b2));
    z21=double(input_matrix(a2,b1));
    z22=double(input_matrix(a2,b2));

    z1=z11+(b-b1)*(z12-z11)/(b2-b1);
    z2=z21+(b-b1)*(z22-z21)/(b2-b1);
    z=z1+(z2-z1)*(a-a1)/(a2-a1);

    t_matrix3(x,y) = uint8(z);

    end
end
```



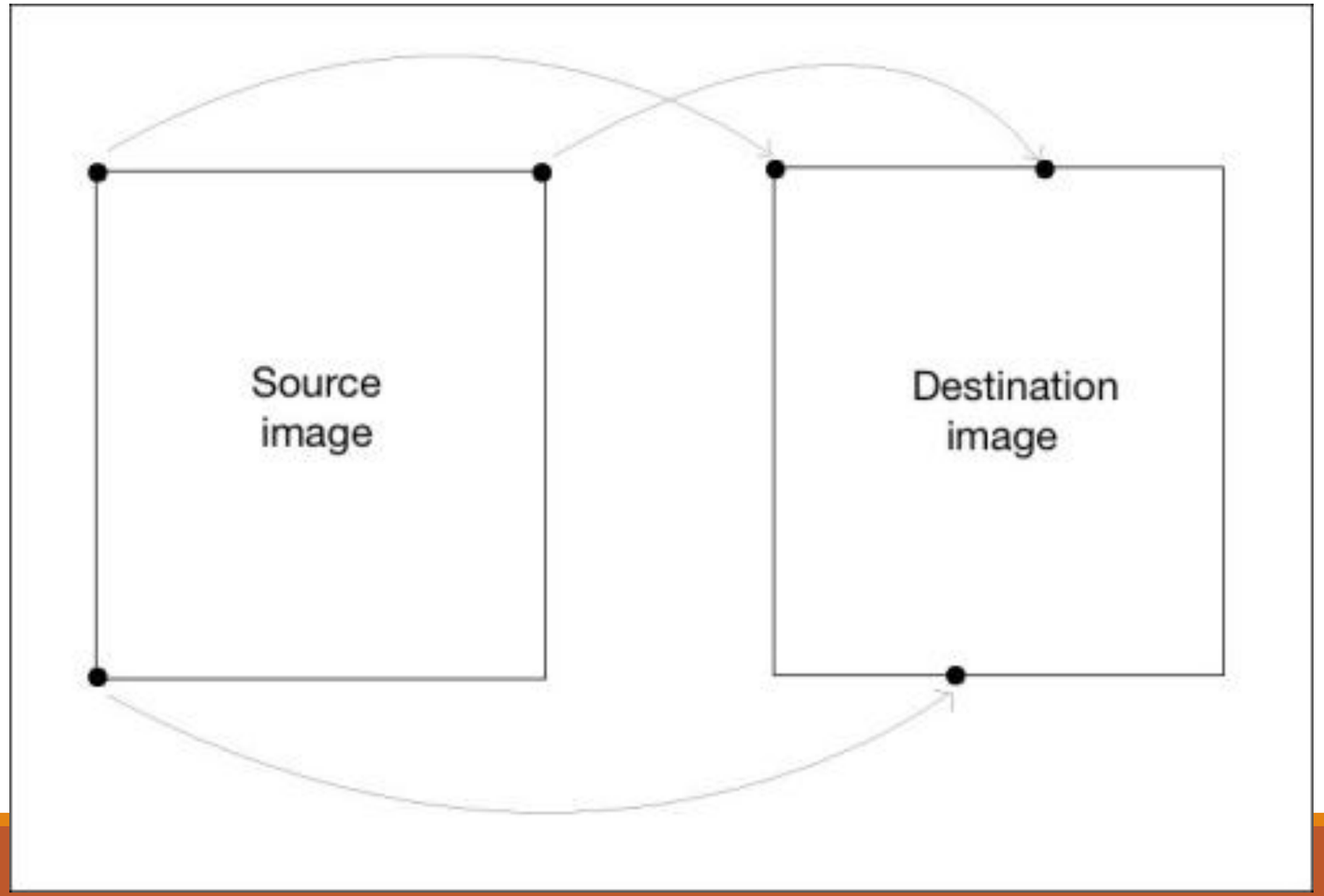
Stage 1 (original values)

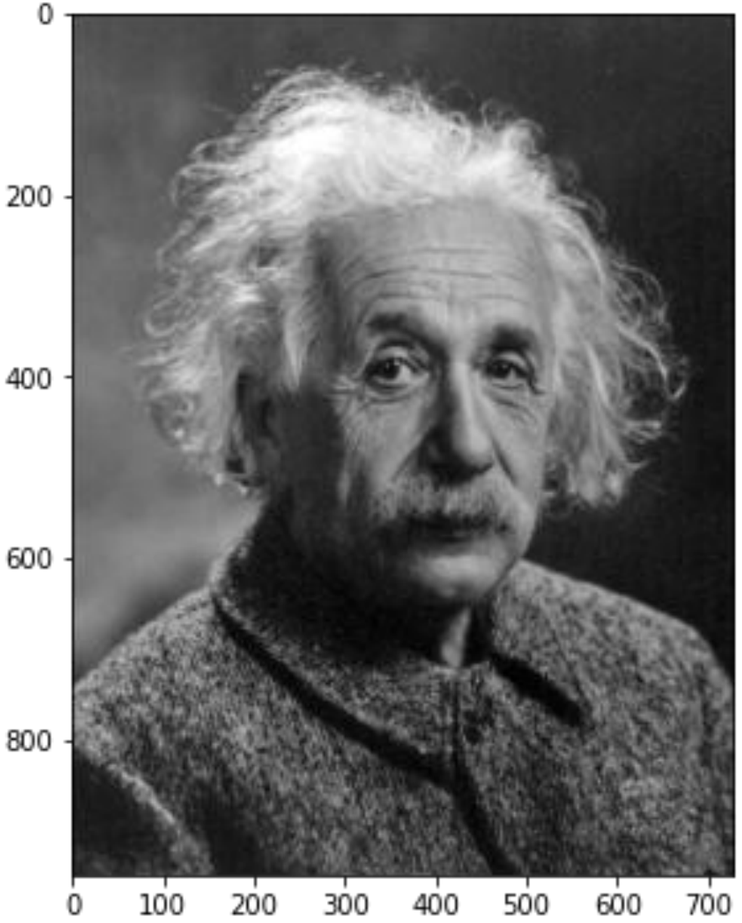
Stage 2 horizontal interpolation

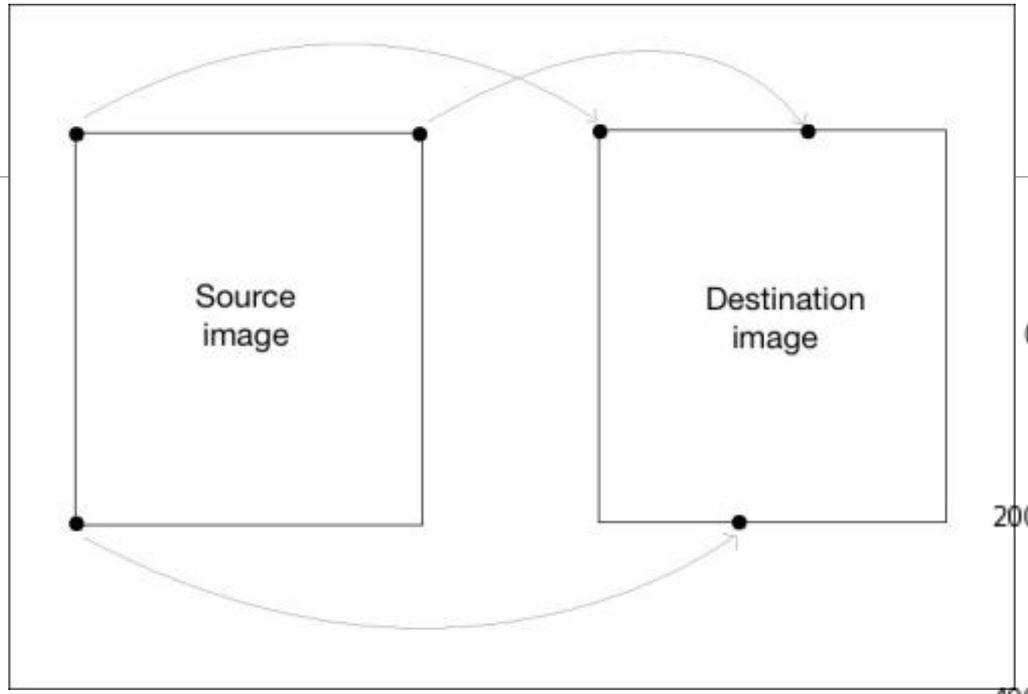
Stage 3 vertical interpolation (final)

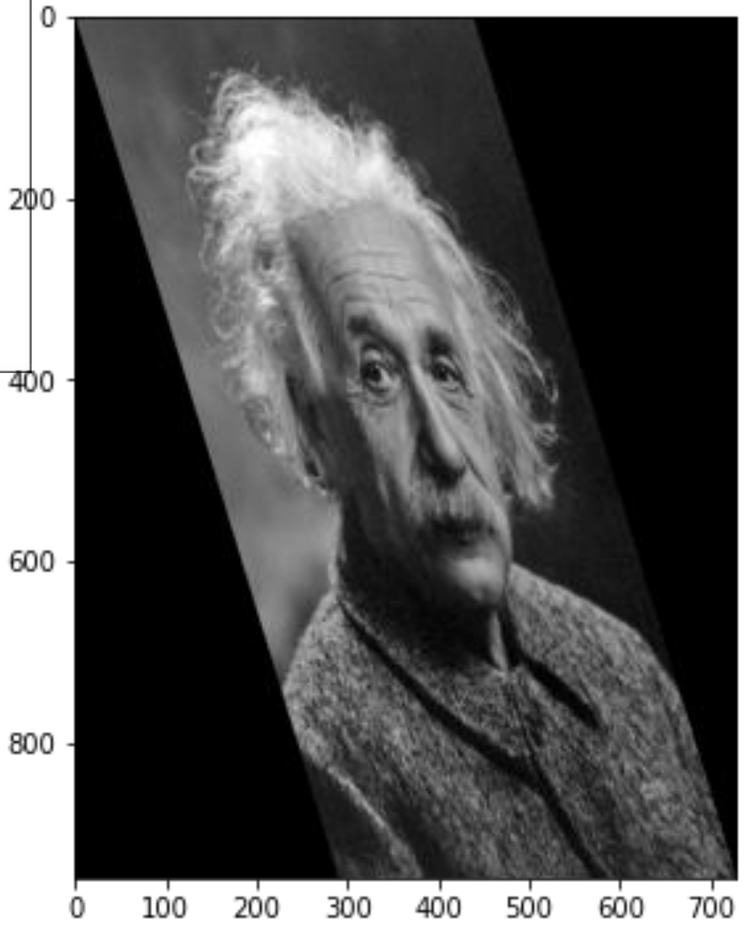
Using getAffineTransform()

Ask OpenCV to generate the compound transformation matrix for you by defining three reference points in the original image and their corresponding locations in the resultant image.



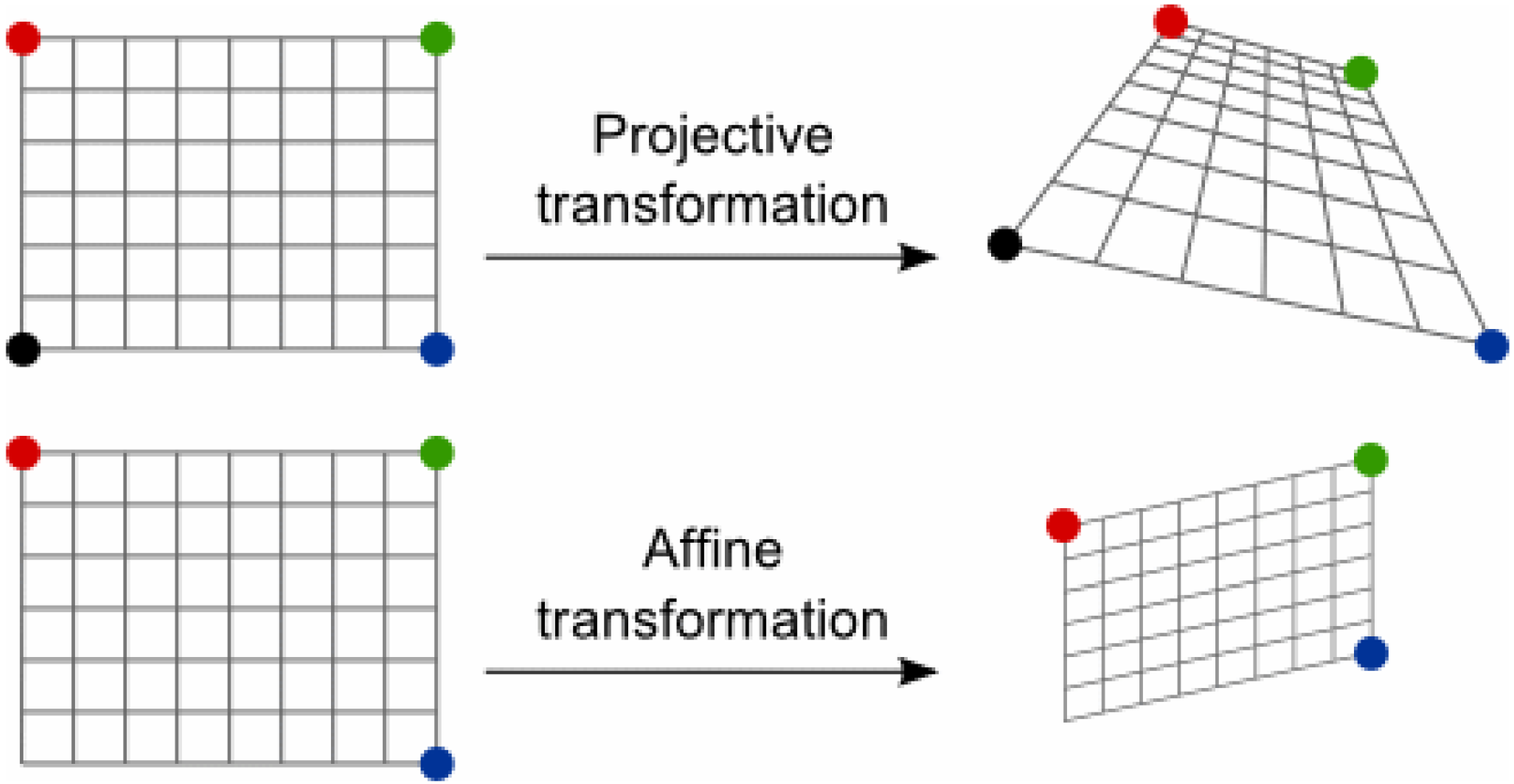

0
200
400
600
800
0
100
200
300
400
500
600
700

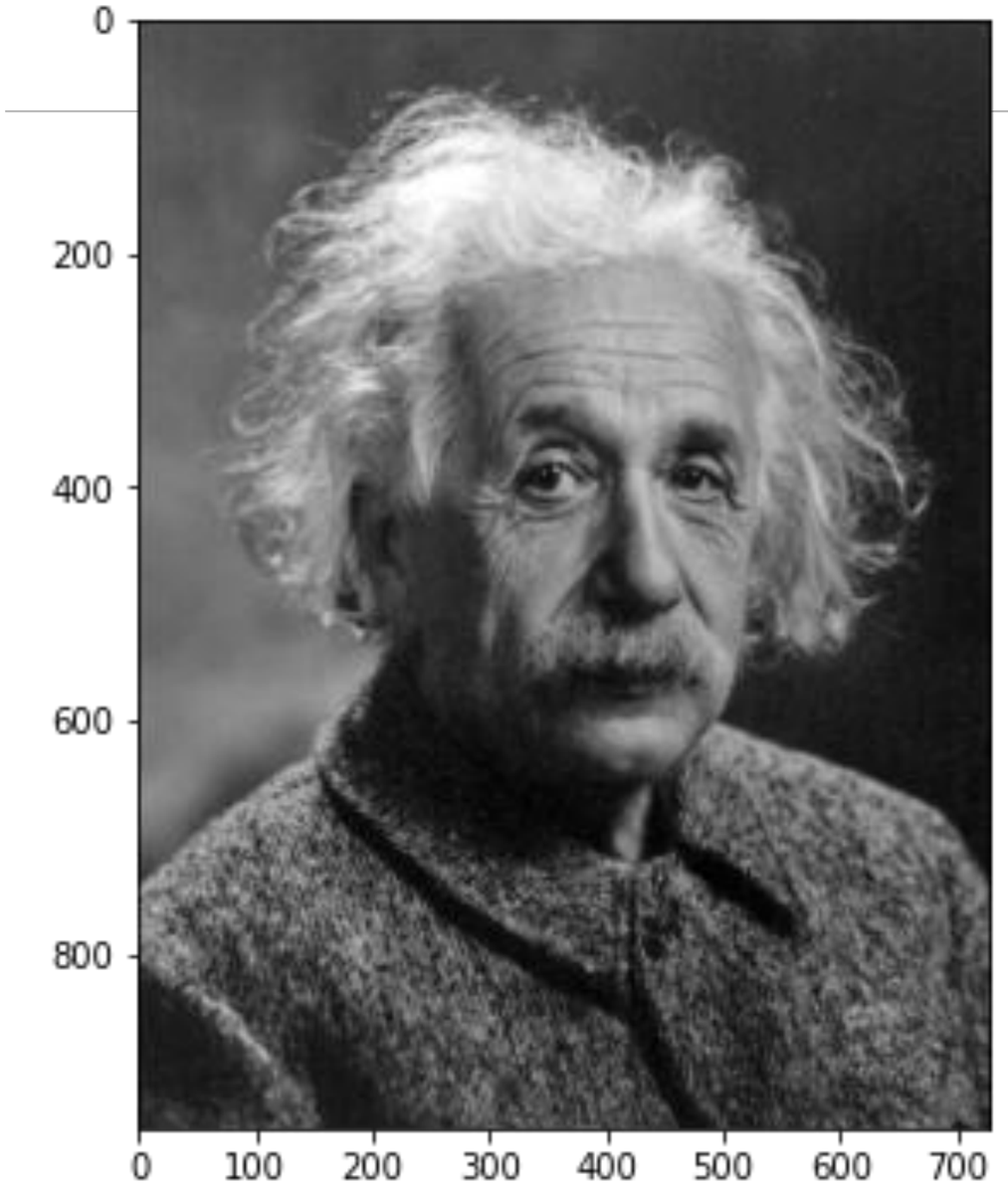

Source
image
Destination
image

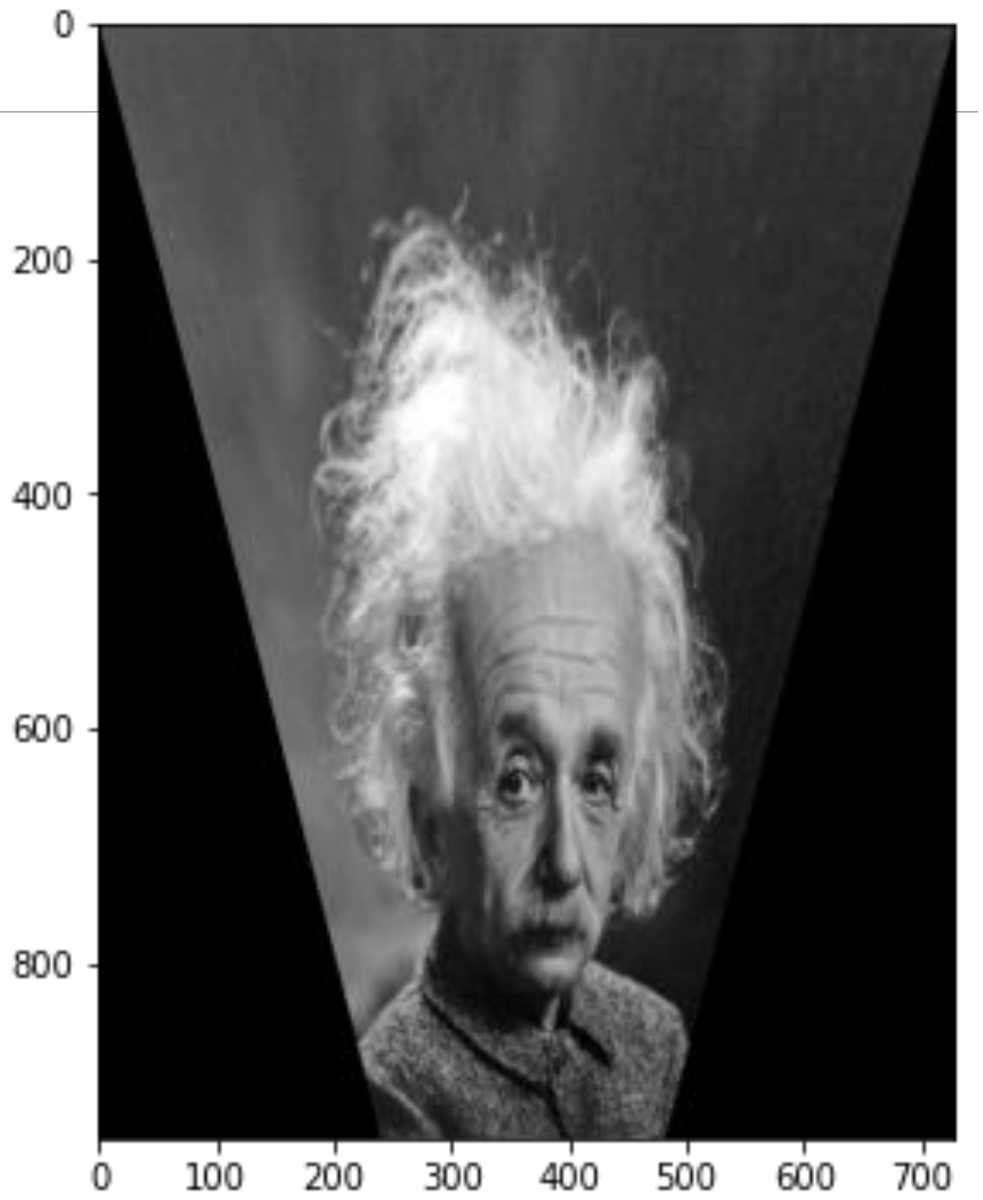

0
200
400
600
800
0
100
200
300
400
500
600
700

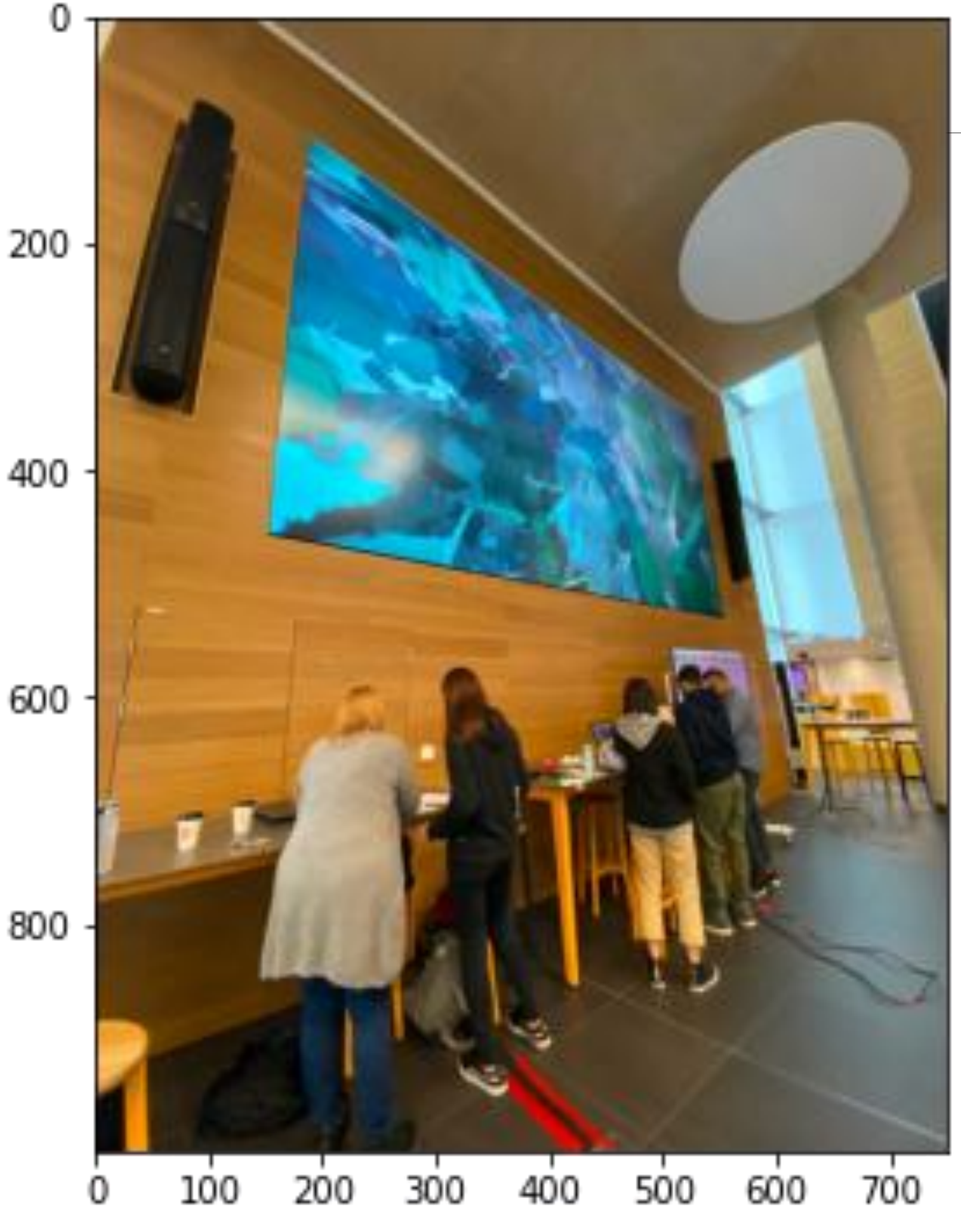
Projective transformations

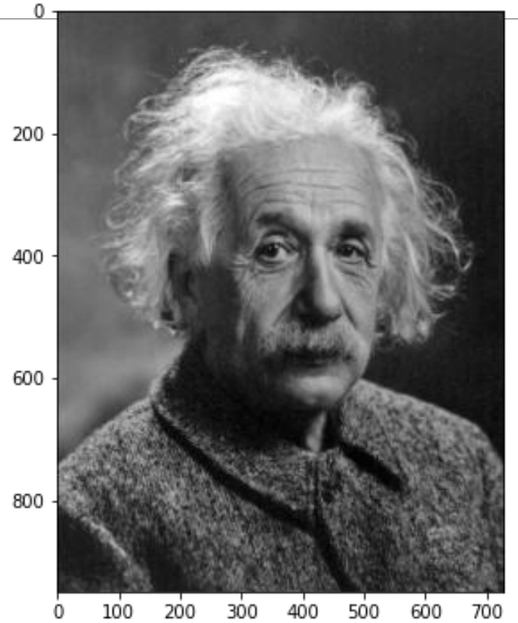
Four control points. More freedom.




0
200
400
600
800
0
100
200
300
400
500
600
700


0
200
400
600
800
0
100
200
300
400
500
600
700


0
200
400
600
800
0
100
200
300
400
500
600
700


0
200
400
600
800
0
100
200
300
400
500
600
700


0
200
400
600
800
0
100
200
300
400
500
600
700