Introduction to Computer Vision

5. 2D Image Transformation

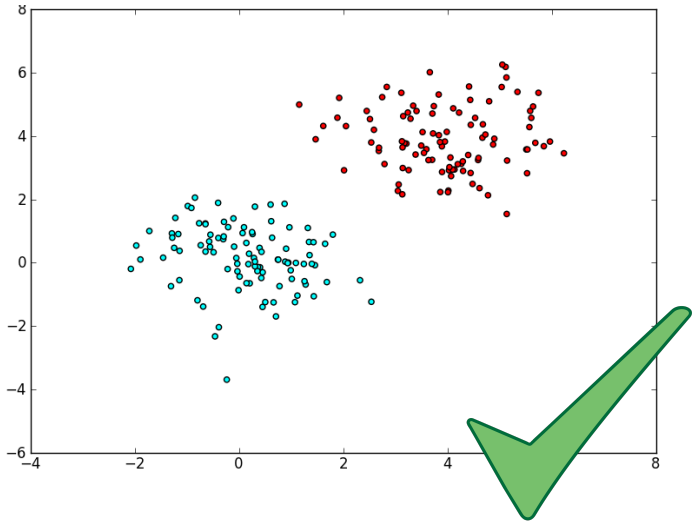
UCLA – CS 188 – Fall 2019

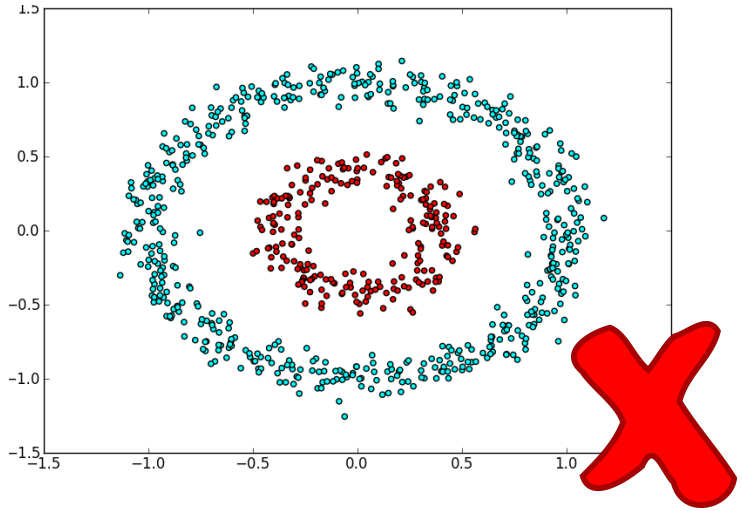
Fabien Scalzo, Ph.D.

Today

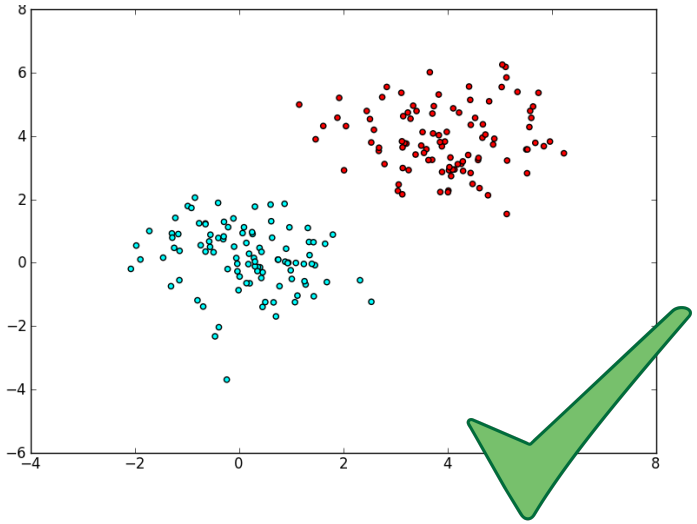
Week 1			26-Sep	Introduction
Week 2	1-Oct	Basic Image Processing	3-Oct	Feature Extraction and Classification
Week 3	8-Oct	Classification of Visual Features	10-Oct	**2D Image transformation**
Week 4	15-Oct	SVD, 2D camera model, projective plane	17-Oct	Euclidean geometry, rigid body motion
Week 5	22-Oct	Epipolar Geometry	24-Oct	3D Cameras and processing
Week 6	29-Oct	Midterm	31-Oct	Statistical decision theory/Pattern Recognition
Week 7	5-Nov	Deep Learning	7-Nov	Deep Learning for Image Classification
Week 8	12-Nov	Object Detection	14-Nov	Generative models
Week 9	19-Nov	Medical Imaging	21-Nov	Autonomous Navigation
Week 10	26-Nov	Guest Lecture (Nikhil Naik)	28-Nov	
Week 11	3-Dec	Recursive 3D reconstruction and pose estimation	5-Dec	Recap
Week 12	10-Dec		12-Dec	Final

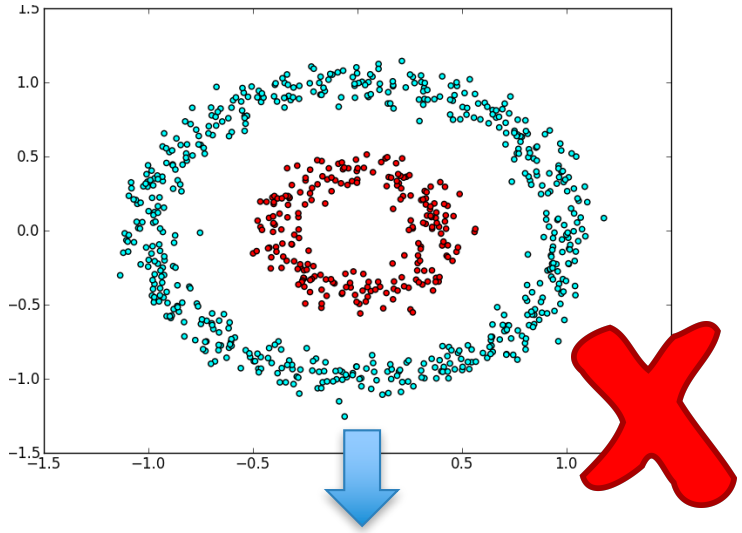
Linear Classifier



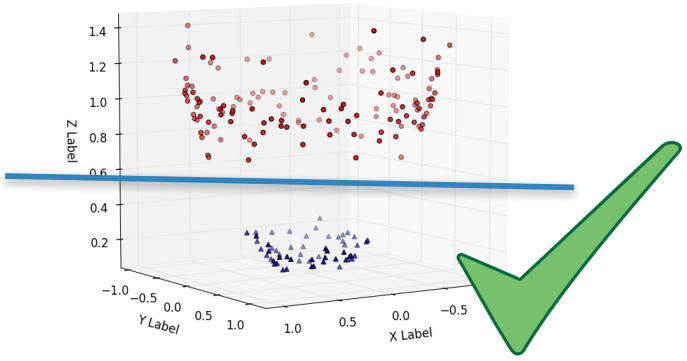


Kernel Projection + Linear Classifier





Data in R^3 (separable)



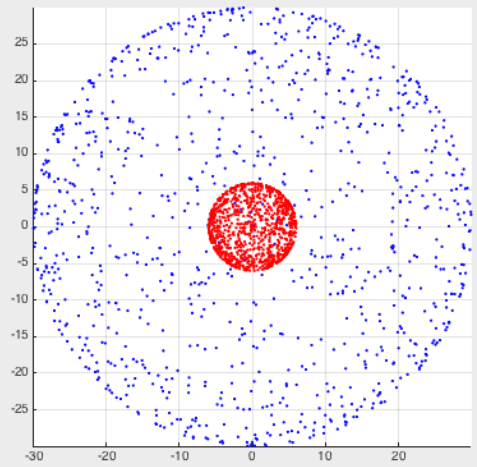
$$[x_1, x_2] = [x_1, x_2, {x_1}^2 + {x_2}^2]$$

Kernel Projection + Linear Classifier

Radial Basis Function (RBF) Kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

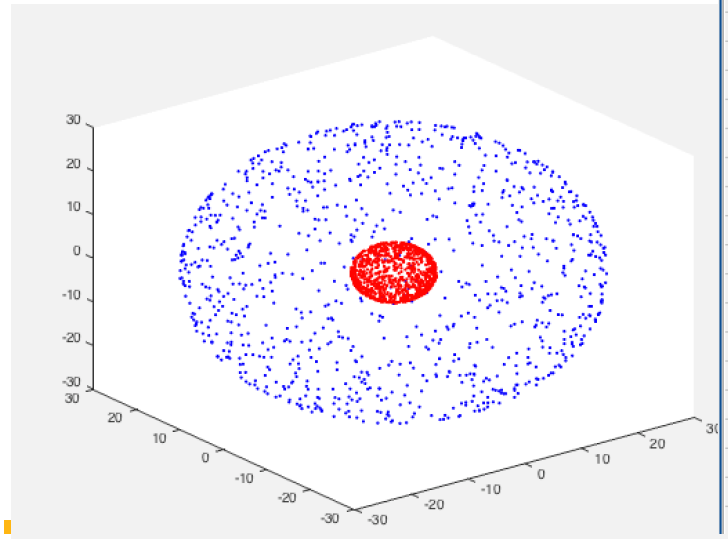
Trick: represents each datapoint as a vector of weights to all the other datapoints



Kernel Projection + Linear Classifier

Represents each datapoint as a vector of weights to all the datapoints

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$



datapoints

2000x3 double

	1	2	3
1	-0.1503	-4.6598	3.7767
2	2.7325	-2.1957	4.8695
3	0.1856	3.9912	-4.4762
4	2.3168	2.4547	4.9605
5	0.9027	5.7151	1.5883
6	-2.0929	2.8802	-4.8295
7	-1.2421	5.2338	-2.6580
8	5.3642	-2.6284	0.5626
9	2.2958	0.7670	5.4901
10	-1.8448	-1.2146	5.5787
11	2.2753	3.7339	-4.1086
12	1.0691	-1.7225	5.6471
13	1.2030	2.1110	5.4860
14	-5.9969	-0.0829	-0.1755
15	4.7974	-0.0202	3.6034
16	-2.5748	3.3020	-4.2974
17	5.6687	1.7275	-0.9389
18	0.7545	3.2469	4.9888
19	-3.8997	2.9150	3.5065
20	-1.1872	2.0463	5.5139

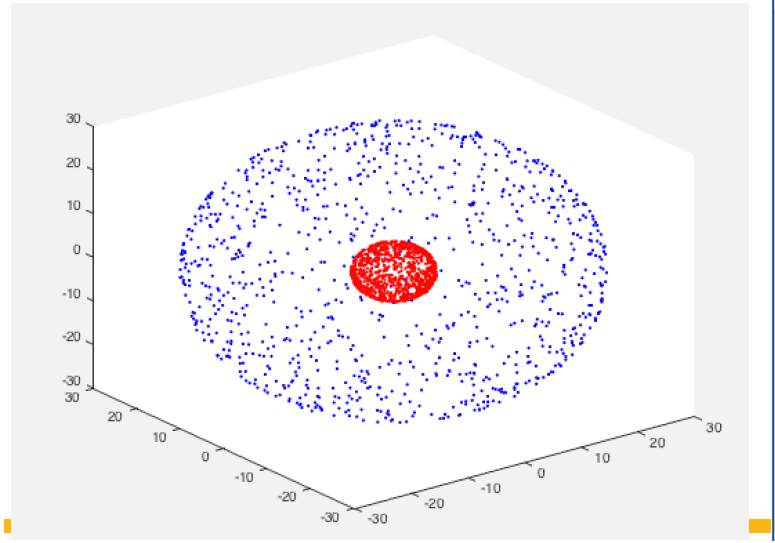
Class 1

Class 2

Kernel Projection + Linear Classifier

Represents each datapoint as a vector of weights to all the datapoints

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$



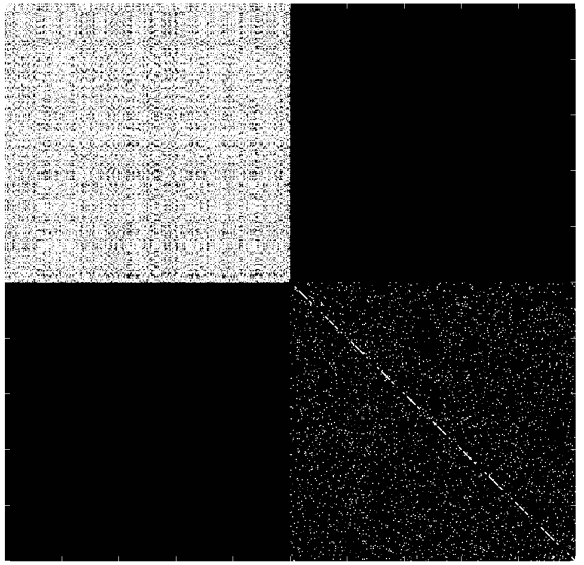
datapoints

2000x3 double

	1	2	3
1	-0.1503	-4.6598	3.7767
2	2.7325	-2.1957	4.8695
3	0.1856	3.9912	-4.4762
4	2.3168	2.4547	4.9605
5	0.9027	5.7151	1.5883
6	-2.0929	2.8802	-4.8295
7	-1.2421	5.2338	-2.6580
8	5.3642	-2.6284	0.5626
9	2.2958	0.7670	5.4901
10	-1.8448	-1.2146	5.5787
11	2.2753	3.7339	-4.1086
12	1.0691	-1.7225	5.6471
13	1.2030	2.1110	5.4860
14	-5.9969	-0.0829	-0.1755
15	4.7974	-0.0202	3.6034
16	-2.5748	3.3020	-4.2974
17	5.6687	1.7275	-0.9389
18	0.7545	3.2469	4.9888
19	-3.8997	2.9150	3.5065
20	-1.1872	2.0463	5.5139

Class 1

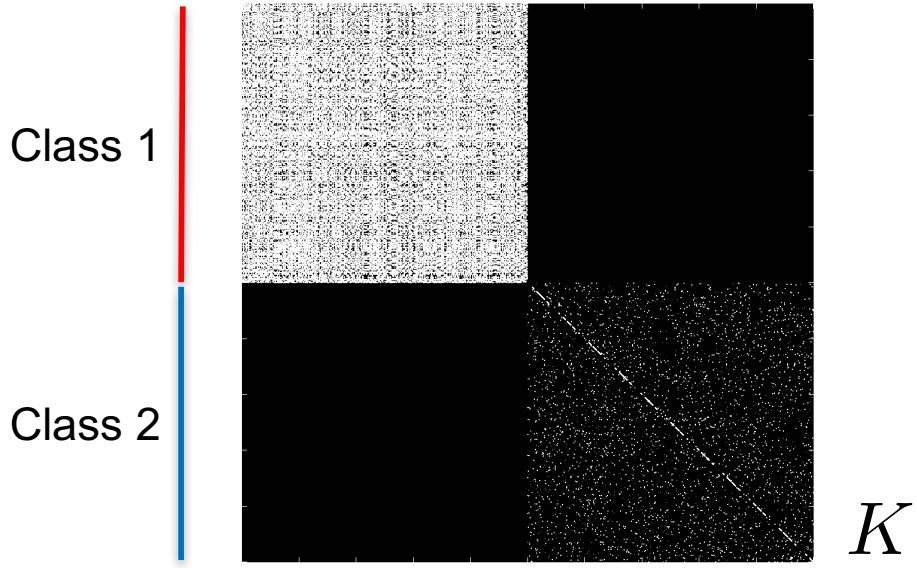
Class 2



K

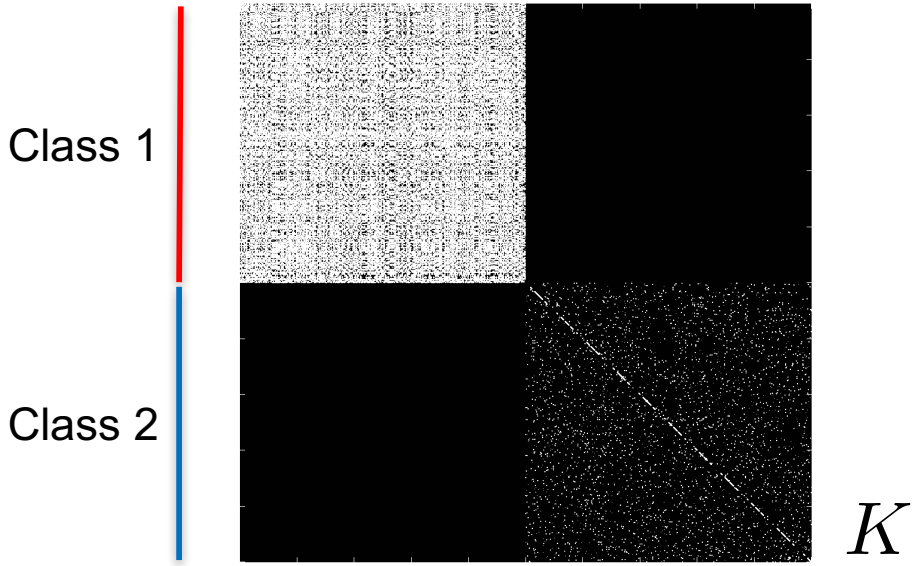
Kernel Projection + Linear Classifier

$$\underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \mathbf{w}^\mathsf{T} K(\mathbf{x}_i, \mathbf{x}))$$



Kernel Projection + Linear Classifier

$$\underset{\mathbf{w}}{\operatorname{argmin}}\ R(\mathbf{w}) + C\sum_{i=1}^{N} L(y_i, \mathbf{w}^{\mathsf{T}} K(\mathbf{x}_i, \mathbf{x}))$$









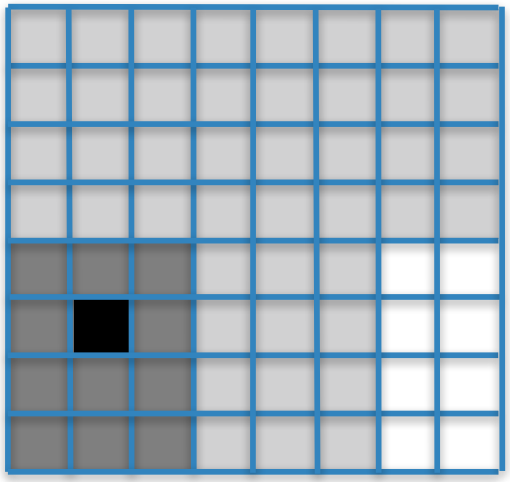




How?

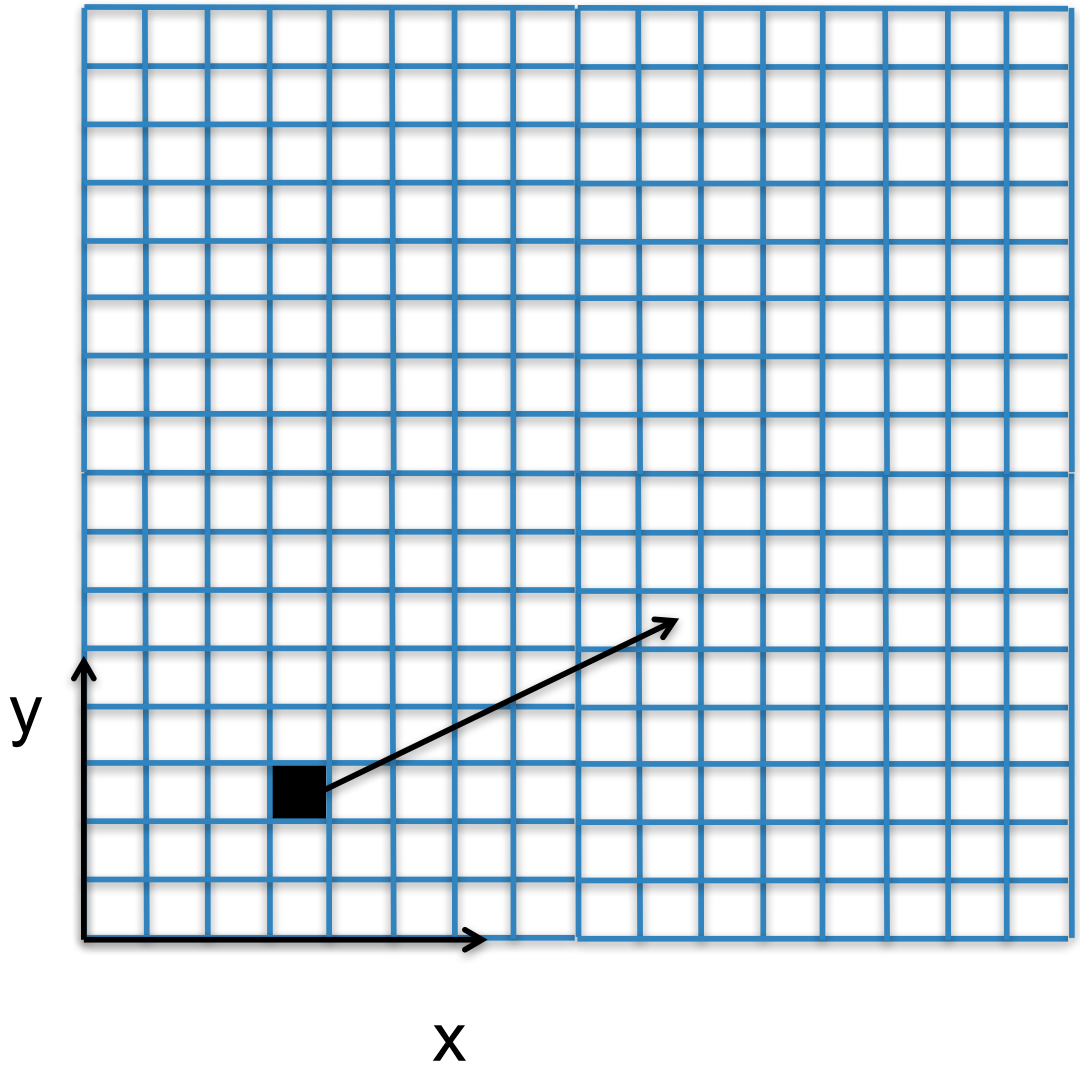
- Rotation, Scaling

Image representation



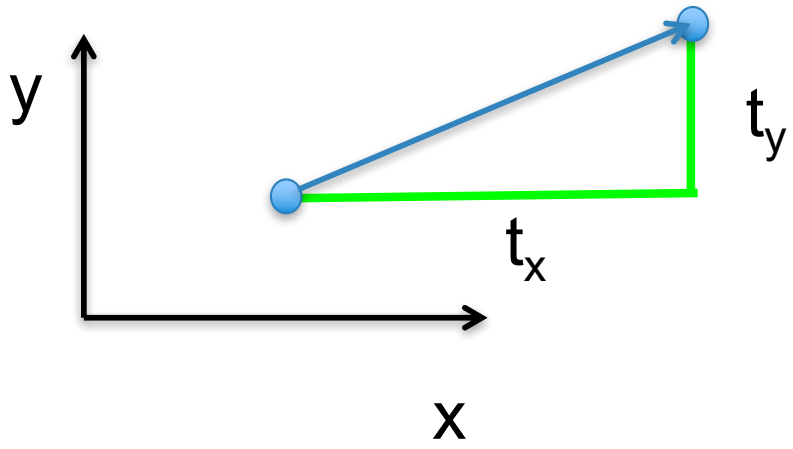
Each pixel is described by a position (x,y) in the image and an intensity value I(x,y)

Translation



Translation

t_x: shift in x
t_y: shift in y

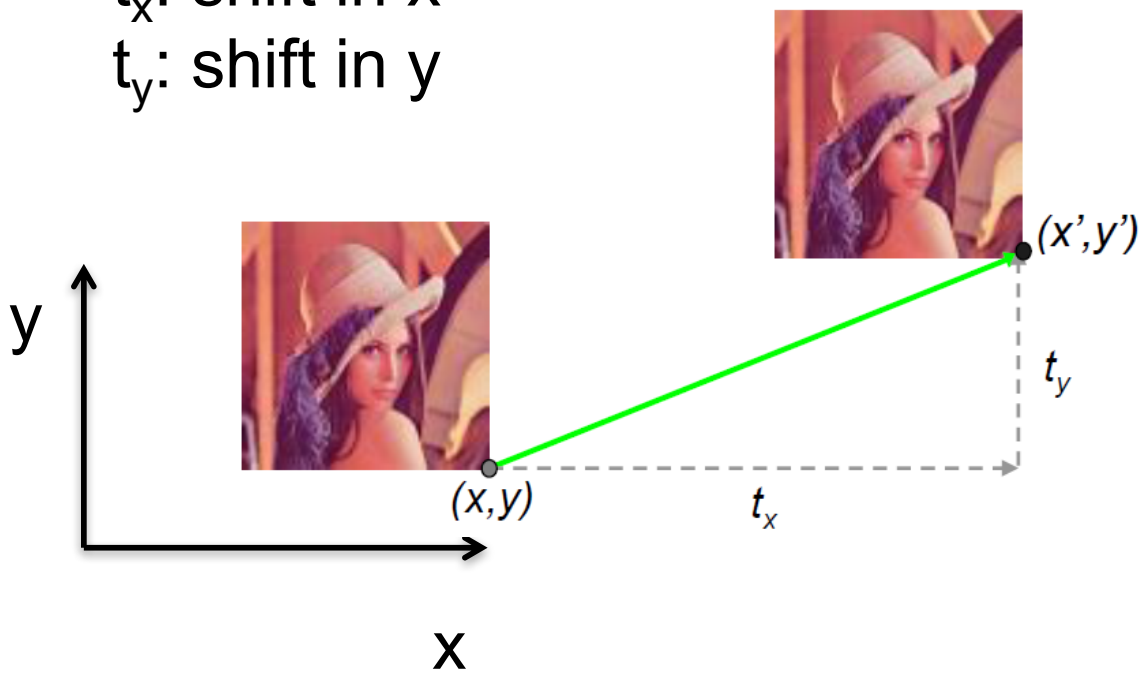


Translation

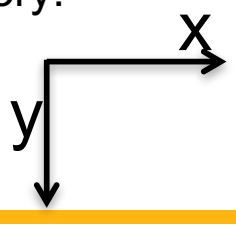
Translation

$$x' = x + t_x \qquad y' = y + t_y$$

t_x: shift in x
t_y: shift in y



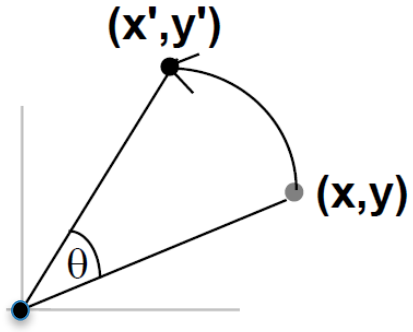
Array in memory:



Rotation

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = y\cos(\theta) + x\sin(\theta)$$

θ is the counter-clockwise rotation angle



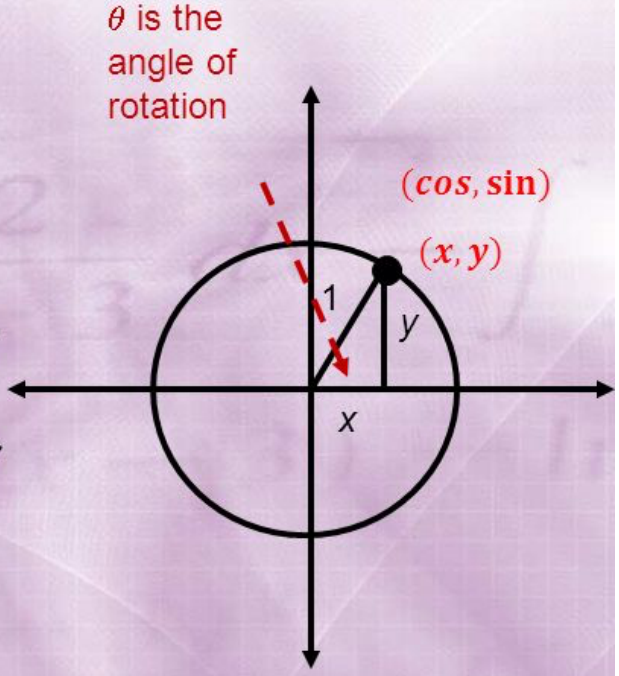
The Unit Circle

- The coordinates of the chosen point are the cosine and sine of the angle θ.

$$\cos\theta = x$$

$$\sin\theta = y$$

  - This provides a way to define functions $\sin(\theta)$ and $\cos(\theta)$ for all real numbers θ.
  - The other trigonometric functions can be defined from these.

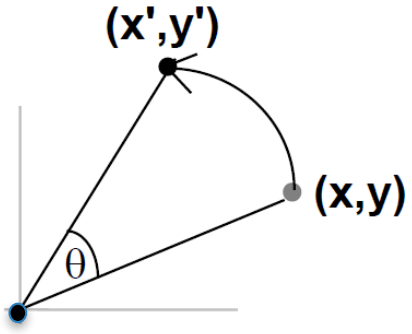


Rotation

Rotation

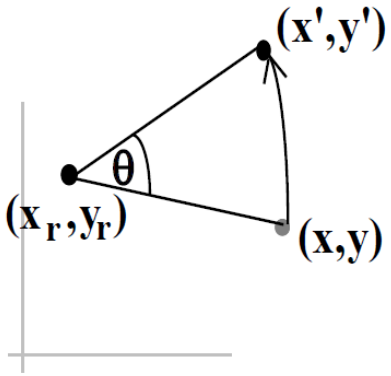
$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = y\cos(\theta) + x\sin(\theta)$$

θ is the counter-clockwise rotation angle



Rotation about a point

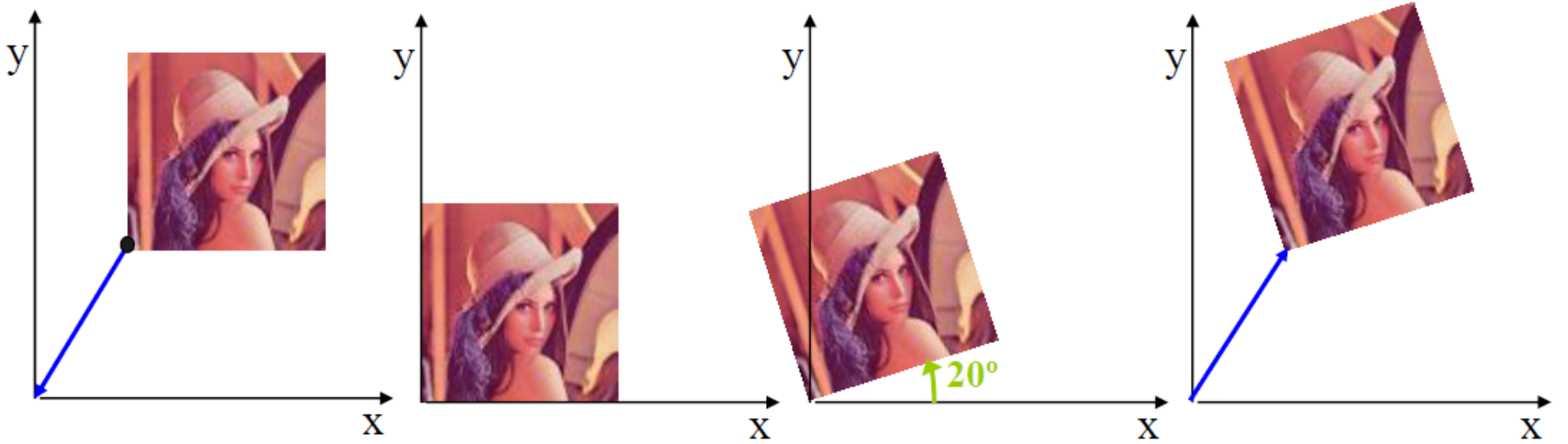
$$x' = x_r + (x - x_r)\cos(\theta) - (y - y_r)\sin(\theta)$$
$$y' = y_r + (y - y_r)\cos(\theta) + (x - x_r)\sin(\theta)$$



Example: rotation about a point

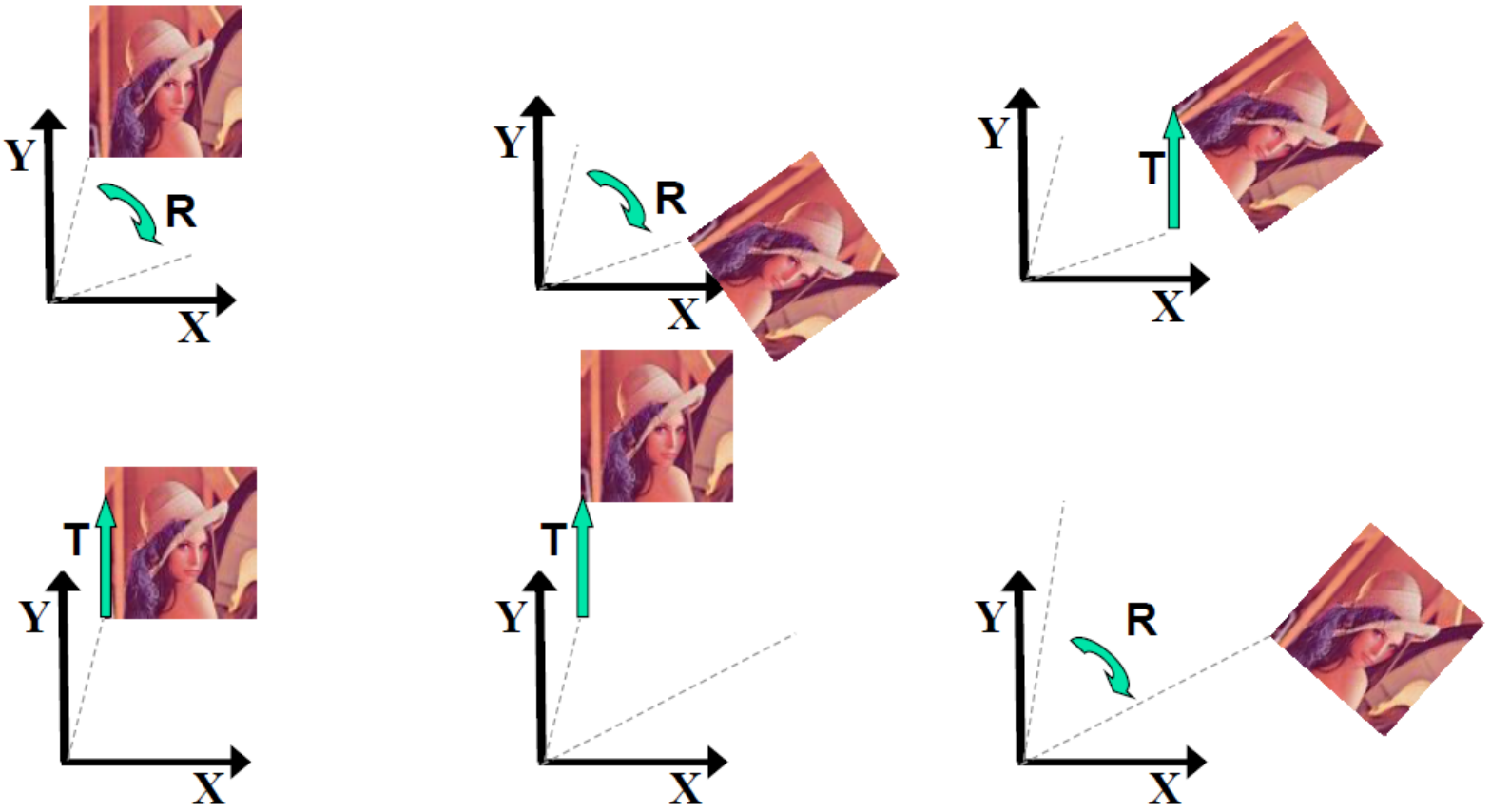
$$x' = x_r + (x - x_r)\cos(\theta) - (y - y_r)\sin(\theta)$$

$$y' = y_r + (y - y_r)\cos(\theta) + (x - x_r)\sin(\theta)$$

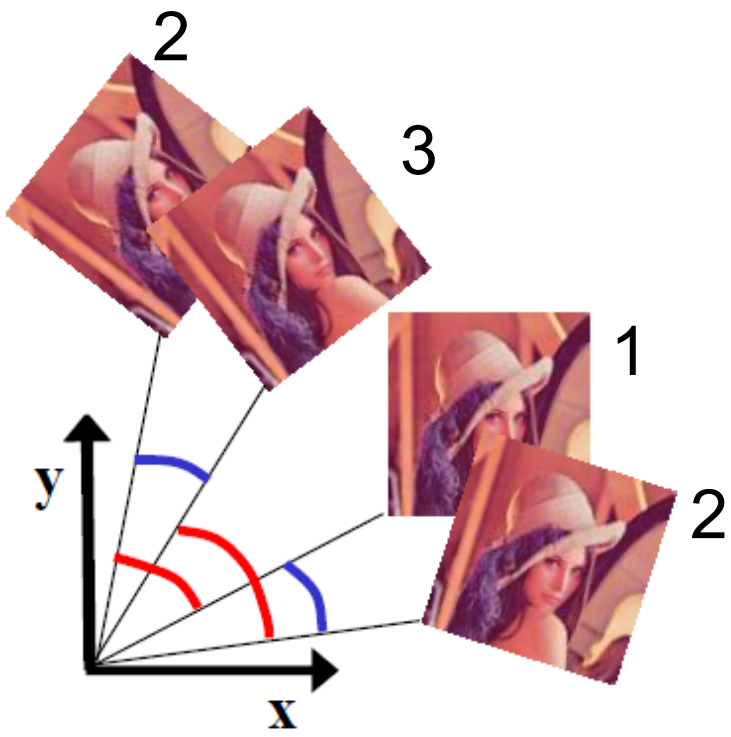


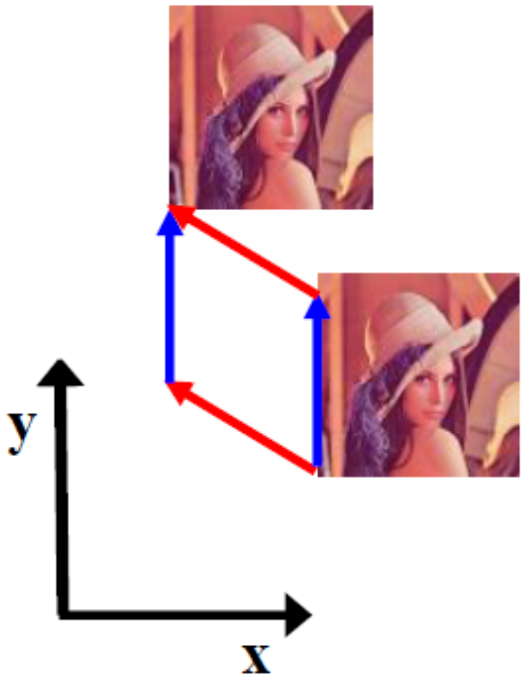
translation rotation translation

Rotation x Translation ≠ Translation x Rotation



Order

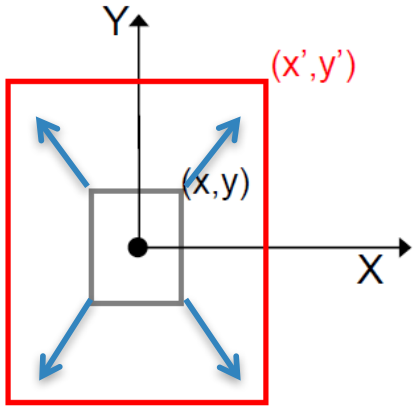




Scaling (w.r.t. the origin)

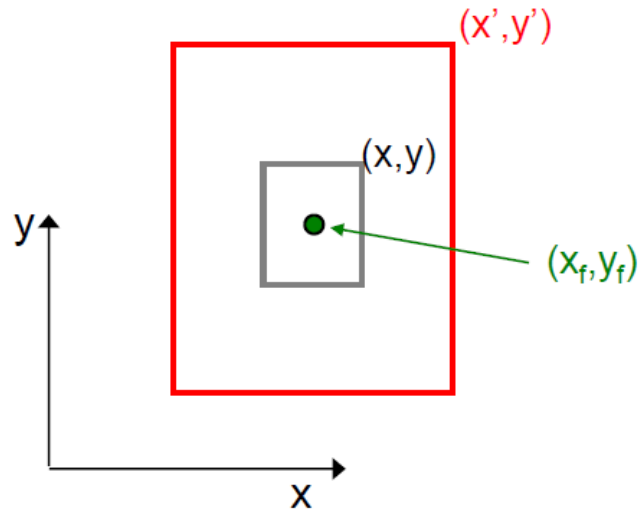
$$x' = s_x \cdot x \qquad y' = s_y \cdot y$$

s_x = scaling factor in x
s_y = scaling factor in y

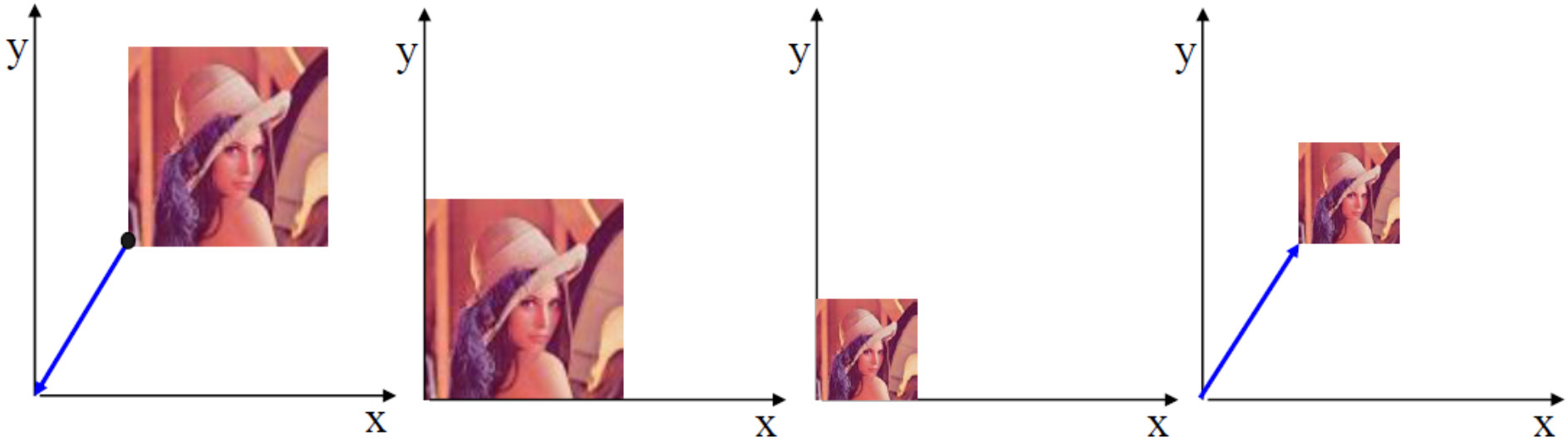


Scaling (about a point)

$$x' = x_f + (x - x_f) \cdot s_x \qquad y' = y_f + (y - y_f) \cdot s_y$$



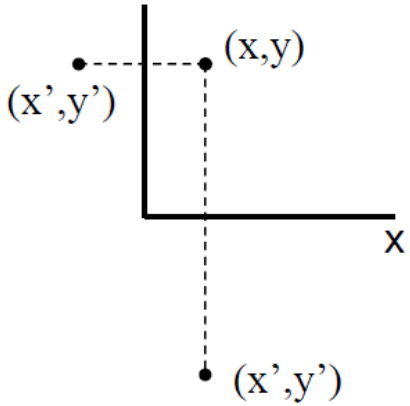
Example



translation scaling translation

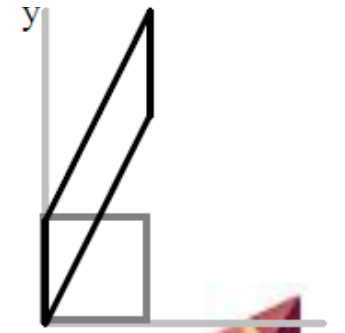
Flip

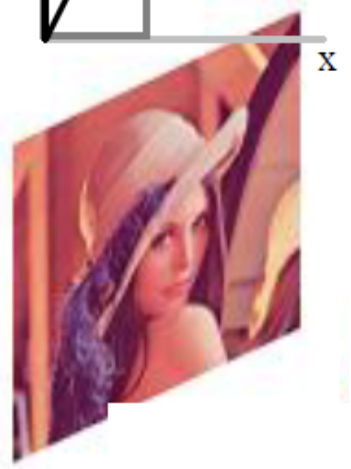
$$\begin{cases} x' = x \\ y' = -y \end{cases} \quad \text{et} \quad \begin{cases} x' = -x \\ y' = y \end{cases}$$



Shearing

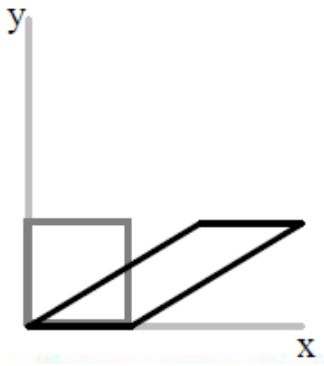
$$\begin{bmatrix} x' = x \\ y' = y + sh_y \cdot x \end{bmatrix}$$

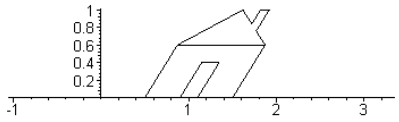




along Y-axis

$$\begin{bmatrix} x' = x + sh_x \cdot y \\ y' = y \end{bmatrix}$$





along X-axis

Inverse transforms

- Translation: $t_x, t_y \rightarrow -t_x, -t_y$
- Rotation: $\theta \rightarrow -\theta$
- Scaling: $s_x, s_y \rightarrow 1/s_x, 1/s_y$
- Shearing: $sh_x, sh_y \rightarrow -sh_x, -sh_y$

Matrix representation

- Rotation

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

- Scaling

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad x' = s_x \cdot x \qquad y' = s_y \cdot y$$

- Shearing

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{aligned} x' &= x + sh_x \cdot y \\ y' &= y \end{aligned}$$

- Translation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- For convenience, we use **Homogeneous Coordinates**
  - For combinations of transforms: easier to read

Translation:

$$x' = 1 \cdot x + 0 \cdot y + t_x \cdot 1$$

$$x' = x + t_x$$

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

T

Homogeneous Coordinates

Scaling:

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

S

Rotation:

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

R

Homogeneous Coordinates

$$M_{ST} = S(s_x, s_y) \cdot T(t_x, t_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & s_x \cdot t_x \\ 0 & s_y & s_y \cdot t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Homogeneous Coordinates

$$M_{ST} = S(s_x, s_y) \cdot T(t_x, t_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & s_x \cdot t_x \\ 0 & s_y & s_y \cdot t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\neq \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = T(t_x, t_y) \cdot S(s_x, s_y) = M_{TS}$$

Example: scaling

$$x' = x_f + (x - x_f) \cdot s_x \qquad y' = y_f + (y - y_f) \cdot s_y$$

$T(-x_f, -y_f)$ $S(s_x, s_x)$ $T(x_f, y_f)$

$$= \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix}$$

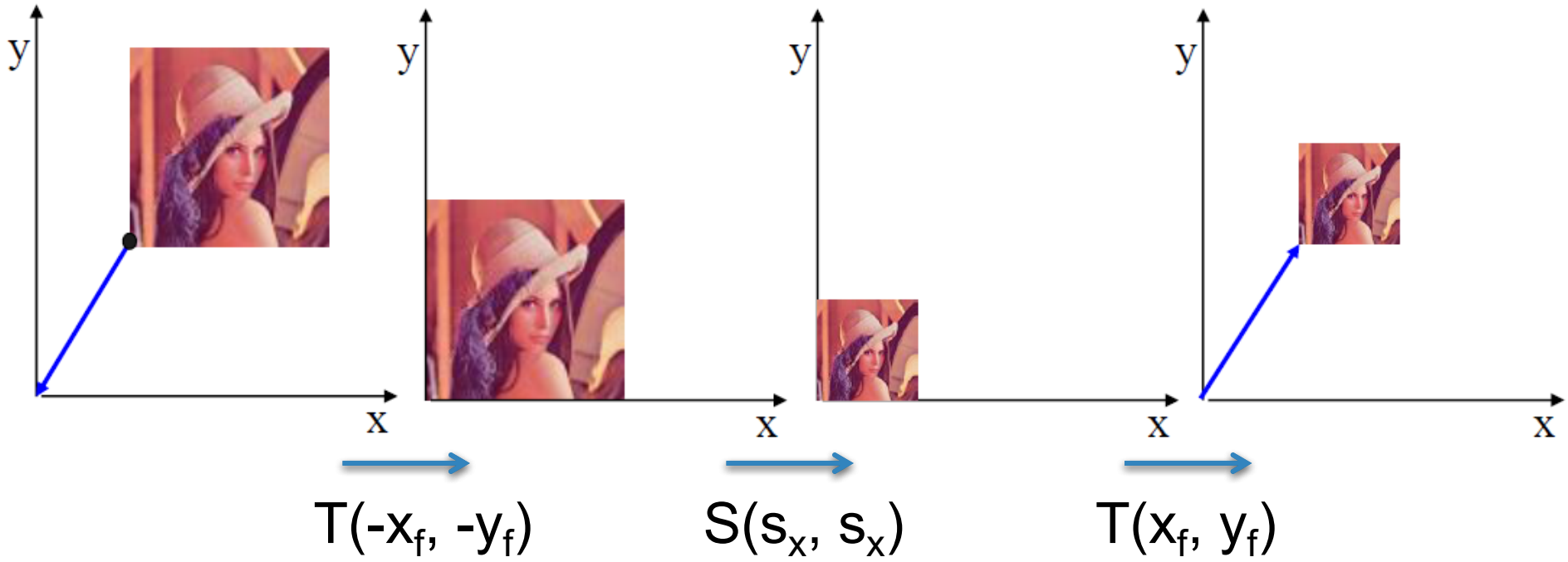
When we write transformations using standard math notation, the closest transformation to the point is applied first:

$$T\ R\ S\ p = T(R(Sp))$$

first, the object is scaled,
then rotated,
then translated

Example: scaling

$$x' = x_f + (x - x_f) \cdot s_x \qquad y' = y_f + (y - y_f) \cdot s_y$$



$$= \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_f - s_x \cdot x_f \\ 0 & s_y & y_f - s_y \cdot y_f \\ 0 & 0 & 1 \end{bmatrix}$$

Homogeneous Coordinates

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = M_{TST^{-1}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_f - s_x \cdot x_f \\ 0 & s_y & y_f - s_y \cdot y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Leftrightarrow$$

$$x' = x_f + (x - x_f) \cdot s_x \qquad y' = y_f + (y - y_f) \cdot s_y$$

Summary

Translation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Scaling

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Rotation

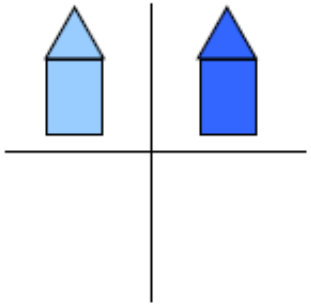
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Shearing

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

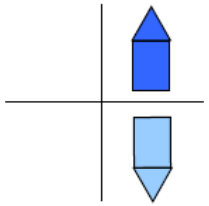
Other transformations

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

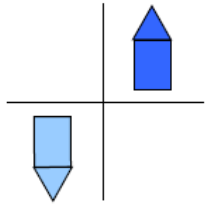


Flip along x

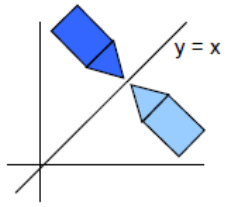
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



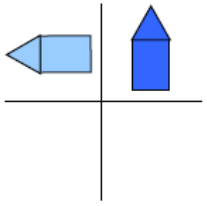
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

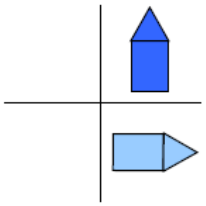


$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



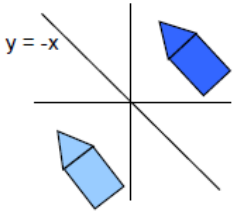
Rotation 90°

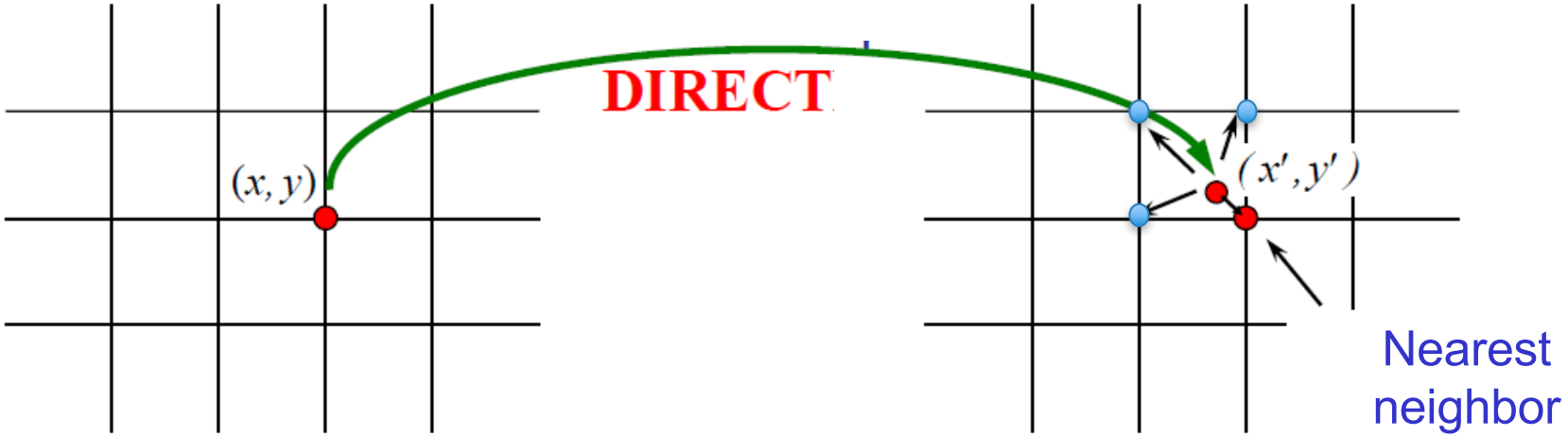
$$\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



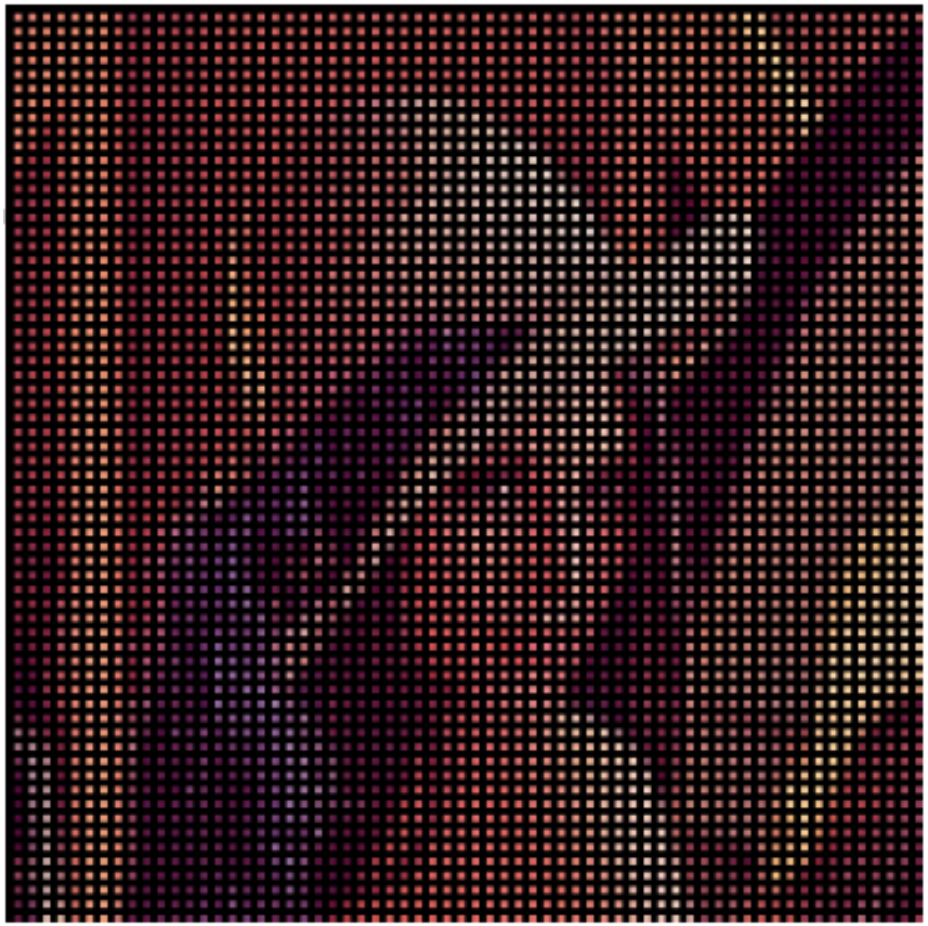
Rotation -90°

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

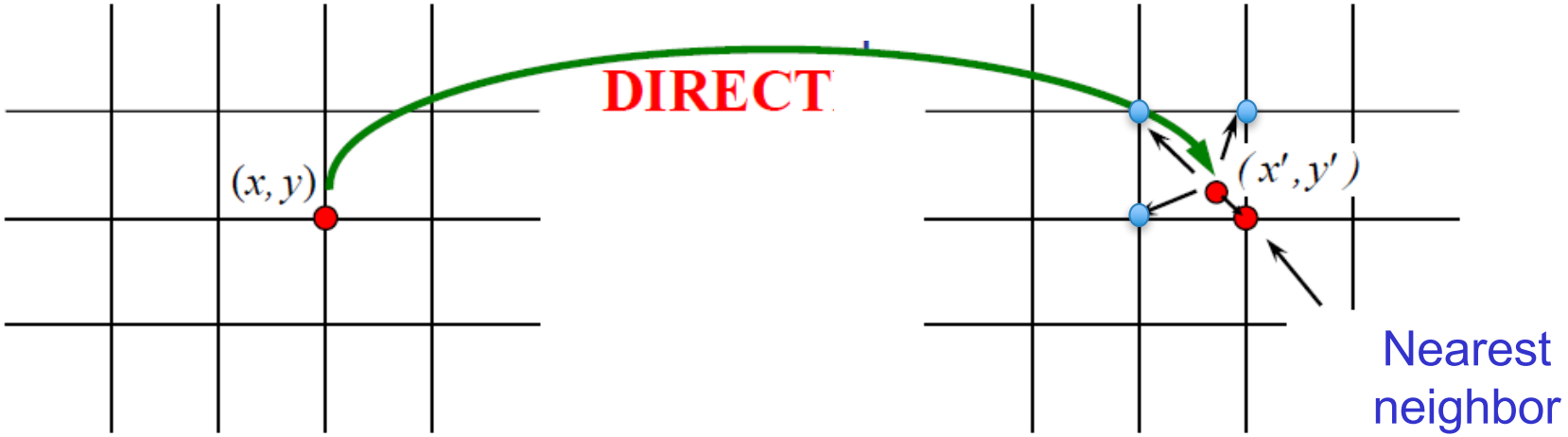



DIRECT
(x, y)
(x′, y′)
Nearest
neighbor

Direct

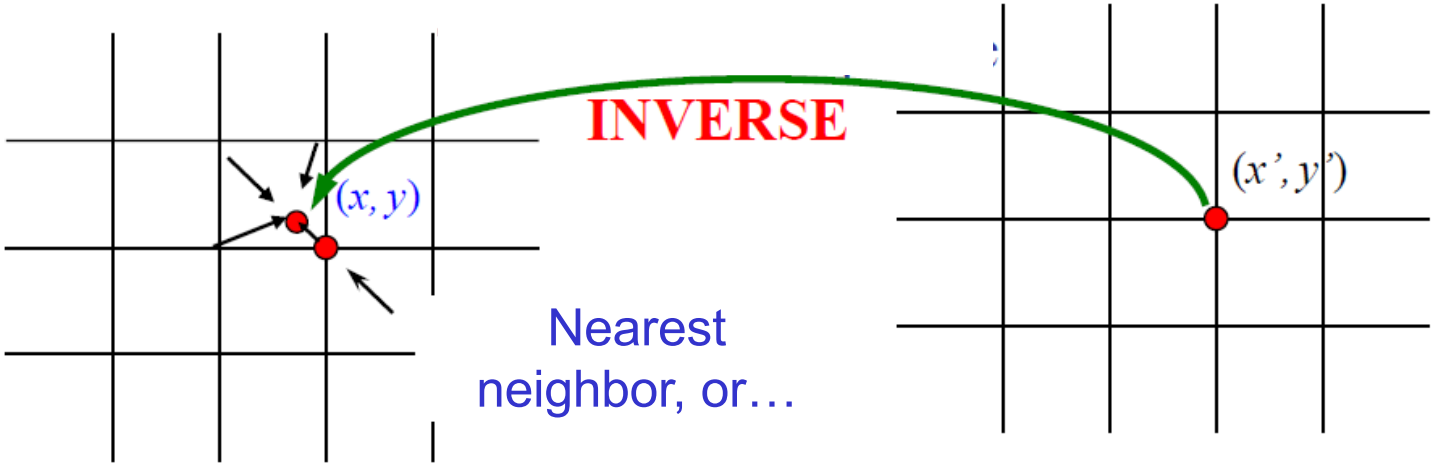


How not to do it



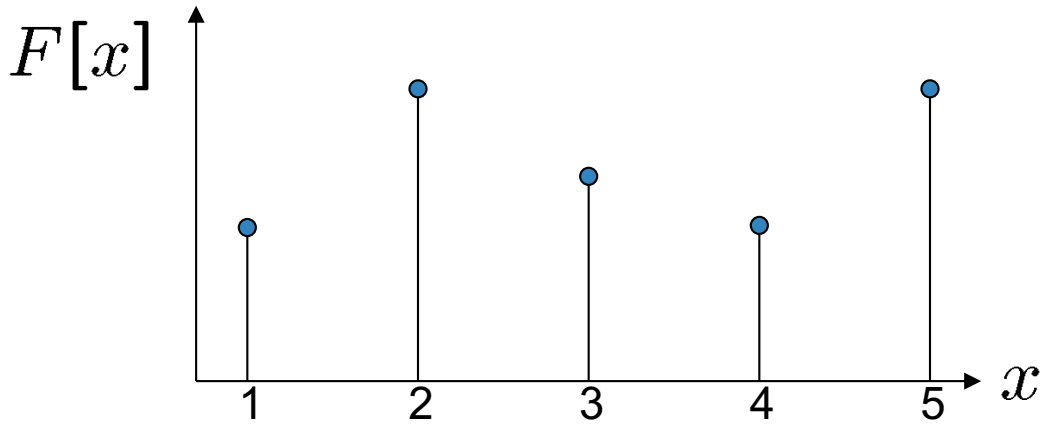
- leads to holes in the image, multiple values for some pixels

How to do it



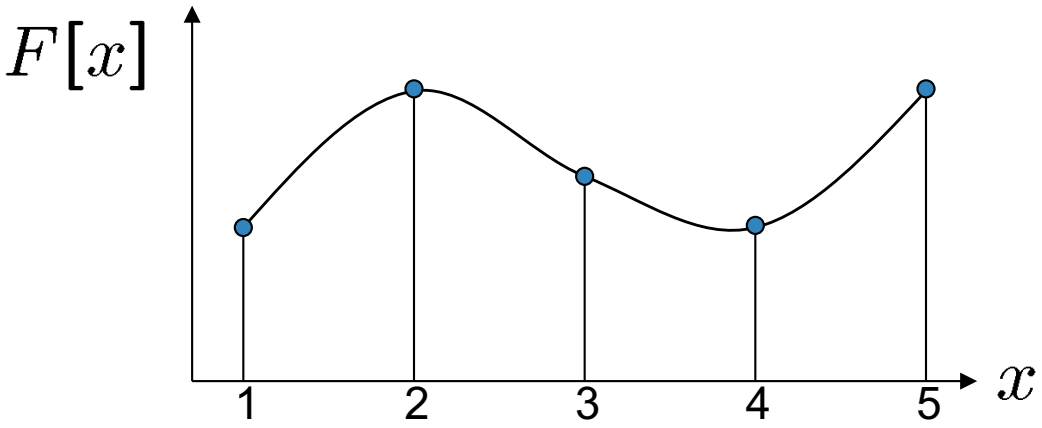
Inverse

Image interpolation



- What if we don't know F?
  - It is a discrete point-sampling of a continuous function
  - If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

Image interpolation



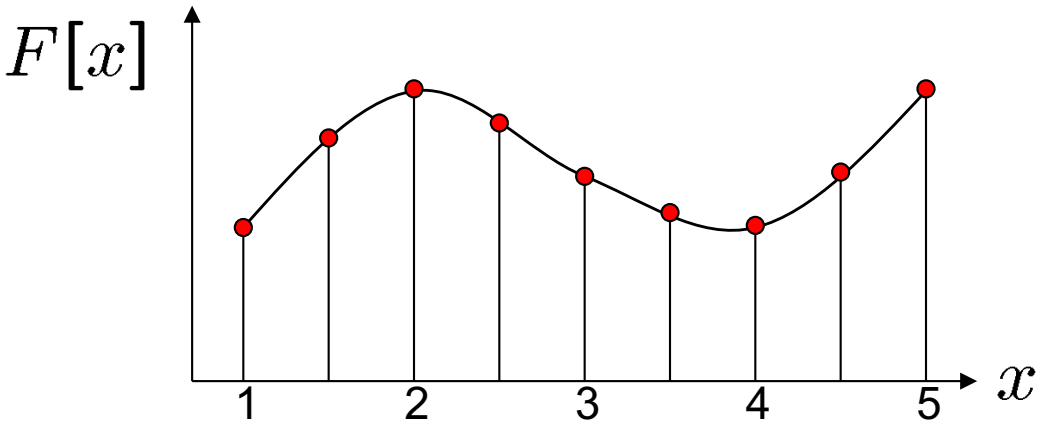

$F[x]$
1
2
3
4
5
x

Image interpolation

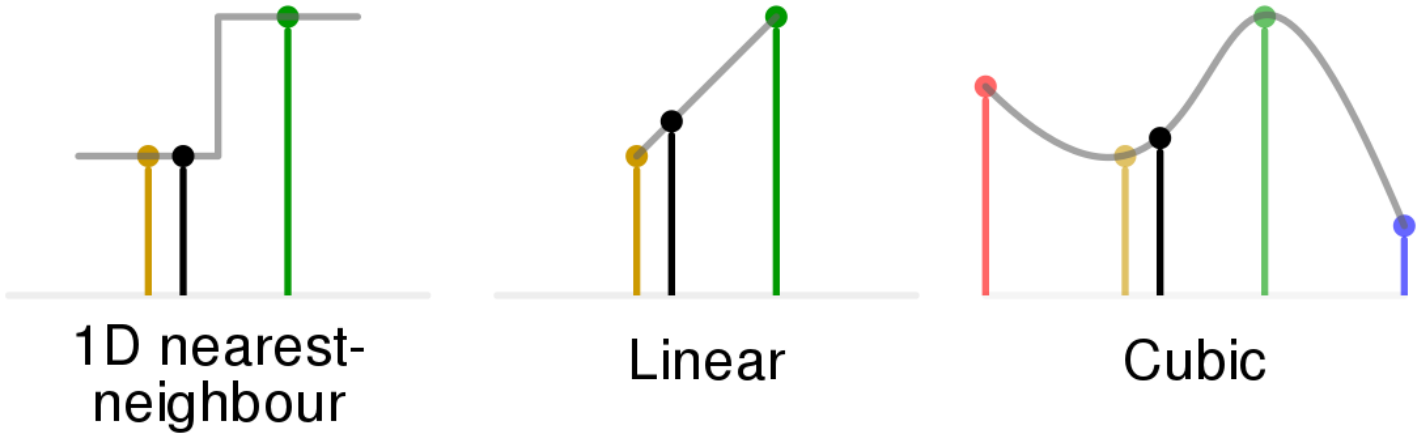
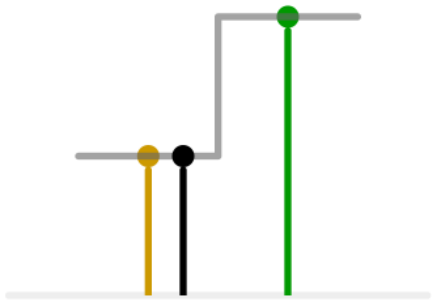
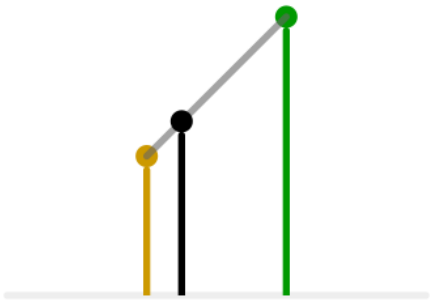


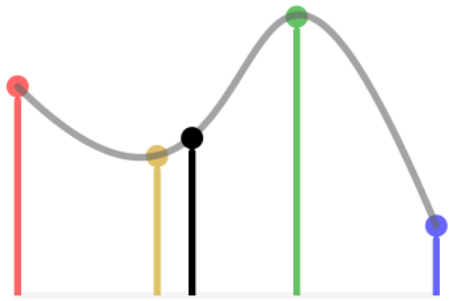
Image interpolation



1D nearest-neighbour



Linear



Cubic

kernel

response

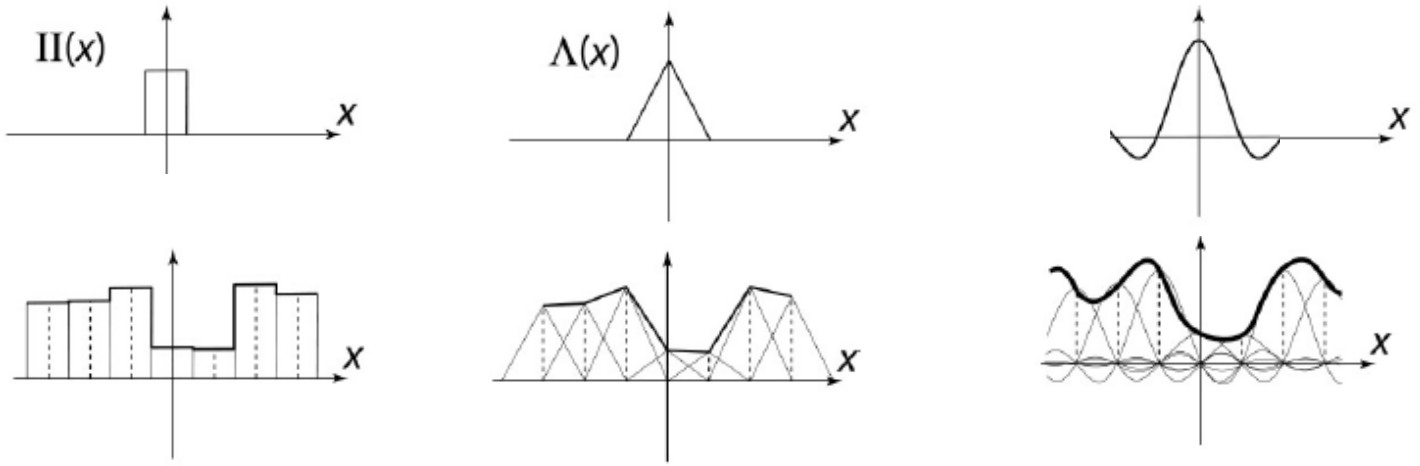
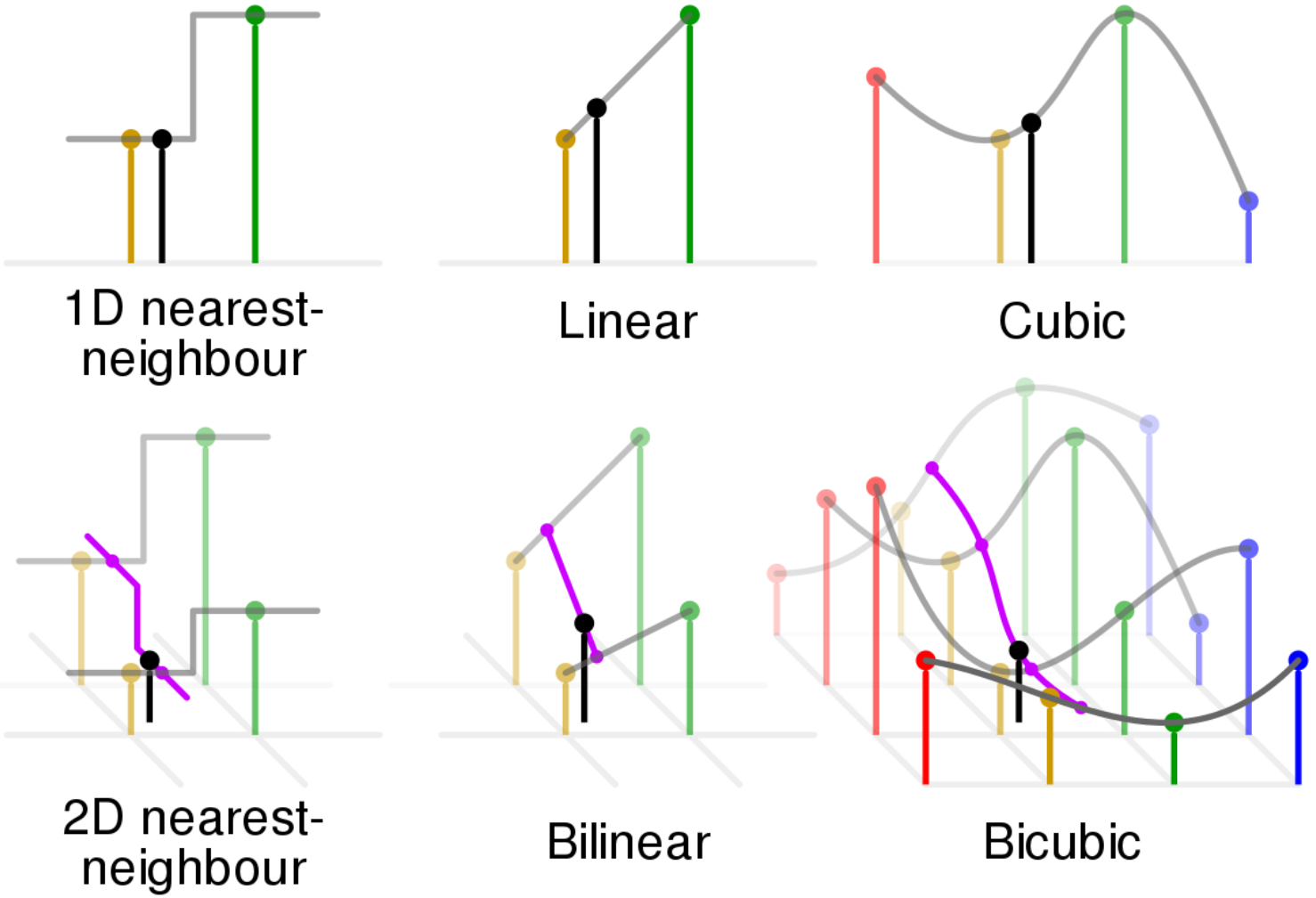
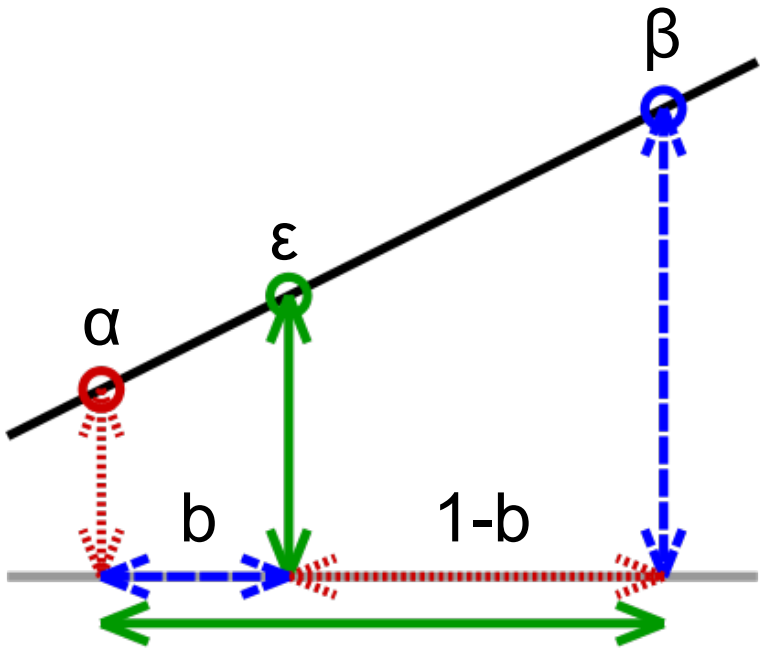


Image interpolation



Linear Interpolation

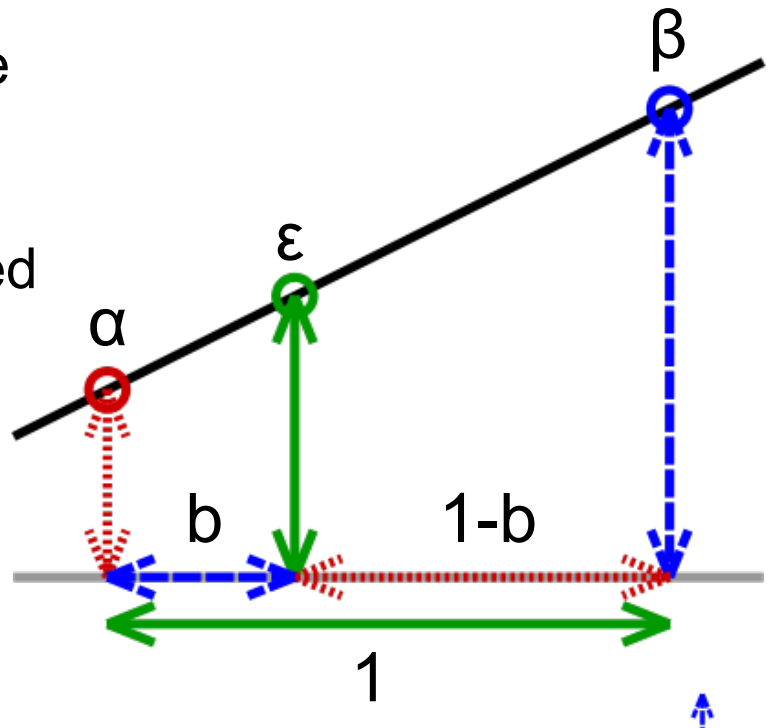


Linear Interpolation

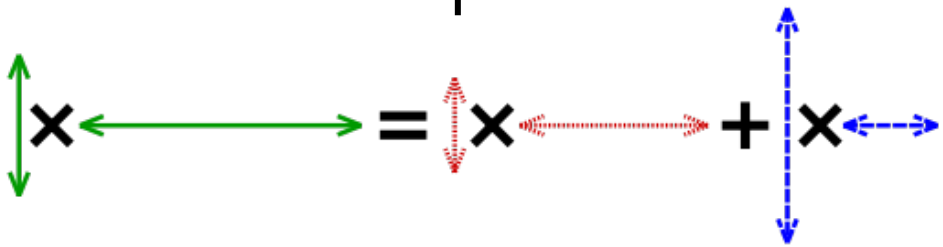
The value at the green circle multiplied by the distance between the red and blue circles

=

the sum of the value at the red circle multiplied by the distance between the green and blue circles, and the value at the blue circle multiplied by the distance between the green and red circles.



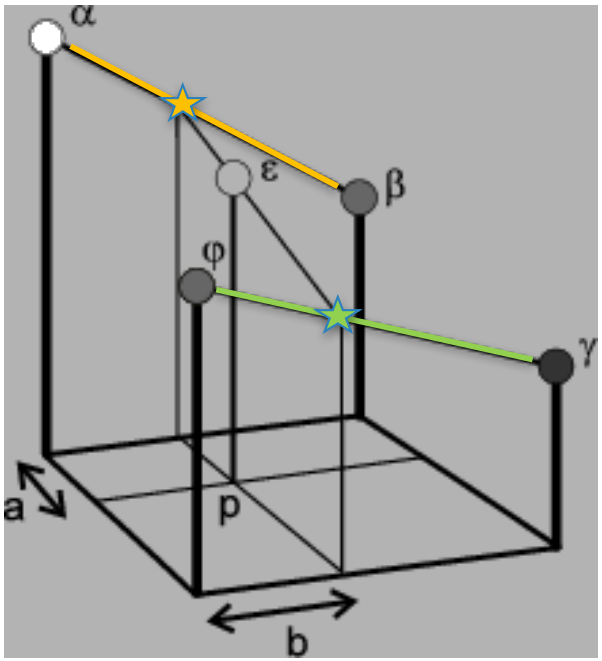
$$\varepsilon = (b\beta + (1-b)\alpha)$$



I(1) = 175 I(2) = 200

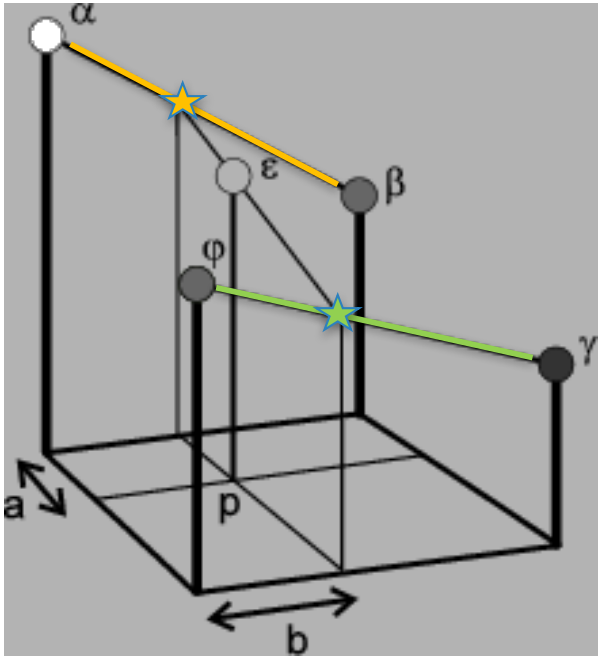
I(1.6) = ?

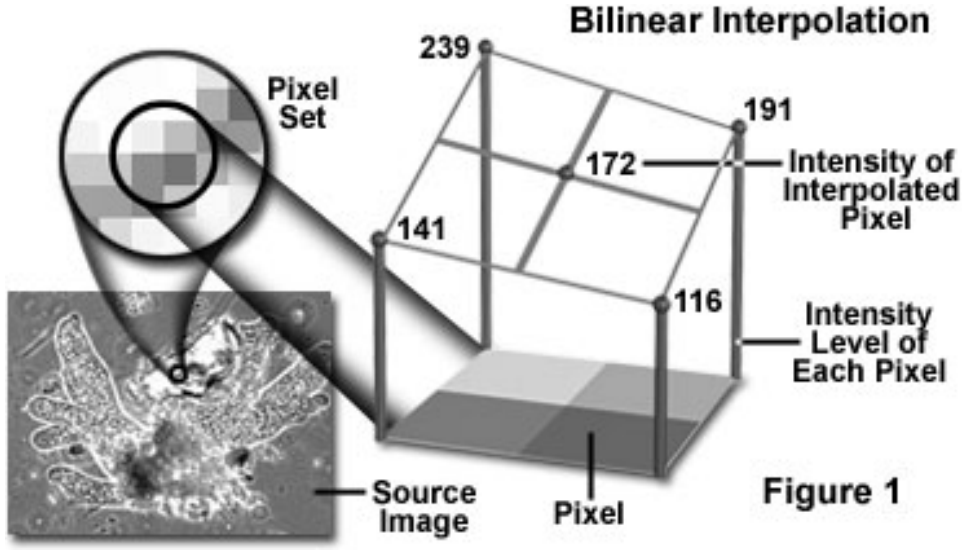
Bilinear interpolation



$$\varepsilon = a(b\gamma + (1-b)\varphi) + (1-a)(b\beta + (1-b)\alpha)$$

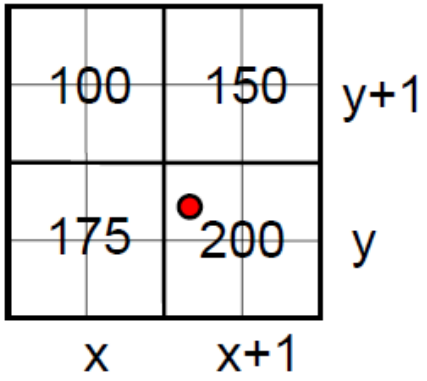
Bilinear interpolation





$$\varepsilon = a(b\gamma + (1-b)\varphi) + (1-a)(b\beta + (1-b)\alpha)$$

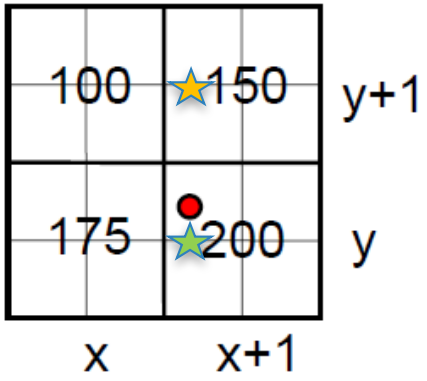
Example



Interpolated value for $(x+0.7, y+0.2)$?
x' y'

- $I(x',y) =$ (175 * .3) + (200 * .7) = 192.5
- $I(x',y+1) =$ (100 * .3) + (150* .7) = 135

Example



Interpolated value for $(x+0.7, y+0.2)$?
x' y'

- $I(x',y) =$ (175 * .3) + (200 * .7) = 192.5
- $I(x',y+1) =$ (100 * .3) + (150* .7) = 135

$I(x',y') =$ (192.5 * .8) + (135* .2) = 181

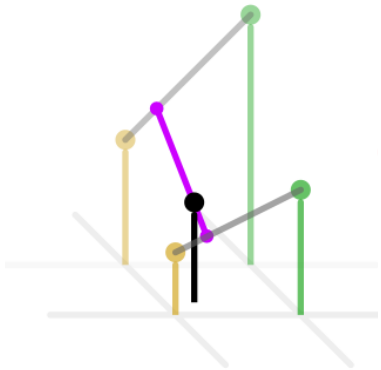
Example

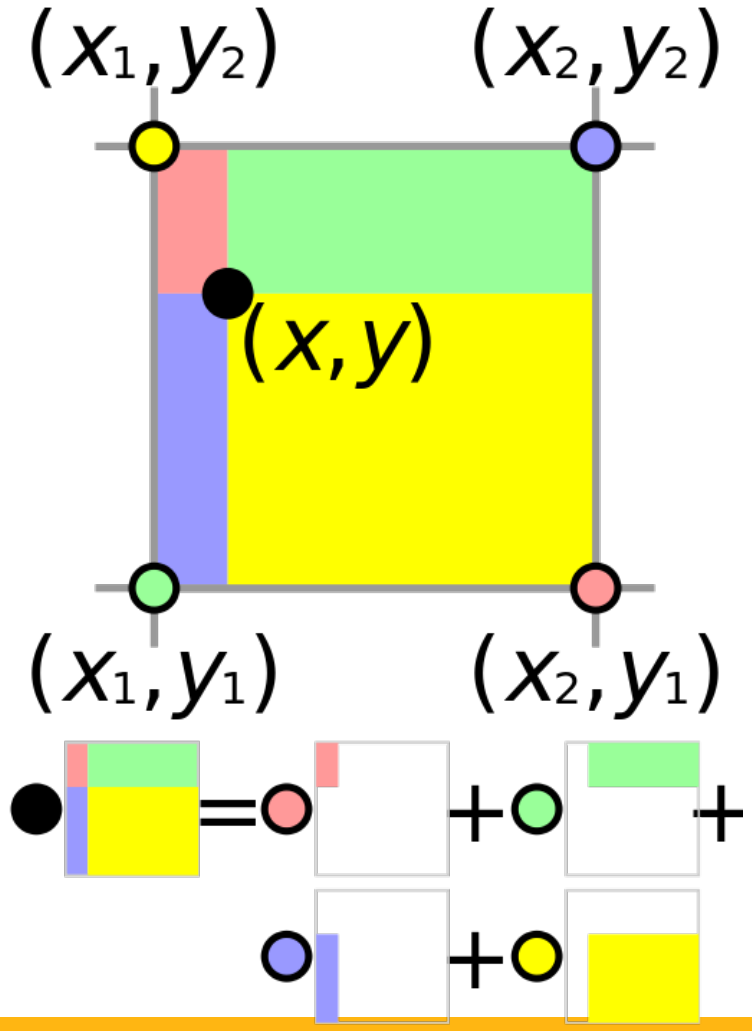
Y \ X	1	2
1	**55**	**45**
2	**16**	**55**

X=1.2, Y=1.35

Bilinear interpolation

The value at the black spot is the sum of the value at each colored spot multiplied by the area of the rectangle of the same color, divided by the total area of all four rectangles.





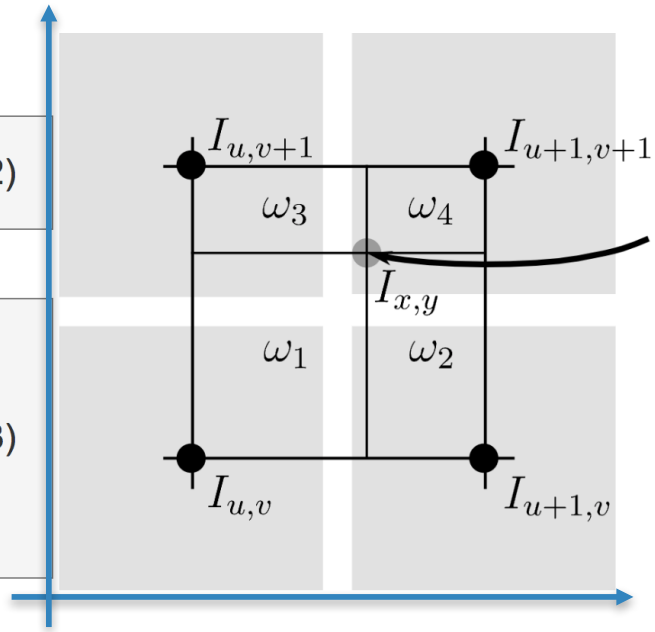
Bilinear interpolation

Bilinear interpolation determines the grey level from the weighted average of the four closest pixels and can be defined as

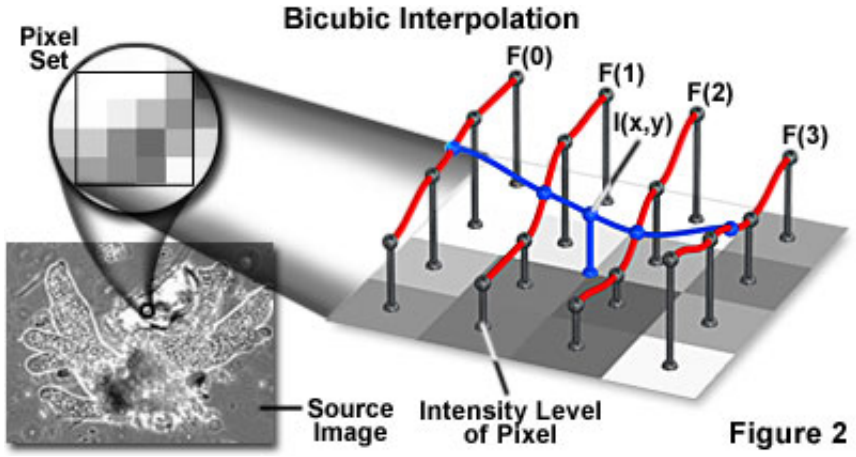
$$I_{x,y} = \omega_4 I_{u,v} + \omega_2 I_{u,v+1} + \omega_3 I_{u+1,v} + \omega_1 I_{u+1,v+1} \quad (3.2)$$

where

$$\begin{aligned} \omega_1 &= (x-u)(y-v) \\ \omega_2 &= (u+1-x)(y-v) \\ \omega_3 &= (x-u)(v+1-y) \\ \omega_4 &= (u+1-x)(v+1-y) \end{aligned} \quad (3.3)$$

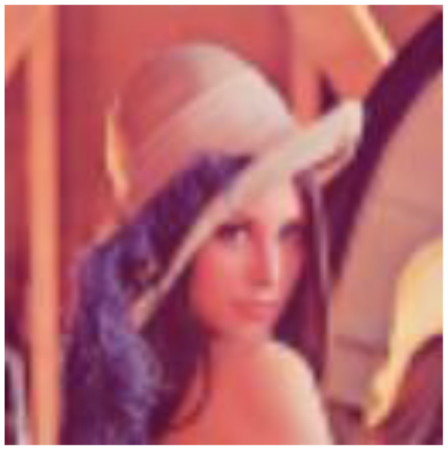


Bicubic interpolation



$$\varphi_{CC3}(x) = \begin{cases} \frac{3}{2}|x|^3 - \frac{5}{2}|x|^2 + 1 & \text{if} \quad 0 \le |x| \le 1, \\ -\frac{1}{2}|x|^3 + \frac{5}{2}|x|^2 - 4|x| + 2 & \text{if} \quad 1 \le |x| \le 2, \\ 0 & \text{if} \quad 2 \le |x|. \end{cases}$$

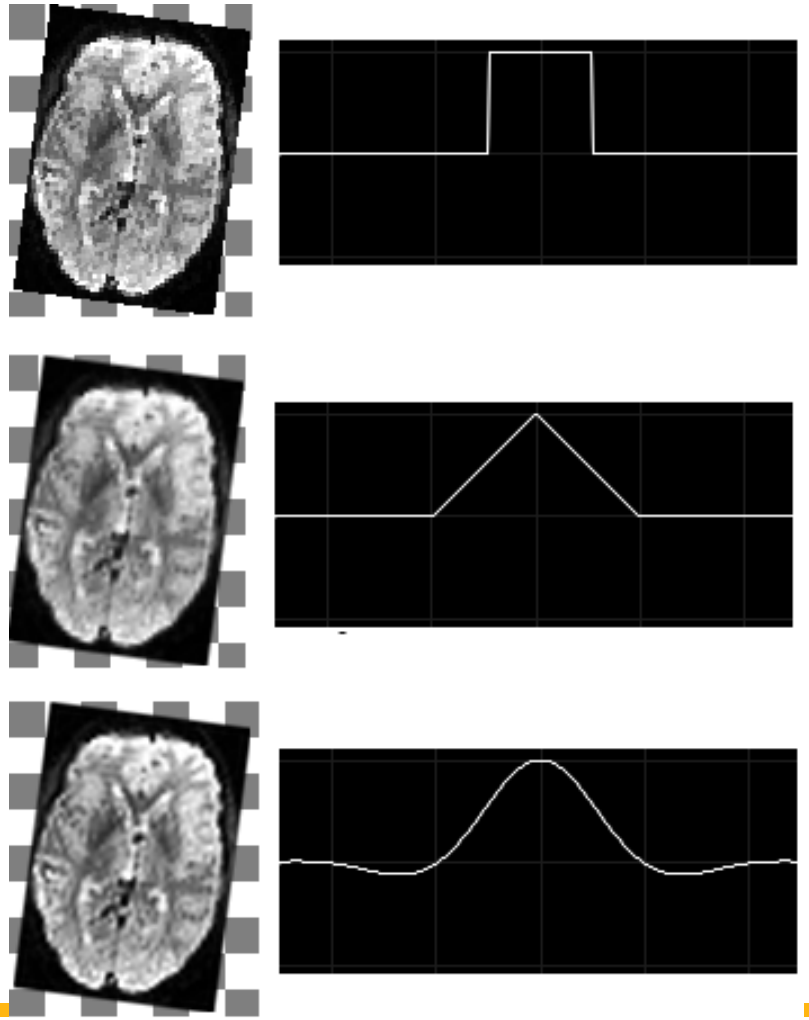
Nearest neighbor



Bilinear

Bicubic

- Cubic interpolation is better at retaining the original grey values than NN and bilinear interpolation.
- Computational requirement is up to 10 times higher than NN interpolation.
- Computational requirement is up to 2 times higher than bi-linear interpolation.



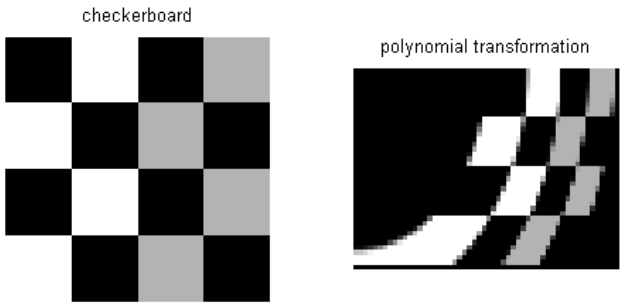
Matlab: how to infer a transformation?

- t = *cp2tform(input_points, base_points*);

```
pairs1 = [1 1; 5 21; 17 40; 28 1; 32 20; 45 40; 72 1; 77 20; 90 40];
pairs2 = [1 1; 1 21; 1 40; 20 1; 20 20; 20 40; 40 1; 40 20; 40 40];
t_poly = cp2tform(pairs1, pairs2, 'polynomial',2);
I = checkerboard(10, 2);
J = imtransform(I,t_poly);

subplot(1,2,1)
imshow(I), title('checkerboard')

subplot(1,2,2)
imshow(J), title('polynomial transformation')
```



How to find correspondence automatically?