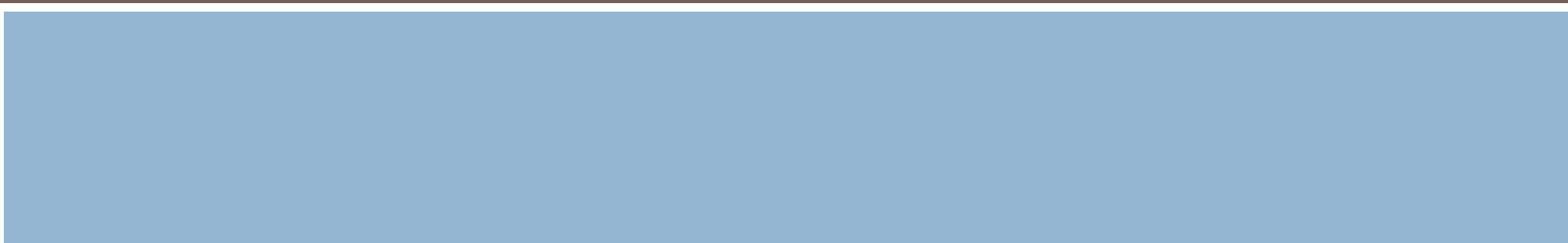
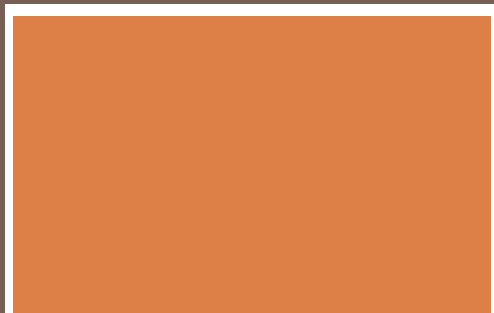
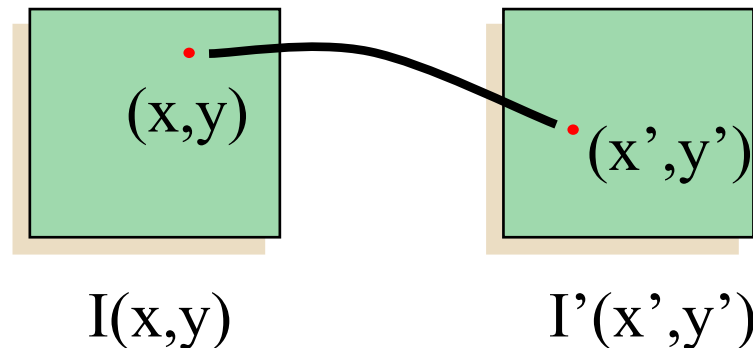


# GEOMETRIC OPERATIONS

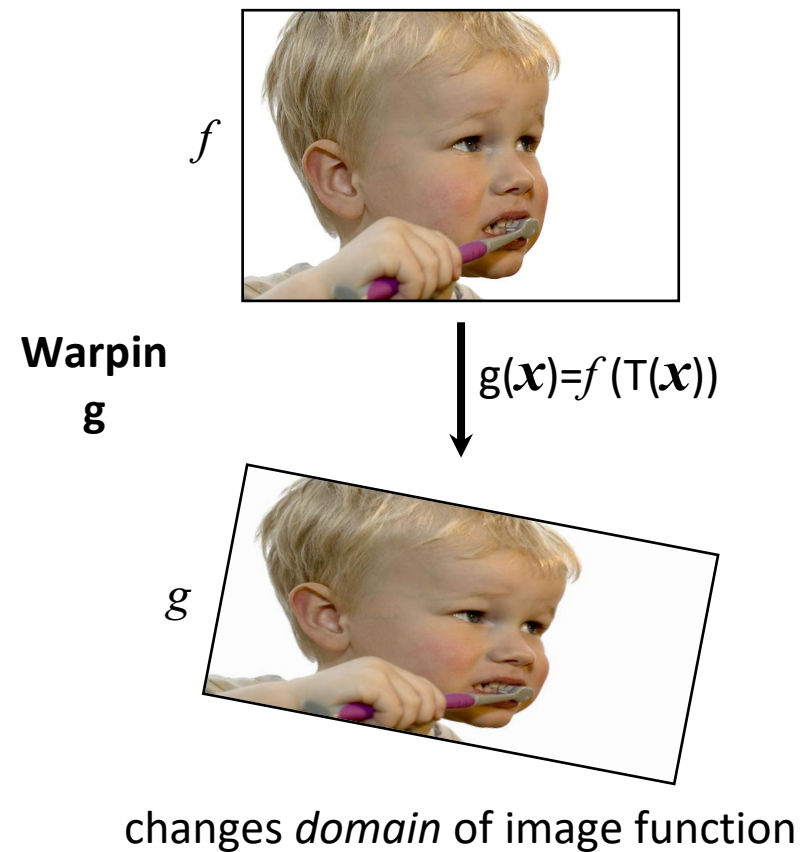
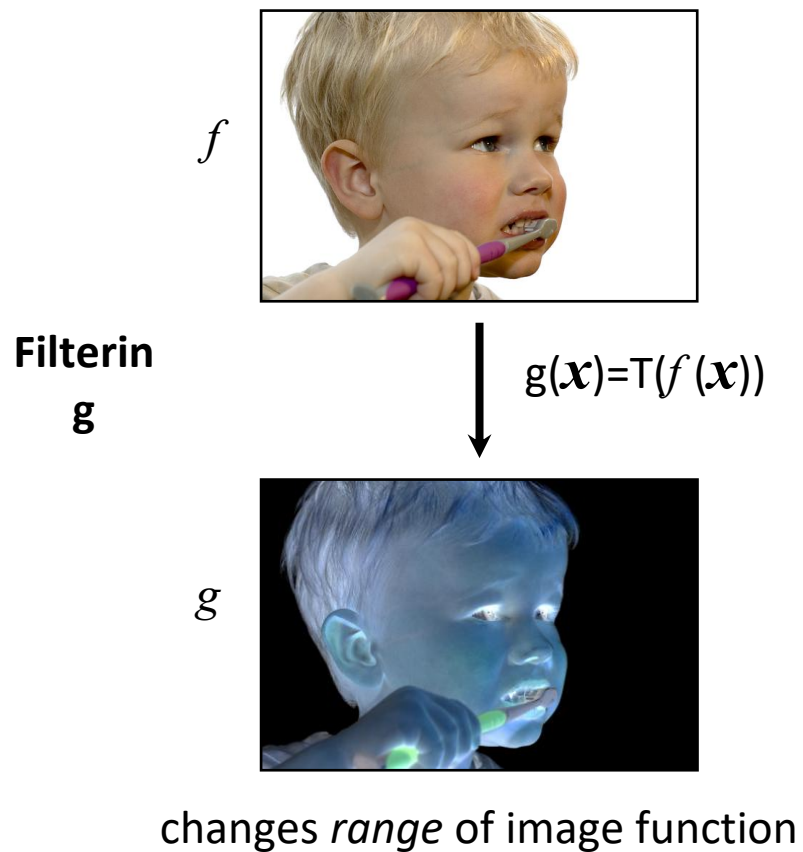


# Geometric Operations

- Previous operations have taken a sample at some location and changed the sample value (the light intensity) but left the location unchanged.
- Geometric operations take a sample and change its location in the destination while leaving the sample value unchanged.
- In general, geometric operations take a source pixel at some location  $(x,y)$  and map it to location  $(x', y')$  in the destination.



# Image transformations can we do?



# Warping example

- object recognition
- 3D reconstruction
- augmented reality
- image stitching

Given a set of matched feature points:

$$\{x_i, x'_i\}$$

point in one  
image

point in the  
other image

and a  
transformation:

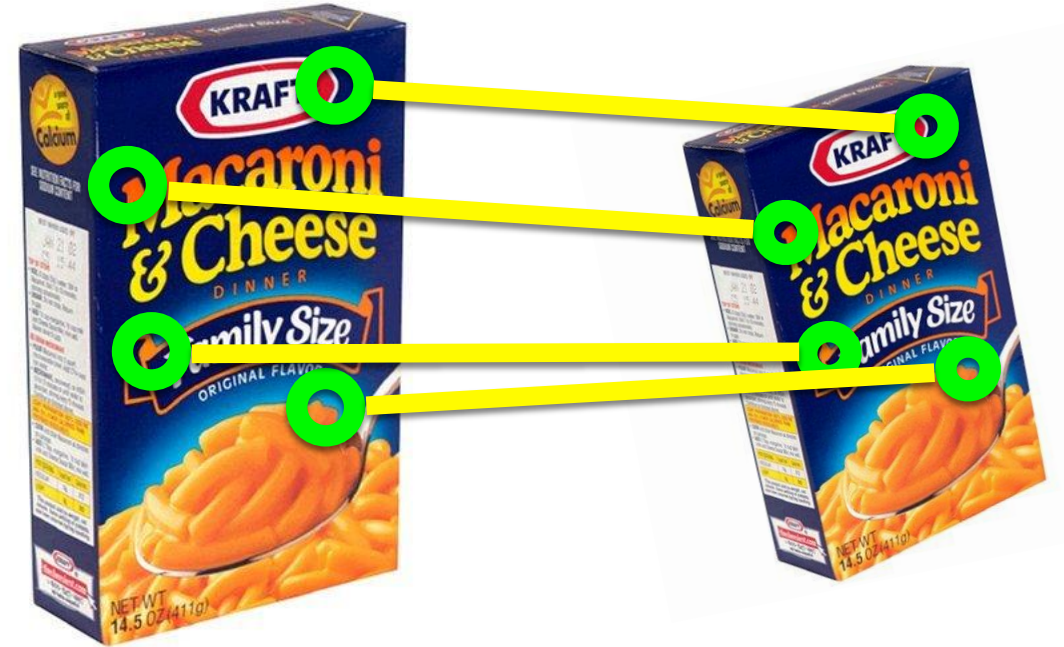
$$x' = f(x; p)$$

transformation  
function

parameters

find the best estimate of the  
parameters

$p$



What kind of transformation functions  $f$  are there?

# 2D transformations



translation



rotation



aspect



affine



perspective



cylindrical

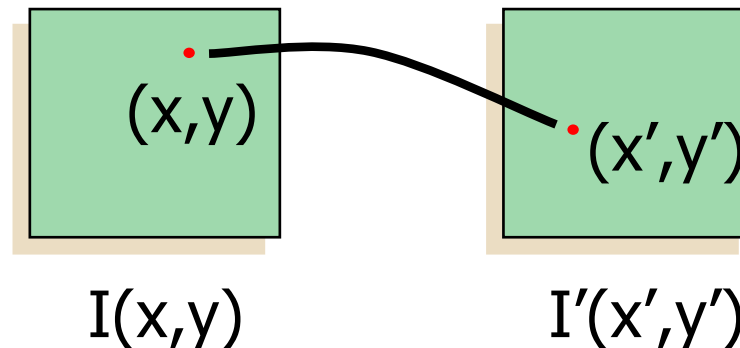
# Affine Transformations

- The mapping between  $(x,y)$  and  $(x', y')$  can be generalized as

$$\begin{aligned}x' &= T_x(x, y), \\ y' &= T_y(x, y).\end{aligned}$$

- $T_x$  and  $T_y$  are transformation functions that produce output coordinates based on the  $x$  and  $y$  coordinates of the input pixel.
- Both functions produce real values as opposed to integer coordinates and are assumed to be well defined at all locations in the image plane.

$$I(x,y) = I'(x',y') = I'(T_x(x,y), T_y(x,y))$$



# Affine Transformations

- The simplest kind of transformations are linear
  - ▢  $x'$  and  $y'$  are linearly related to  $(x, y)$
  - ▢ All linear transformations are known as ***affine*** transformations

$$x' = T_x(x, y) = a_1x + b_1y + c_1$$

$$y' = T_y(x, y) = a_2x + b_2y + c_2$$

- **Properties of affine transformations**
  - ▢ A **straight line** in the source is **straight** in the destination
  - ▢ **Parallel lines** in the source are **parallel** in the destination
- The six coefficients determine the exact effect of the transform.

# Affine Transformations

$$x' = T_x(x, y) = a_1x + b_1y + c_1$$

$$y' = T_y(x, y) = a_2x + b_2y + c_2$$

- Can be written in homogeneous matrix equation

$$\mathbf{x}' = \mathbf{T}.\mathbf{x}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



# Affine Transformations - Translation



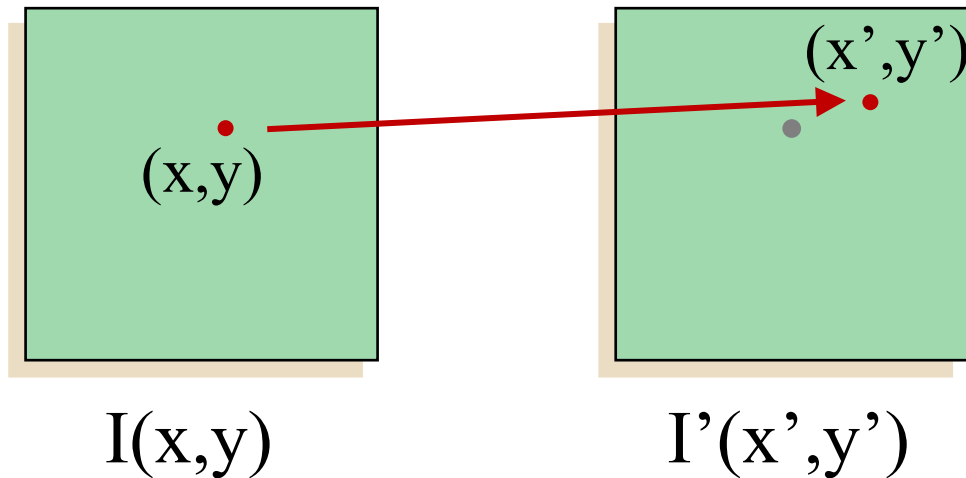
$$x' = T_x(x, y) = x + 3$$

$$y' = T_y(x, y) = y - 1$$

$$a_1 = 1, \quad b_1 = 0, \quad c_1 = 3 = t_x$$

$$a_2 = 0, \quad b_2 = 1, \quad c_2 = -1 = t_y$$

$$I'(x + 3, y - 1) = I(x, y)$$



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

# Affine Transformations - Scaling

- To shrink or zoom the size of an image (or part of an image).
- To change the visual appearance of an image;
- To alter the quantity of information
- To use as a low-level pre-processor in multi-stage image processing chain which operates on features of a particular scale.



Uniform  
Scaling



Non-  
uniform  
Scaling

$$\begin{aligned}x' &= T_x(x, y), = x * s_x \\y' &= T_y(x, y), = y * s_y\end{aligned}$$

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

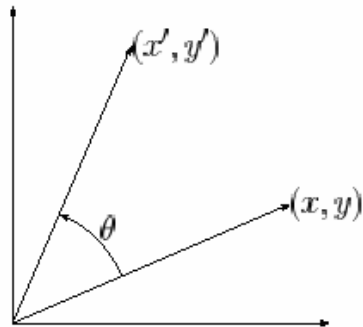
# Affine Transformations - Rotation

- maps the input position onto a output position by rotating it through an angle about an origin.
- Commonly used to improve the visual appearance of an image.
- useful as a pre-processor in applications where directional operators are involved.
- Mapping of a point  $(x,y)$  to another  $(x',y')$  through a counter-clockwise rotation of  $\theta$



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

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



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

# 2D planar transformations

## Polar coordinates...

$$x = r \cos(\varphi)$$

$$y = r \sin(\varphi)$$

$$x' = r \cos(\varphi + \theta)$$

$$y' = r \sin(\varphi + \theta)$$

## Trigonometric Identity...

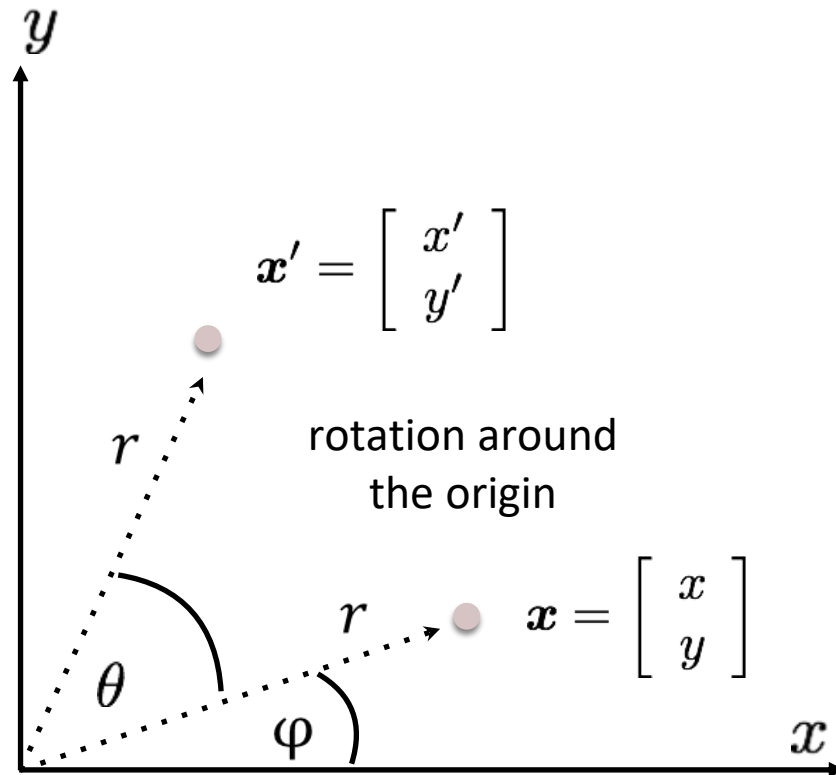
$$x' = r \cos(\varphi) \cos(\theta) - r \sin(\varphi) \sin(\theta)$$

$$y' = r \sin(\varphi) \cos(\theta) + r \cos(\varphi) \sin(\theta)$$

## Substitute...

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

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



or in matrix  
form:

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

# Affine Transformations

## □ Shear:



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

## □ Reflection



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

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

# Types of linear 2D transformations

- **Rigid (Euclidean)** transformation:

- Translation + Rotation (distance preserving).

- **Similarity** transformation:

- Translation + Rotation + Uniform Scale (angle preserving).

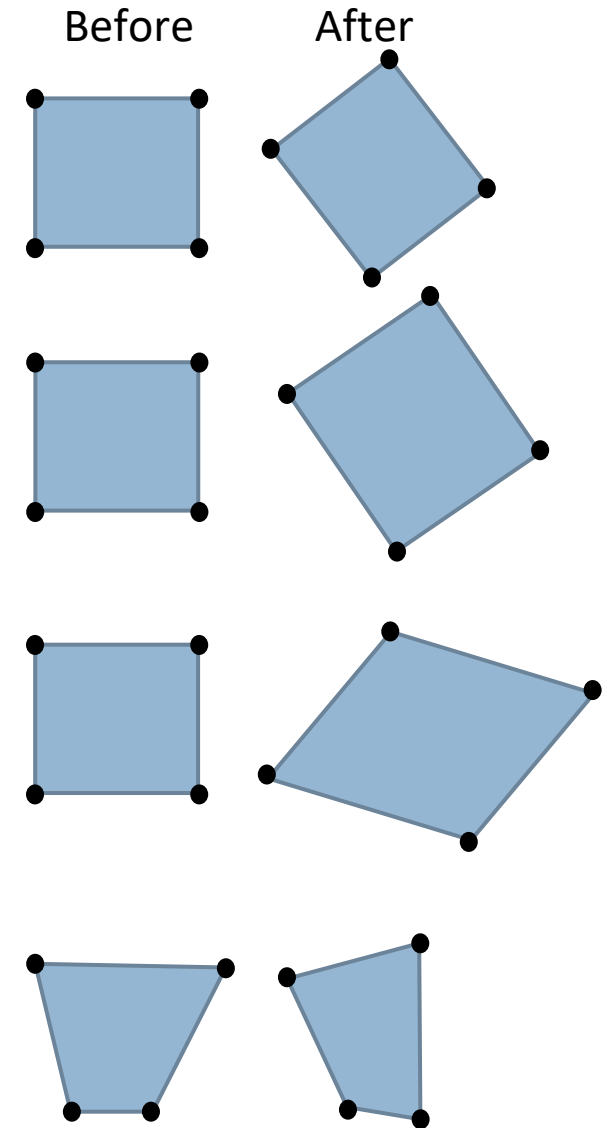
- **Affine** transformation:

- Translation + Rotation + Scale + Shear (parallelism preserving).

- **Projective** transformation

- Cross-ratio preserving

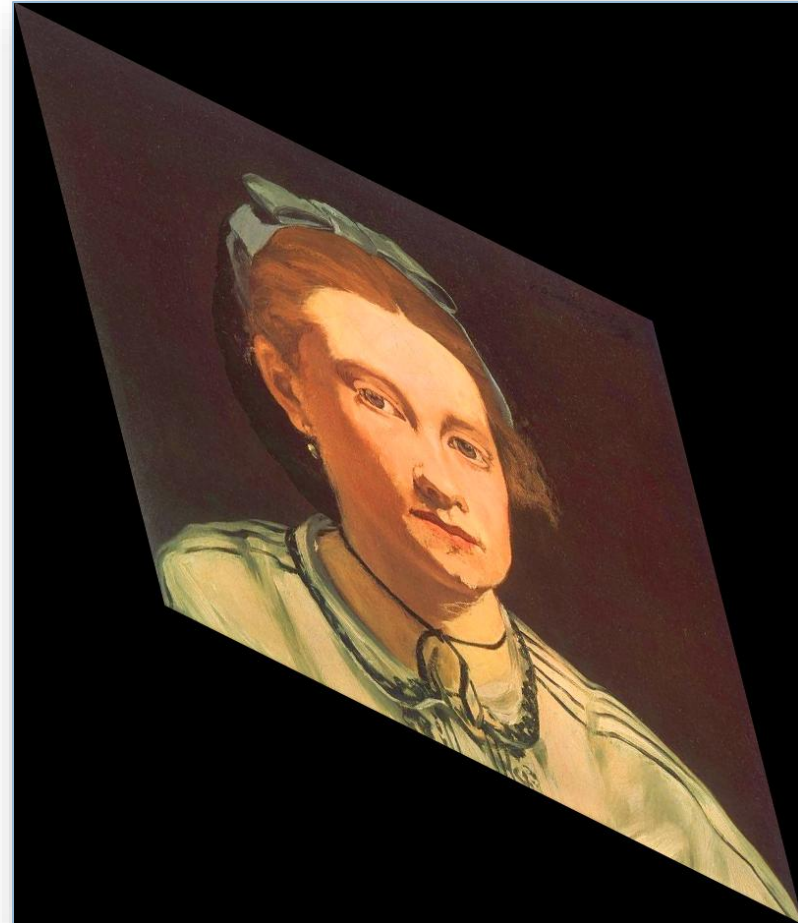
- All above transformations are groups where  $\text{Rigid} \subset \text{Similarity} \subset \text{Affine} \subset \text{Projective}$



# Affine Transformations

Explain what the following transformation matrices accomplishes

1.0	0.25	0.0
0.5	1.0	0.0
0.0	0.0	1.0

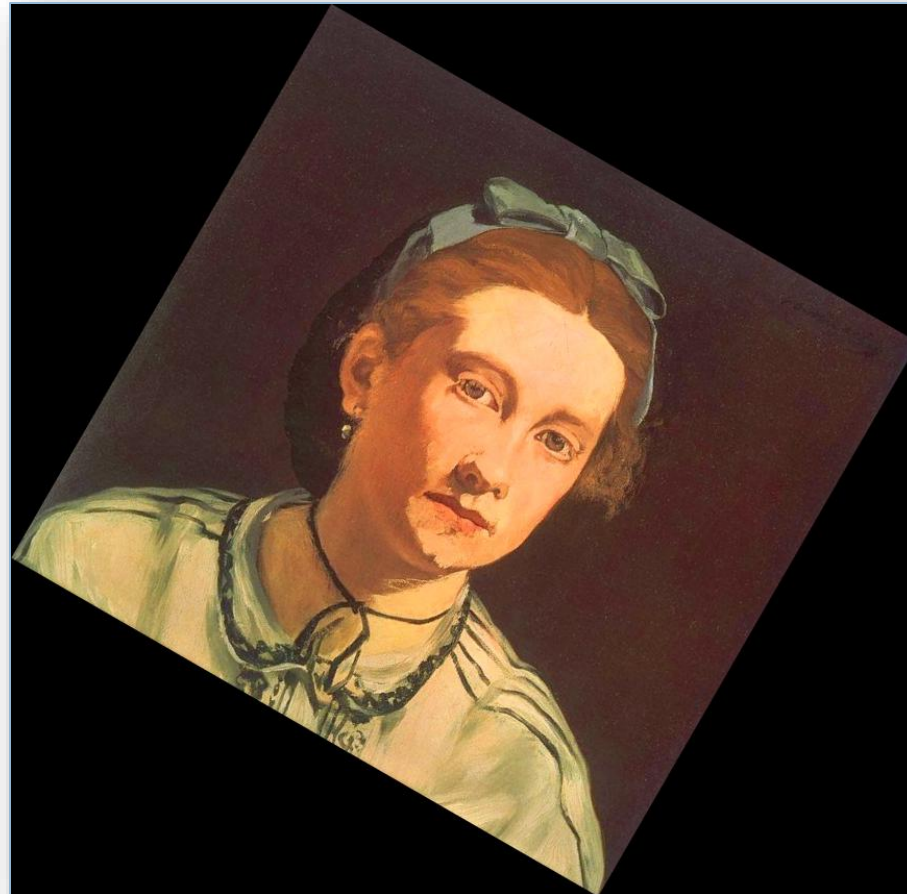




# Affine Transformations

Explain what the following transformation matrices accomplishes

.87	-0.5	0.0
0.5	.87	0.0
0.0	0.0	1.0

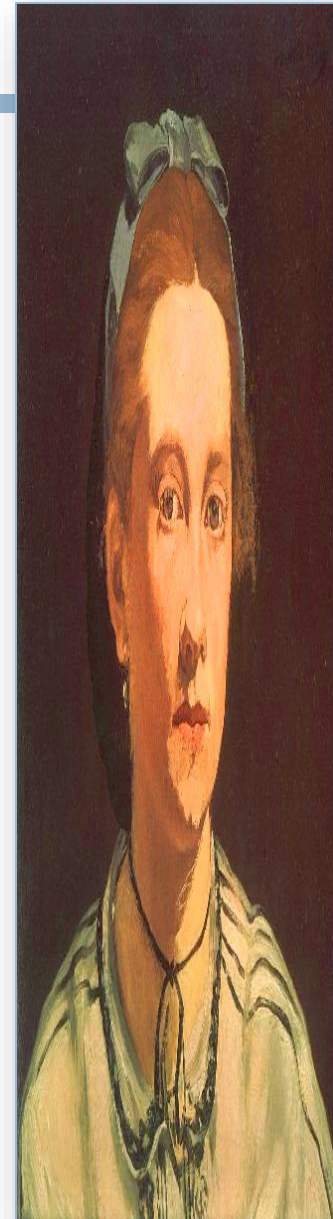




# Affine Transformations

Explain what the following transformation matrix accomplishes

0.5	0.0	0.0
0.0	2.0	0.0
0.0	0.0	1.0



# Affine Transformations

Explain what the following transformation matrices accomplishes

1.0	0.25	0.0
0.0	1.0	0.0
0.0	0.0	1.0



# Homogeneous Coordinates

- Homogeneous Coordinates is a mapping from  $\mathbb{R}^n$  to  $\mathbb{R}^{n+1}$ :

$$(x, y) \rightarrow (X, Y, W) \equiv (tx, ty, t)$$

- Note:  $(tx, ty, t)$  all correspond to the same non-homogeneous point  $(x, y)$ . E.g.  $(2, 3, 1) \equiv (6, 9, 3) \equiv (4, 6, 2)$ .

- Inverse mapping:

$$(X, Y, W) \rightarrow \left( \frac{X}{W}, \frac{Y}{W} \right) = (x, y)$$

# Some 2D Transformations – homogeneous eqn

□ Translation :

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

$$\mathbf{X}' = \mathbf{T}_{(t_x, t_y)} \mathbf{X}$$

□ Rotation:

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

$$\mathbf{X}' = \mathbf{R}_{(\theta)} \mathbf{X}$$

□ Scale:

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

$$\mathbf{X}' = \mathbf{S}_{(s_x, s_y)} \mathbf{X}$$

□ Shear:

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

$$\mathbf{X}' = \mathbf{Sh}_{(sh_x, sh_y)} \mathbf{X}$$

# Affine Transformations

- A single homogeneous matrix can also represent a *sequence* of individual affine operations.
- Let A and B represent affine transformation matrices
  - the affine matrix corresponding to the application of A followed by B is given as BA
  - BA is itself a homogeneous transformation matrix.
  - Matrix multiplication, also termed concatenation, therefore corresponds to the sequential composition of individual affine transformations.
  - Note that the order of multiplication is both important and opposite to the way the operations are mentally envisioned.

# Affine Transformations

- While we speak of transform A followed by transform B, these operations are actually composed as matrix B multiplied by (or concatenated with) matrix A.
- Assume, for example, that matrix A represents a rotation of 30 degrees about the origin and matrix B represents a horizontal shear by a factor of .5. The affine matrix corresponding to the rotation followed by shear is given as BA.

$$\begin{array}{c} \mathbf{B} \\ \left[ \begin{array}{ccc} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right] \end{array} \cdot \begin{array}{c} \mathbf{A} \\ \left[ \begin{array}{ccc} .866 & -0.5 & 0 \\ 0.5 & .866 & 0 \\ 0 & 0 & 1 \end{array} \right] \end{array} = \begin{array}{c} \mathbf{B.A} \\ \left[ \begin{array}{ccc} 1.116 & -0.067 & 0 \\ 0.5 & .866 & 0 \\ 0 & 0 & 1 \end{array} \right] \end{array} . \quad (7.4)$$

# Matrix composition

Transformations can be combined by matrix multiplication:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \left( \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \Theta & -\sin \Theta & 0 \\ \sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$p' = \text{translation}(t_x, t_y) \quad \text{rotation}(\theta) \quad \text{scale}(s, s) \quad p$

Does the multiplication order matter?

# Some 2D Transformations – homogeneous eqn

□ Translation :

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

□ Rotation:

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

□ Scale:

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

□ Shear:  $\begin{bmatrix} X' \\ Y' \\ W' \end{bmatrix} = \begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

□ Affine:

$$\begin{bmatrix} X' \\ Y' \\ W' \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

□ Projective:

$$\begin{bmatrix} X' \\ Y' \\ W' \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ e & f & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



# Affine transformations

□ Combinations of arbitrary (4-DOF) linear transformations; and **translations**

□ Properties of affine transformations:

▣ origin does not necessarily map to origin

▣ lines map to lines

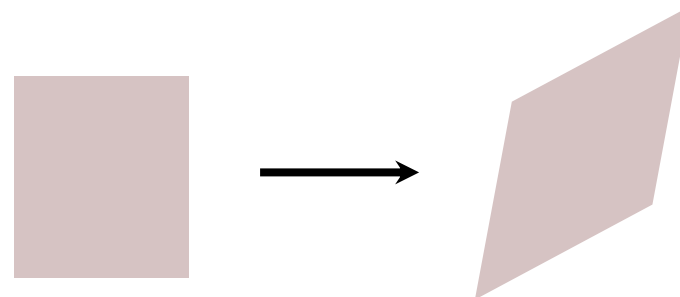
▣ parallel lines map to parallel lines

▣ ratios are preserved

▣ compositions of affine transforms are also affine transforms

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

Remains  
same



# Projective transformations

- Combinations of
  - affine transformations; and
  - projective warps



Define input quadrilateral



Define base quadrilateral



After projective transformation



Cropped image

# Projective transformations

## □ Combinations of

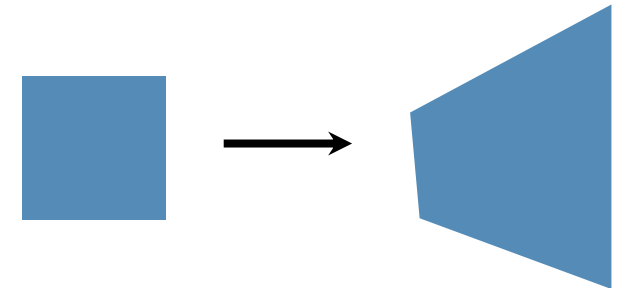
- affine transformations; and
- projective warps

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$





8 DOF: vectors (and therefore matrices) are defined up to scale)

## □ Properties:

- origin does not necessarily map to origin
- lines map to lines
- parallel lines do not necessarily map to parallel lines
- ratios are not necessarily preserved
- compositions of projective transforms are also projective transforms



# Hierarchy of Linear 2D Transformations

Group	Matrix	Distortion	Invariant properties
Projective 8 dof	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$		Concurrency, collinearity, <b>order of contact</b> : intersection (1 pt contact); tangency (2 pt contact); inflections (3 pt contact with line); tangent discontinuities and cusps. cross ratio (ratio of ratio of lengths).
Affine 6 dof	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Parallelism, ratio of areas, ratio of lengths on collinear or parallel lines (e.g. midpoints), linear combinations of vectors (e.g. centroids). The line at infinity, $l_\infty$ .
Similarity 4 dof	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Ratio of lengths, angle. The circular points, <b>I, J</b> (see section 1.7.3).
Euclidean 3 dof	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Length, area

# Issues with geometric ops

---

- During point and spatial processing
  - ▢ Source and destination images were the same size
  - ▢ Color depth was occasionally different
  
- During geometric processing
  - ▢ Source and destination images may not be the same size
  - ▢ Output locations may not be integer values!
  - ▢ 'Gaps' may occur when mapping inputs to outputs

# Point Transformation

- Example. Consider rotating an image by 30 degrees clockwise. Note that  $\cos(30)$  is .866 and  $\sin(30)$  is -.5.
- The transformation is given by

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} .866 & -0.5 & 0 \\ 0.5 & .866 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \quad (7.5)$$

- Consider relocating the sample at (10, 20)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} .866 & -0.5 & 0 \\ 0.5 & .866 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 20 \\ 1 \end{bmatrix} = \begin{bmatrix} -1.34 \\ 22.32 \\ 1 \end{bmatrix}. \quad (7.6)$$

# Two Issues

## □ Two issues:

- ❓ **Dimensionality:** The destination image may not be large enough to contain all of the processed samples
- ❓ **Mapping:** Transformed locations are not integers: How can we place a source sample at a non-integer location in the destination?



(a) Source image.



(b) Rotation.



(c) Expanded view of rotated image.

Figure 7.2. Destination dimensionality under rotation.

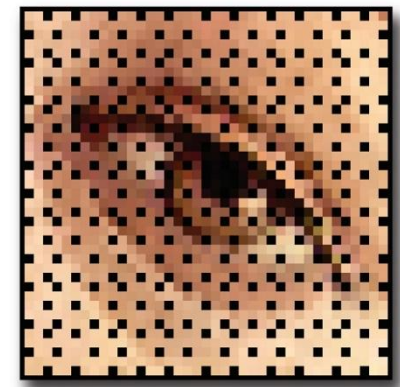
# Two Issues: Dimensionality

- Consider a source image that is rotated about the origin such that some pixels are mapped outside of the bounds of the source.
- Implementations must allow the destination to contain the entire rotated image.
  - Both the width and height of the destination image must be increased beyond that of the source.
  - Can compute the destination dimensions by **transforming the bounds** and using the **width and height of the bounds** as the destination dimensions.



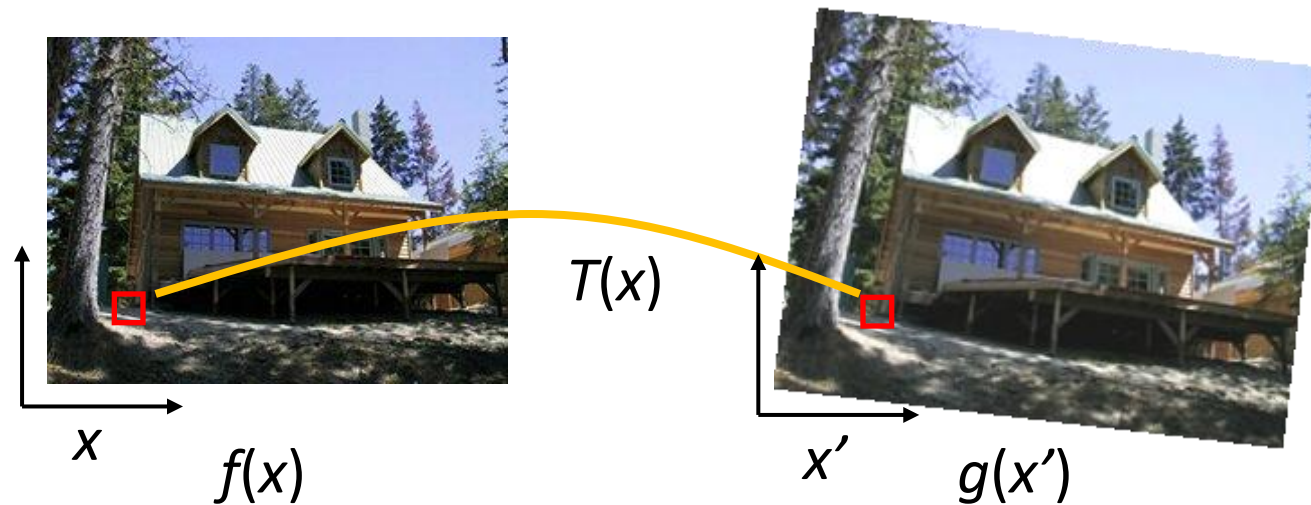
# Two Issues: Mapping

- how integer-valued source coordinates are mapped onto integer-valued destination coordinates
  - **Forward mapping** takes each pixel of the source image and copies it to a location in the destination by **rounding** the destination coordinates so that they are integer values.
    - generally poor results since certain pixels of the destination image may remain unfilled.
    - Example: a source image is rotated by 45 degrees using a forward mapping strategy.
    - Example: scaling an image to make it larger!



# Forward Warping

- **Send** each pixel's intensity/color  $f(\mathbf{x})$  **to** its corresponding location  $\mathbf{x}' = T(\mathbf{x})$  in  $g(\mathbf{x}')$ 
  - What if pixel lands “between” two pixels?

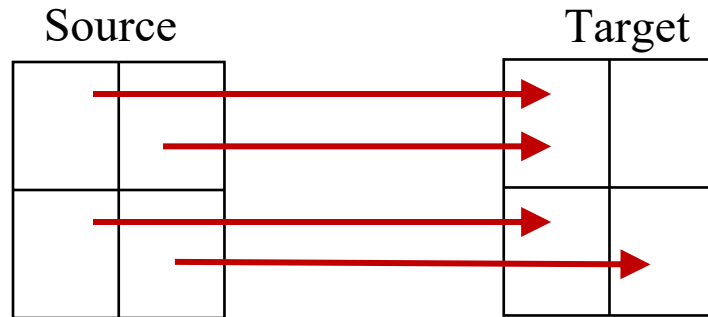


# Forward Mapping

Forward mapping:

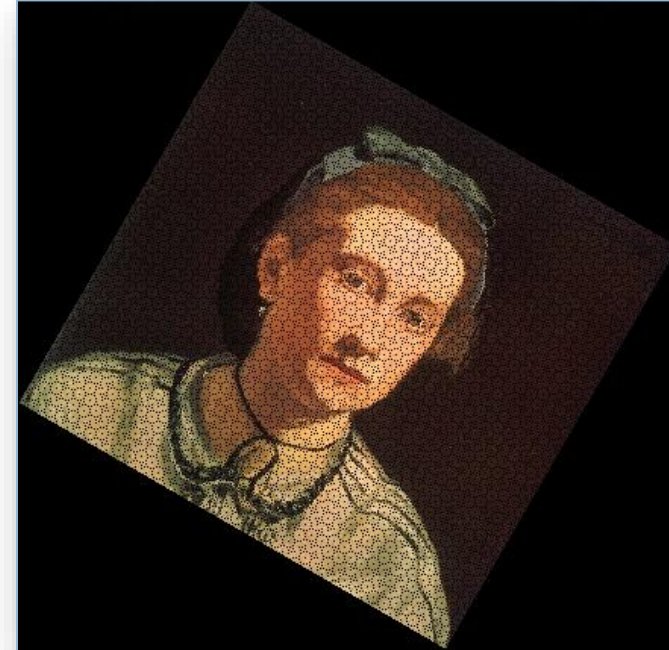
$$x' = T_x(x, y)$$

$$y' = T_y(x, y)$$



Problems with forward mapping due to sampling:

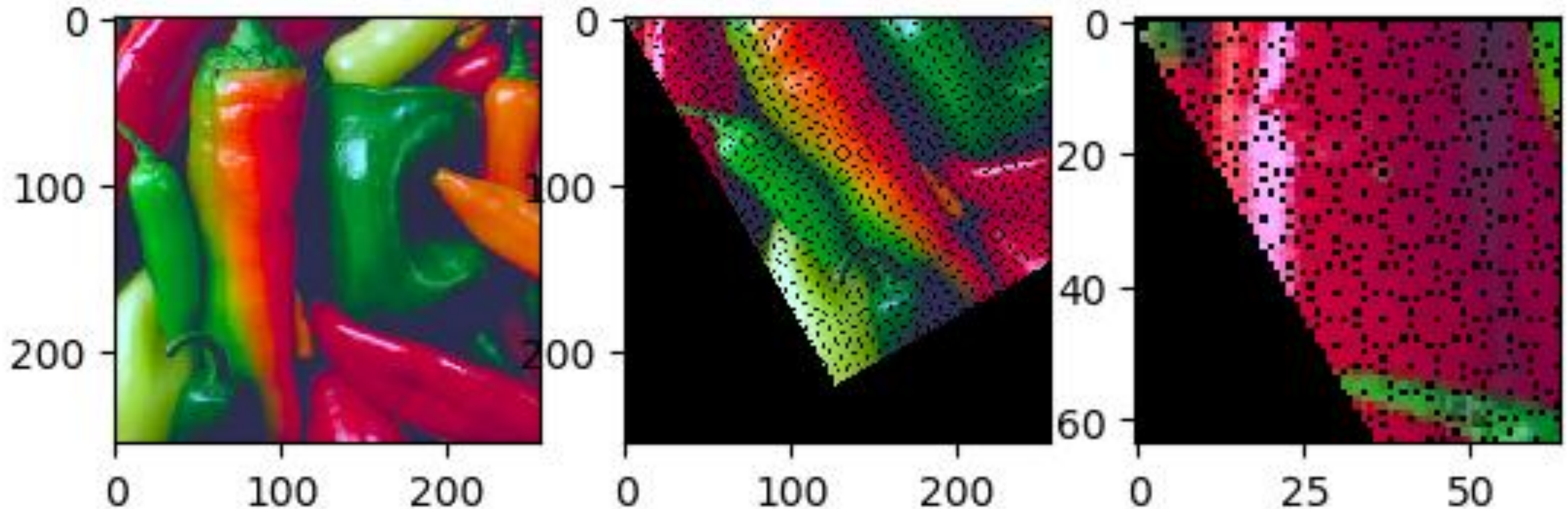
- Holes (some target pixels are not populated)
- Overlaps (some target pixels assigned few colors)



# Forward Mapping

```
def rotator(angle):  
    ca = np.cos(angle)  
    sa = np.sin(angle)  
    R = np.array([[ca, -sa], [sa, ca]])  
    def rotate(t):  
        return pixel(R @ np.array(t))  
    return rotate
```

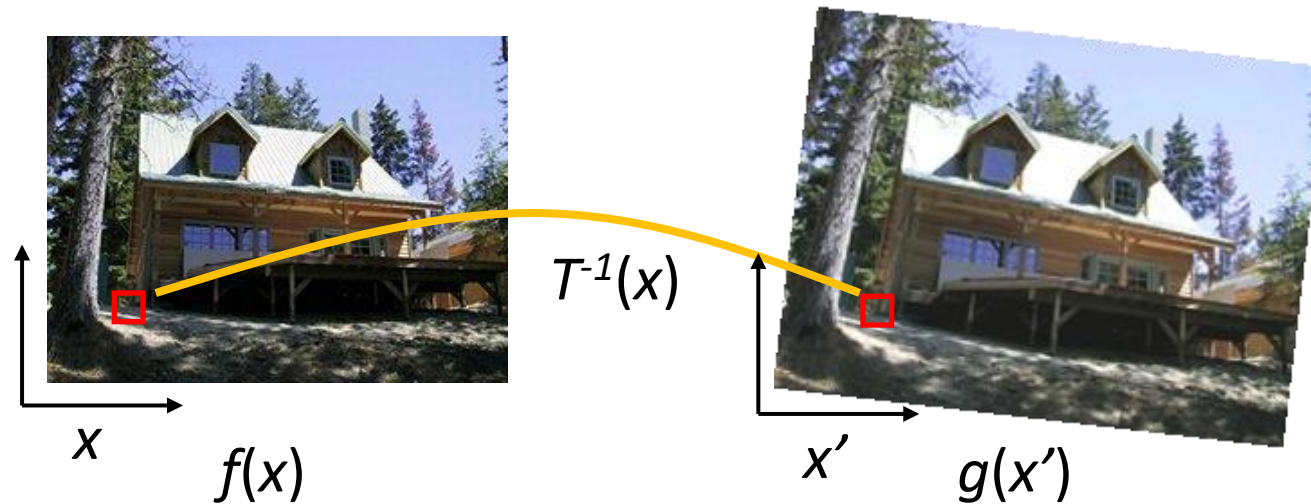
```
a = plt.imread('images/peppers.png')  
plt.subplot(131); plt.imshow(a);  
b = geoOp_fwd(a, rotator(np.pi/6))  
plt.subplot(132); plt.imshow(b);  
plt.subplot(133); plt.imshow(b[:64,:64]);  
plt.show();
```





# Inverse/Backward Warping

- **Get** each pixel's color/intensity  $g(\mathbf{x}')$  **from** its corresponding location  $\mathbf{x} = T^{-1}(\mathbf{x}')$  in  $f(\mathbf{x})$ 
  - What if pixel comes from “between” two pixels?



# Backward mapping

- Backward mapping solves the gap problem caused by forward mapping.
  - ▢ An empty destination image is created and each location in the destination is mapped backwards onto the source.
  - ▢ The source location may not be integer-valued coordinates; hence a sample value is obtained via interpolation.
- Let  $\mathbf{T}$  be a given affine transform matrix and
  - ▢ let  $\mathbf{x} = [x, y, 1]^T$  be a location in the given source image
  - ▢  $\mathbf{x}' = [x', y', 1]^T$  be a location in the destination image such that
$$\mathbf{x}' = \mathbf{T}\mathbf{x}$$
- We can backward map the transformation as

$$\mathbf{x} = \mathbf{T}^{-1} \mathbf{x}'$$

# Basic Inverse Operations

- Inverse Mapping of a point  $(x',y')$  to  $(x,y)$

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

Translation

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

scaling

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

Rotation

# Inverse Transformations

□ Translation :

$$\mathbf{T}_{(t_x, t_y)}^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

□ Scale:

$$\mathbf{S}_{(s_x, s_y)}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

□ Rotation:

$$\mathbf{R}_{(\theta)}^{-1} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

□ Shear:

$$\mathbf{Sh}_{(sh_x, sh_y)}^{-1} = \begin{bmatrix} \frac{-1}{1 - sh_x sh_y} & \frac{sh_x}{1 - sh_x sh_y} & 0 \\ \frac{sh_y}{1 - sh_x sh_y} & \frac{-1}{1 - sh_x sh_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



# Inverse Transformations

- The inverse of a 2D linear transformation is

$$A^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

- Let an affine transformation matrix  $M = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$

$$\mathbf{x}' = M.\mathbf{x}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \left[ \begin{array}{cc|c} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{array} \right] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = M.\mathbf{x}$$

$$\begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{T} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

# Inverse Transformations

- The inverse of a 2D linear transformation is

$$A^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

- Let an affine transformation matrix  $M = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$

$$\mathbf{x}' = \mathbf{M} \cdot \mathbf{x}$$

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

$$\begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{T} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = \mathbf{A} \cdot \mathbf{x} + \mathbf{T}$$

# Inverse Transformations

- Let an affine transformation matrix

$$\mathbf{x}' = \mathbf{A}.\mathbf{x} + \mathbf{T}$$

$$\mathbf{A}.\mathbf{x} = \mathbf{x}' - \mathbf{T}$$

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{x}' - \mathbf{A}^{-1}\mathbf{T}$$

$$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{T} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix}$$

$$\mathbf{x} = \mathbf{M}^{-1}\mathbf{x}'$$

$$A^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$M = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \\ 0 & 0 \end{bmatrix}$$

$$\mathbf{M}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \\ 0 & 0 \end{bmatrix}$$

# Backward Mapping

- Inverse mapping:

$$x = T_x^{-1}(x', y')$$

$$y = T_y^{-1}(x', y')$$

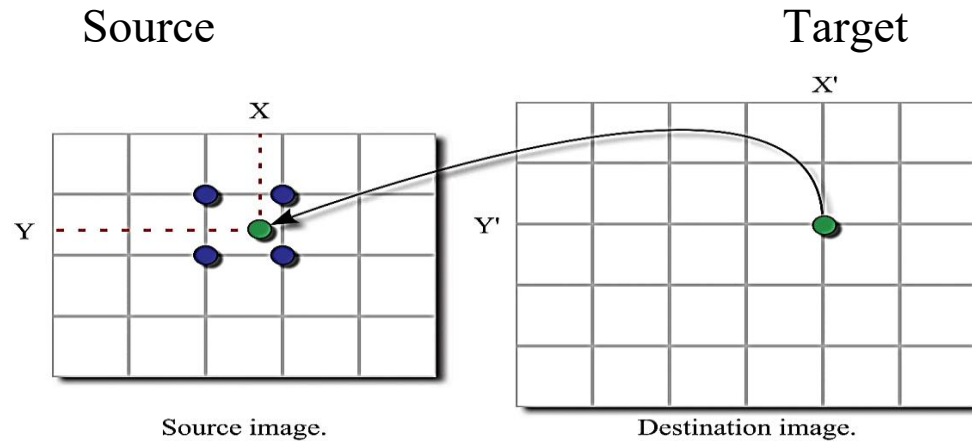


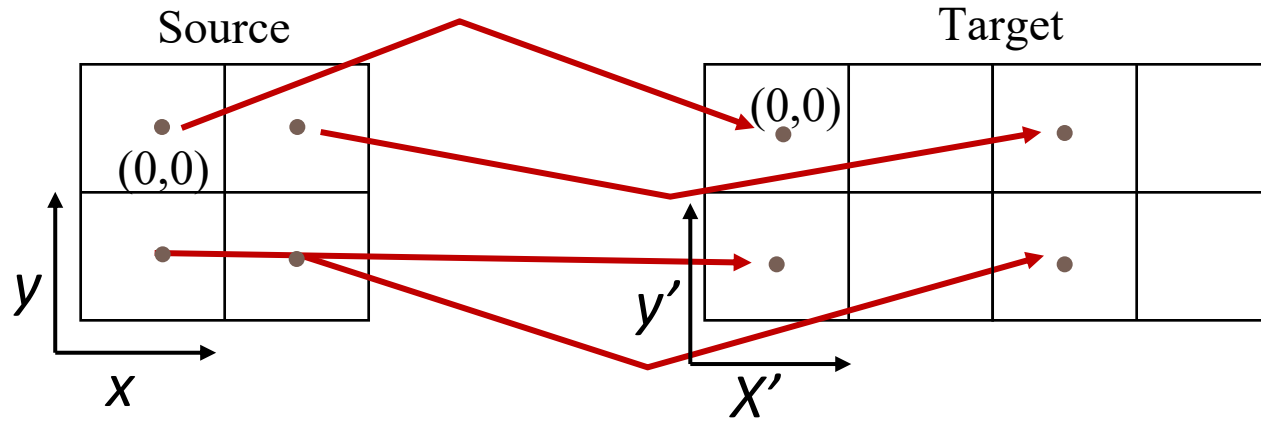
Figure 7.4. Reverse mapping.

- Each target pixel assigned a single color.
- Color Interpolation is required.

# Example: Scaling along X

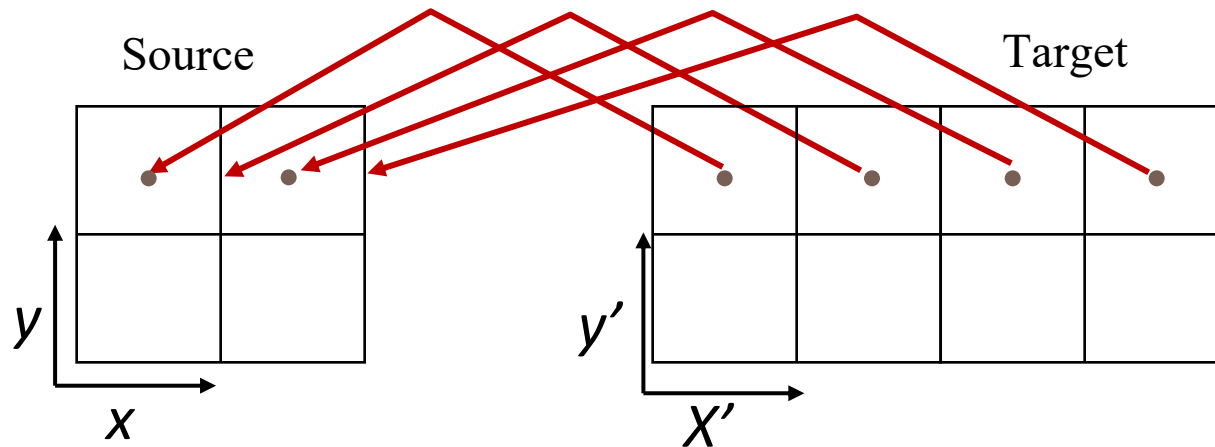
## Forward mapping:

$$x' = 2x ; y' = y$$



## Backward mapping:

$$x = x' / 2 ; y = y'$$



# Interpolation

- What happens when a mapping function calculates a fractional pixel address?



- **Interpolation:** generates a new pixel by analyzing the surrounding pixels.

# Interpolation

- Creates new samples from existing image samples.
- Increases the resolution of an image by adding virtual samples at all points within the image boundary.
- Common interpolation techniques:
  - ▣ zero order – nearest neighbor
  - ▣ first order – (bilinear)
  - ▣ second order – (bicubic)



# Nearest Neighbor

- Nearest neighbor interpolation.
  - ▢ Assume that a destination location  $(x' \ y')$  maps backward to source location  $(x, y)$ .
  - ▢ The source pixel nearest location  $(x, y)$  is located at  $(\text{round}(x), \text{round}(y))$  and the source pixel at that image is then carried over as the value of the destination.
- Nearest neighbor interpolation is computationally efficient but of generally poor quality, producing images with jagged edges and high graininess.

# Nearest Neighbor Interpolation

- The assign value is taken from the pixel closest to the generated location:

$$\begin{aligned} I'(x', y') &= I(\text{round}(T_x^{-1}(x', y')), \text{round}(T_y^{-1}(x', y'))) \\ &= I(\text{round}(x), \text{round}(y)) \end{aligned}$$

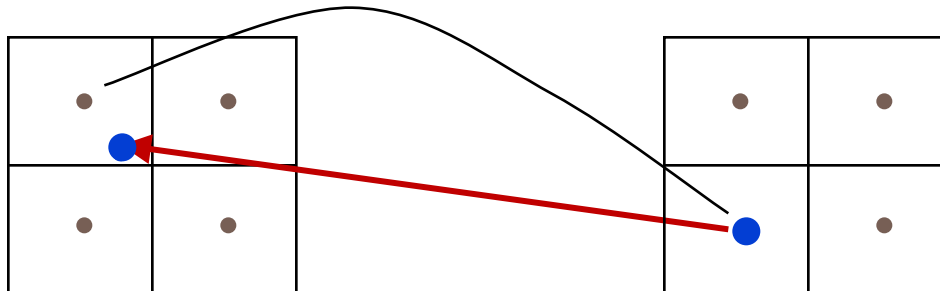
- Advantage:

- Fast

- Disadvantage:

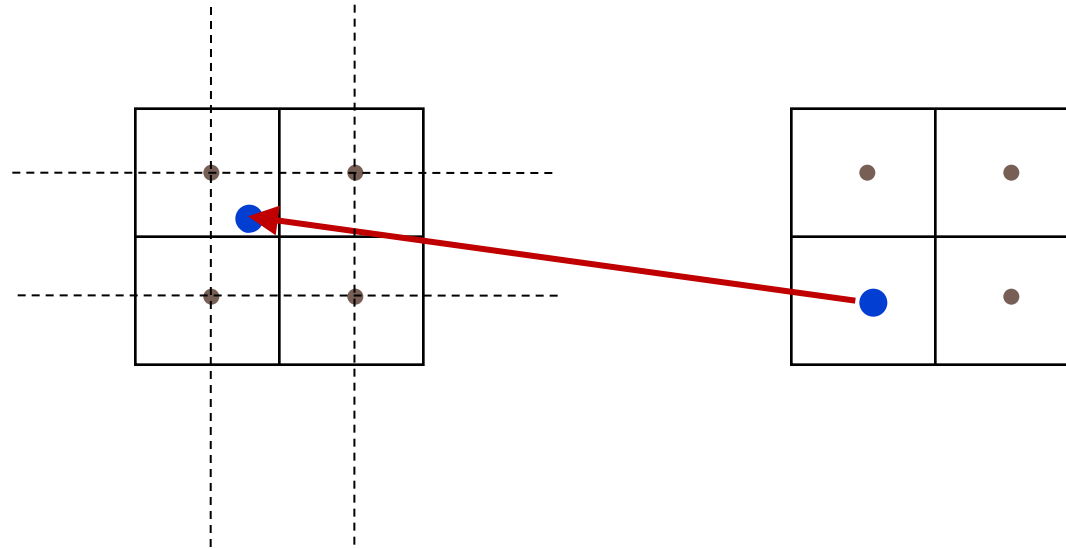
- Jagged results

- Discontinuous results



# Bilinear Interpolation

- The assign value is a weighted sum of the four nearest pixels.
- Each weight is proportional to the distance from each existing pixel.

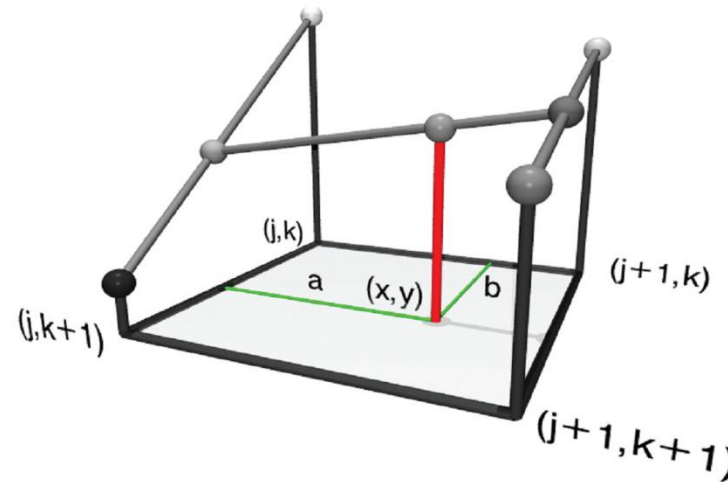


# Bilinear Interpolation

- Bilinear interpolation assumes that the continuous image is a linear function of spatial location.
- Linear, or first order, interpolation combines the four points surrounding location  $(x,y)$
- $(x, y)$  is the backward mapped coordinate that is surrounded by the four samples at  $(j,k)$   $(j, k+1)$ ,  $(j+1, k)$ , and  $(j+1, k+1)$

# Bilinear Interpolation

- Bilinear interpolation is a weighted average where pixels closer to the backward mapped coordinate are weighted proportionally heavier than those pixels further away.
- Bilinear interpolation acts like something of a rigid mechanical system
  - ▢ Two rods vertically connect the four samples surrounding the backward mapped coordinate.
  - ▢ A third rod is connected horizontally which is allowed to slide vertically up and down the fixture.
  - ▢ A ball is attached to this horizontal rod and is allowed to slide freely back and forth across the central rod.
  - ▢ The height of the ball determines the interpolated sample value wherever the ball is located.
  - ▢ In this way it should be clear that all four sample values.



the four corner posts have implicit, or interpolated,

Figure 7.5. Bilinear interpolation.

# Bilinear Interpolation

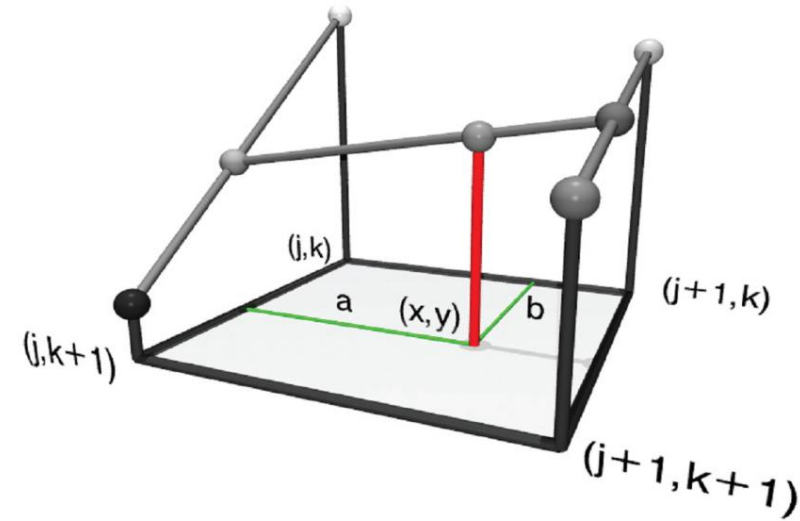
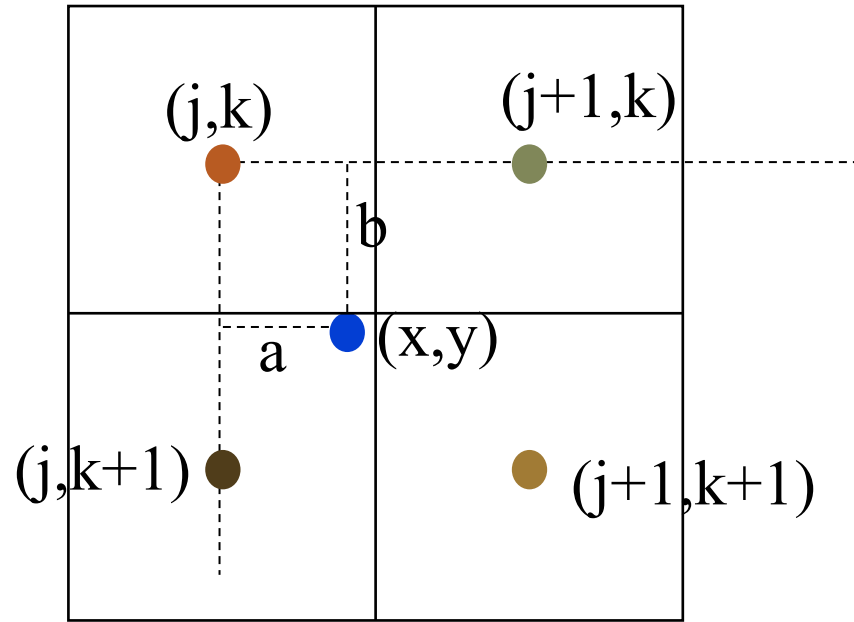


Figure 7.5. Bilinear interpolation.

$$j = \lfloor x \rfloor \quad k = \lfloor y \rfloor \quad a = x - j \quad b = y - k$$

$$I'(x', y') = [1 - b, b] \cdot \begin{bmatrix} I(j, k) & I(j + 1, k) \\ I(j, k + 1) & I(j + 1, k + 1) \end{bmatrix} \cdot \begin{bmatrix} 1 - a \\ a \end{bmatrix}$$

# Example

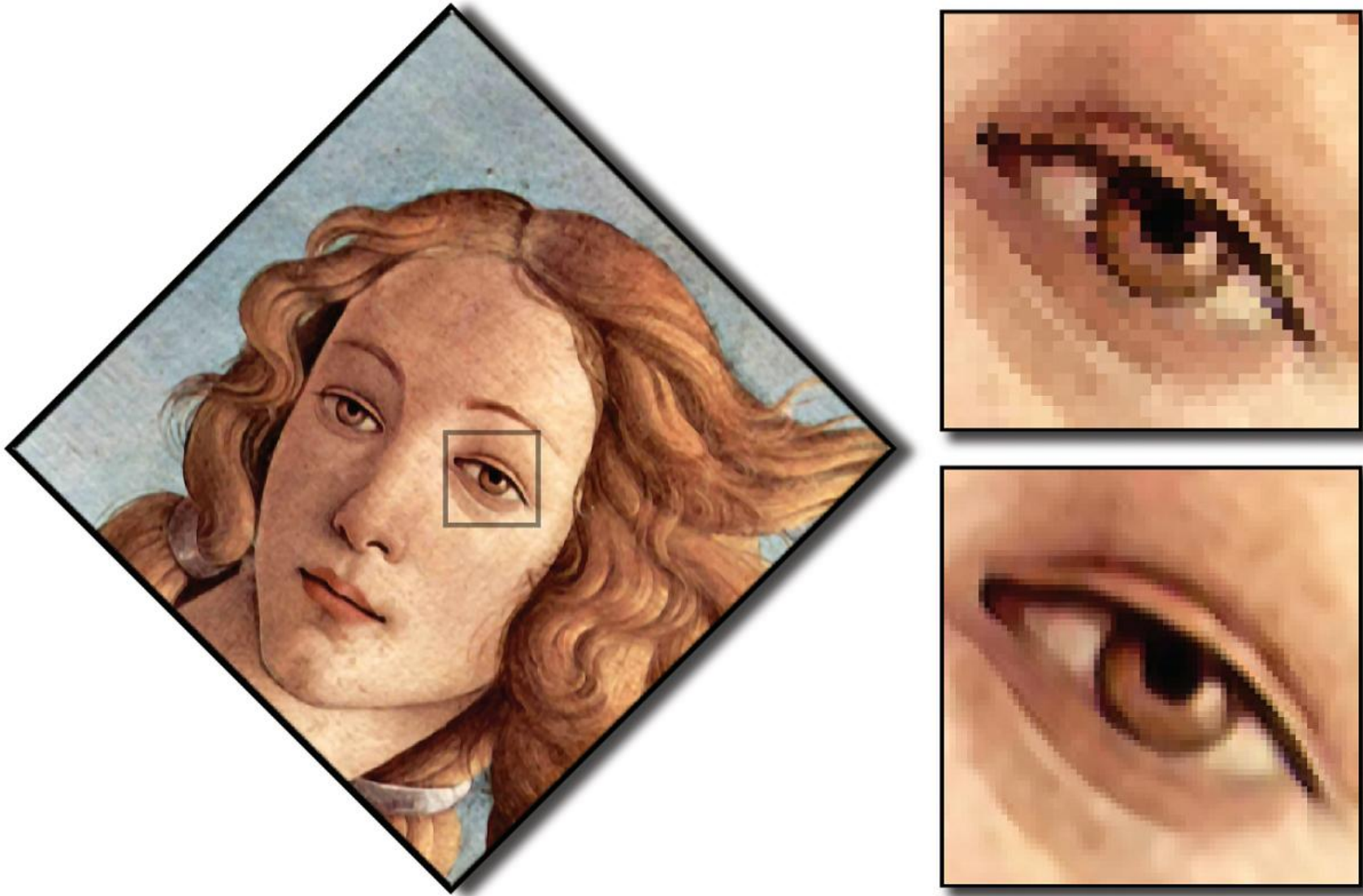


Figure 7.6. Interpolation example.



# Example



Original Image



Nearest N.  
Interpolatio

# Example



Original Image



Bilinear  
Interpolatio



Original Image



Nearest N.  
Interpolatio



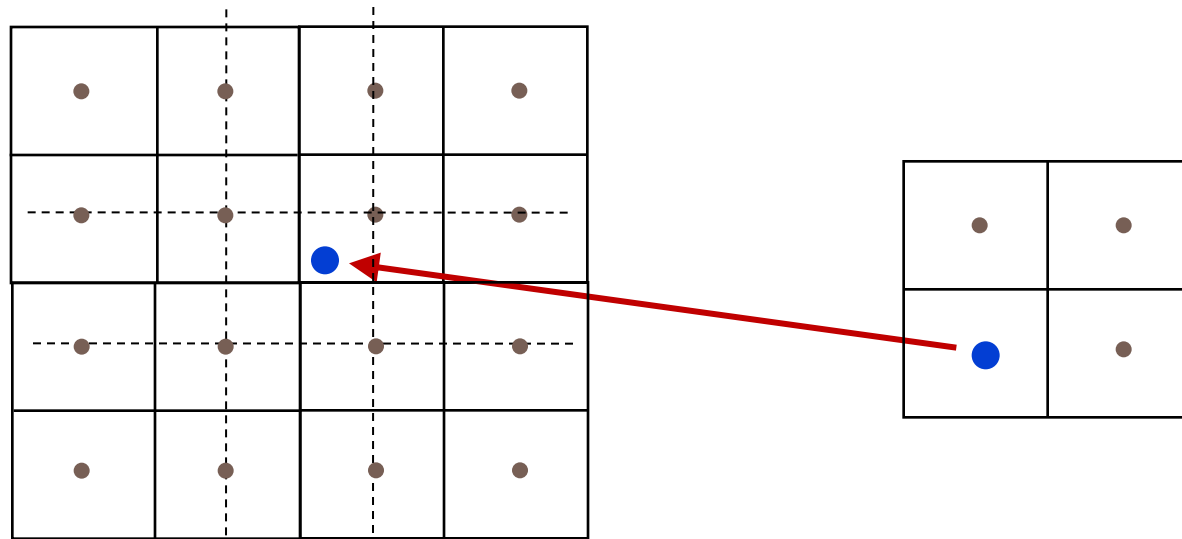
Original Image



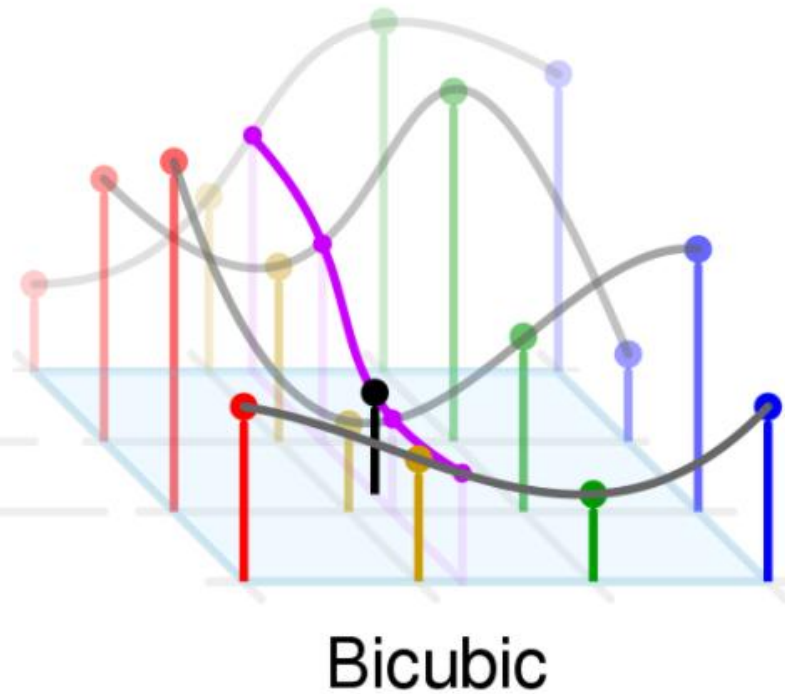
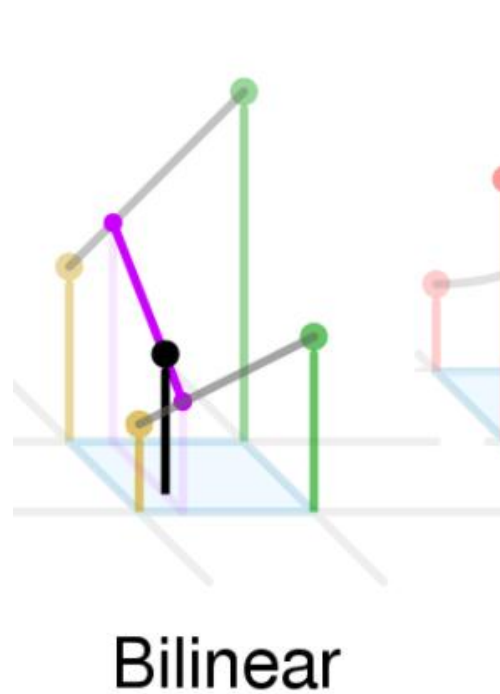
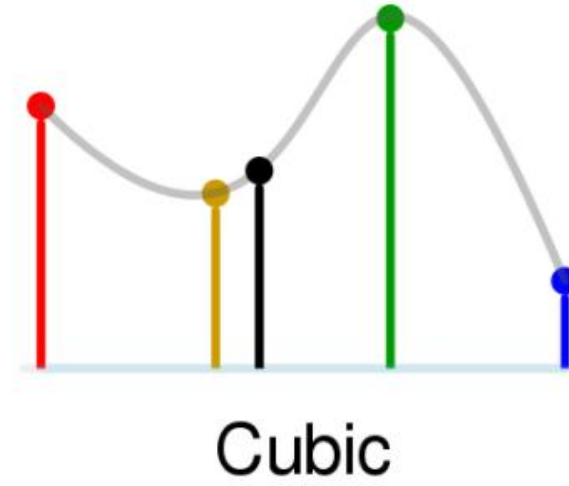
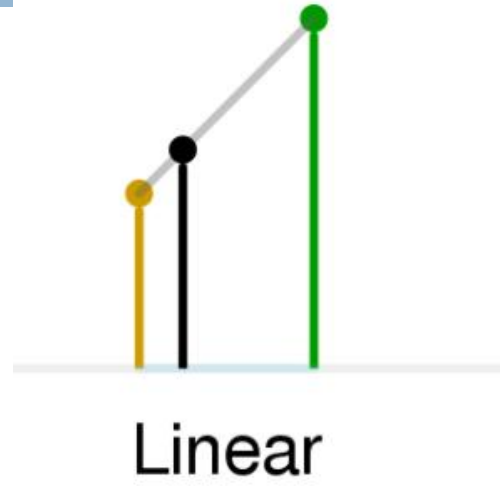
Bilinear  
Interpolatio  
n

# Bicubic Interpolation

- In bi-cubic interpolation, the destination sample is a non-linear weighted sum of the 4x4 nearest pixels of the reverse mapped location.
- Properties of second order interpolation
  - everywhere continuous
  - more computational effort required



# Bicubic Interpolation

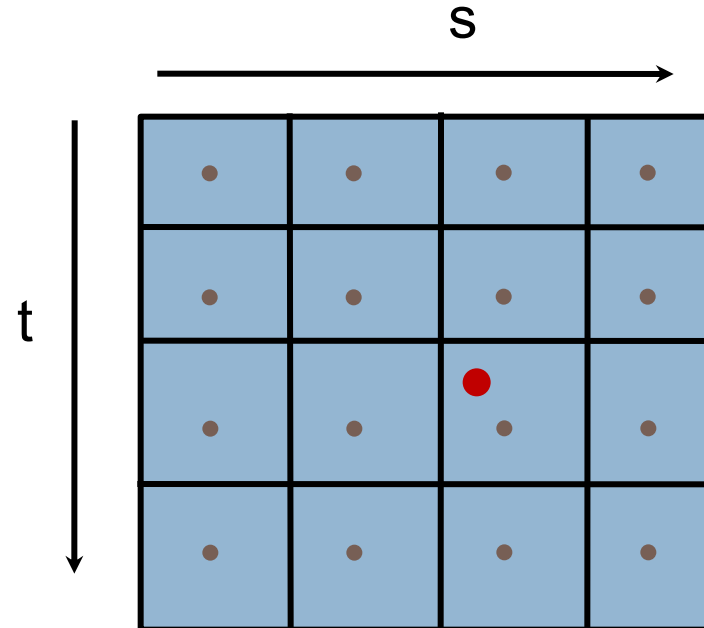


## How can we find the right coefficients?

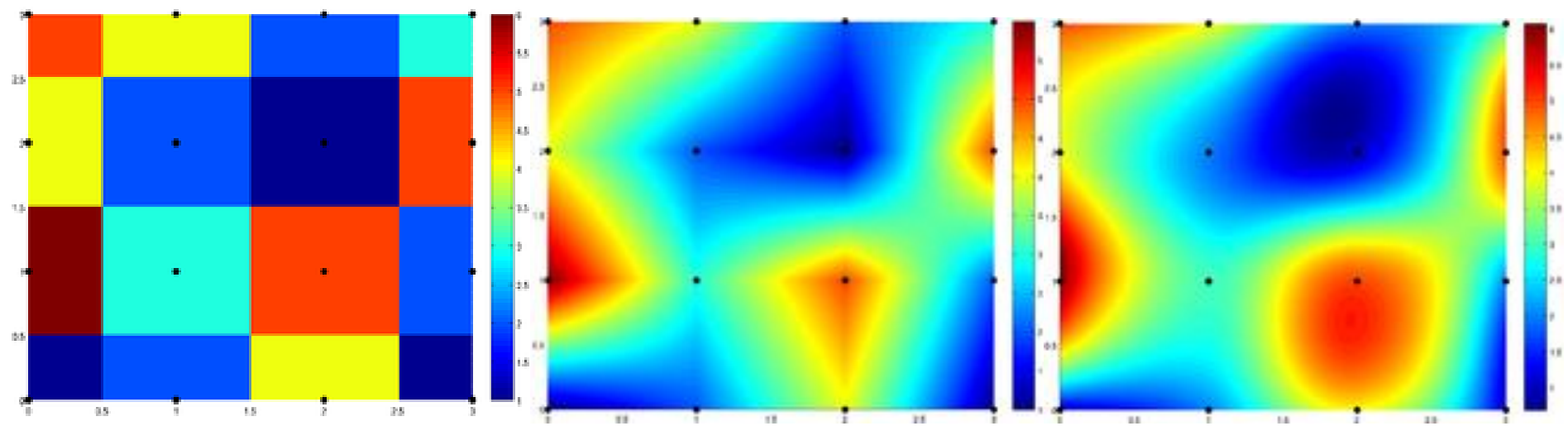
- Denote the pixel values  $V_{pq}$   $\{p,q=0..3\}$
- The unknown coefficients are  $a_{ij}$   $\{i,j=0..3\}$

$$v_{pq} = \sum_{i,j=0}^3 a_{ij} s^i t^j \quad \text{for } p, q = \{0..3\}, \quad s, t \in [-1, 2]$$

- We have a linear system of 16 equations with 16 coefficients.
- The pixel's boundaries are  $C_1$  continuous (continuous derivatives across boundaries).







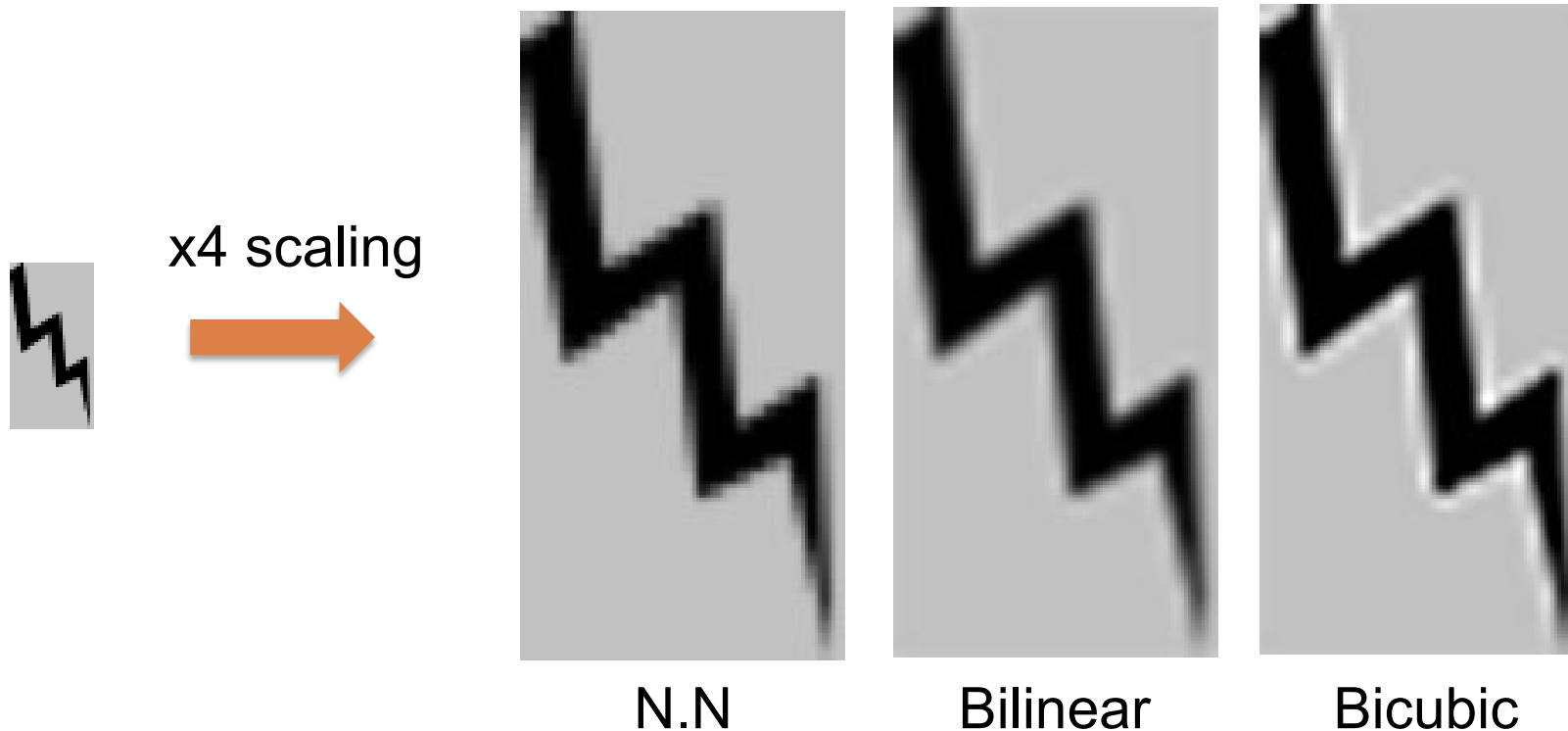
N.N

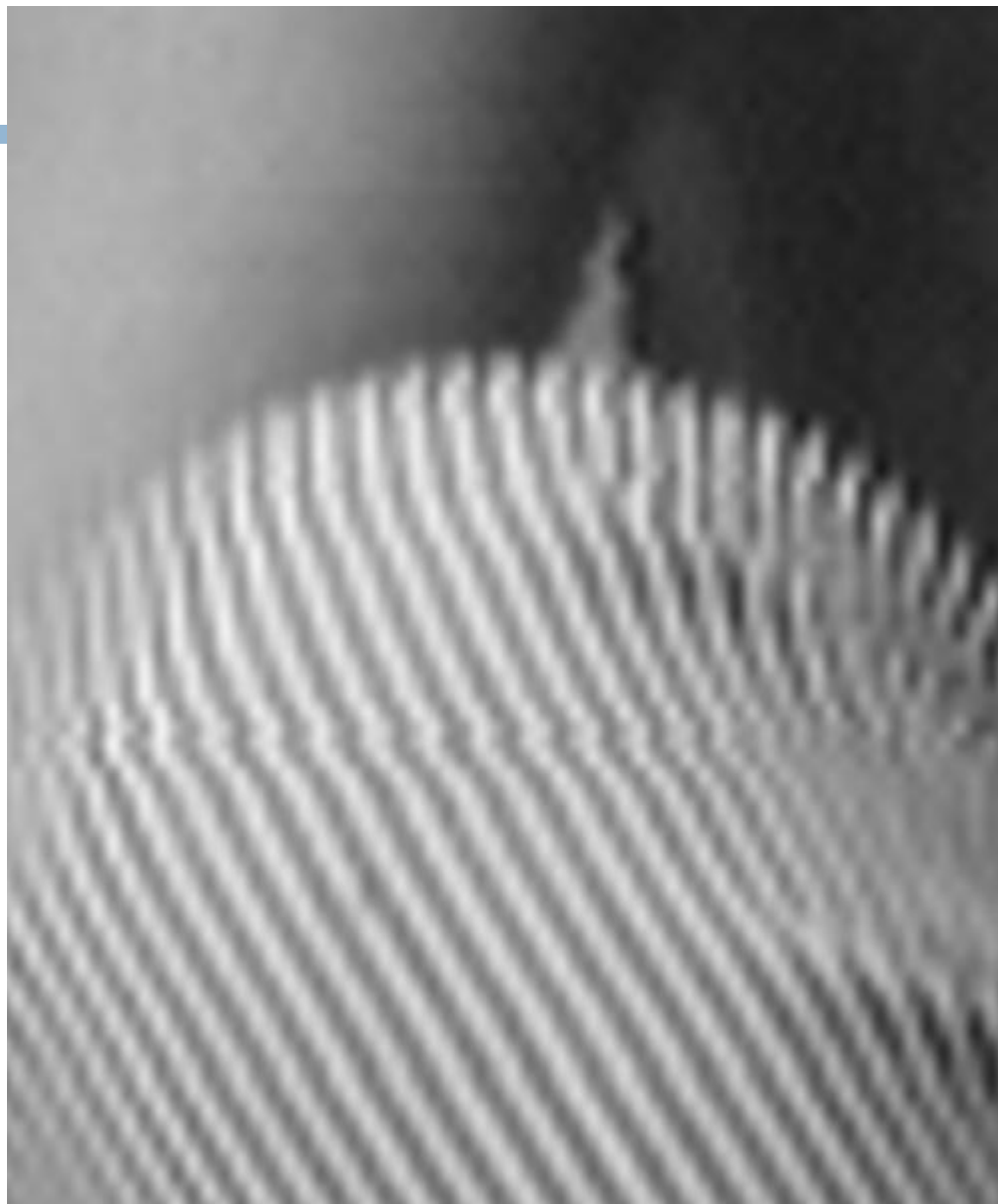
Bilinear

Bicubic

# Interpolation

- Good interpolation techniques attempt to find an optimal balance between three undesirable artifacts: edge halos, blurring and aliasing.





Bicubic

# Applying the Transformation

```
T = ..... % 2x2 transformation matrix
[r,c] = size(img)

% create array of destination x,y coordinates
[X,Y]=meshgrid(1:c,1:r);

% calculate source coordinates
sourceCoor = inv(T) * [X(:) Y(:)]' ;

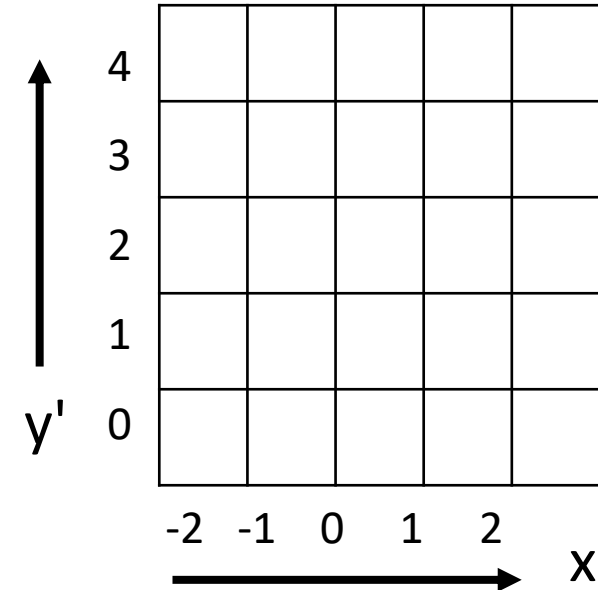
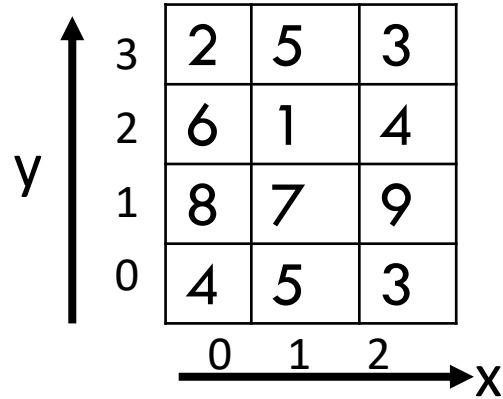
% calculate nearest neighbor interpolation
Xs = round(sourceCoor(1,:));
Ys = round(sourceCoor(2,:));

indx=find(Xs<1 | Xs>r); %out of range pixels
Xs(indx)=1; Ys(indx)=1;

indy=find(Ys<1 | Ys>c); %out of range pixels
Xs(indy)=1; Ys(indy)=1;

% calculate new image
newImage = img((Xs-1).*r+Ys);
newImage(indx)=0; newImage(indy)=0;
newImage = reshape(newImage,r,c);
```

# Numerical Example: 30° Rotation



$$x' = 0\cos 30^\circ - 0\sin 30^\circ = 0$$

$$y' = 0\sin 30^\circ + 0\cos 30^\circ = 0$$

Min  
y'

$$x' = 0\cos 30^\circ - 3\sin 30^\circ = -1.5 = -2$$

$$y' = 0\sin 30^\circ + 3\cos 30^\circ = 2.6 = 3$$

Min  
x'

$$x' = 2\cos 30^\circ - 0\sin 30^\circ = 1.73 = 2$$

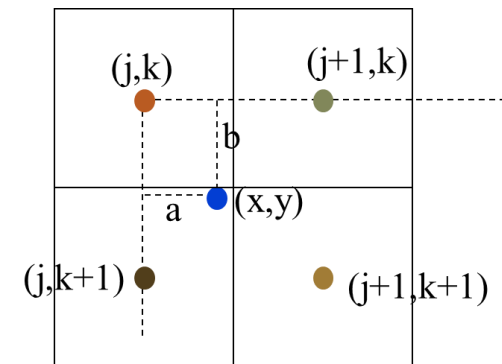
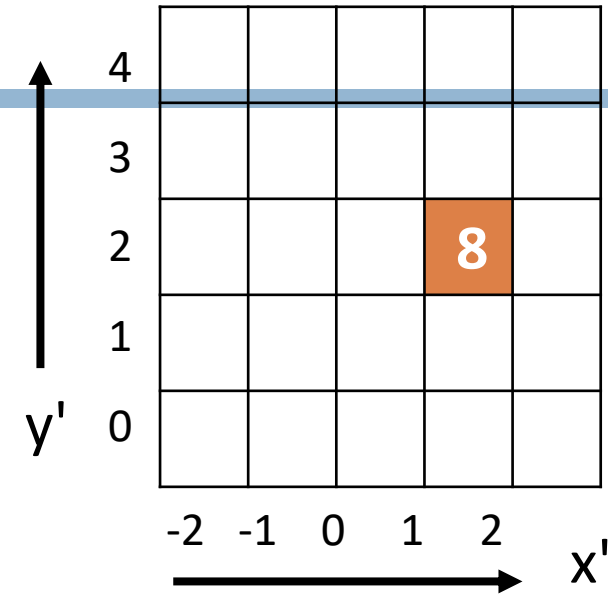
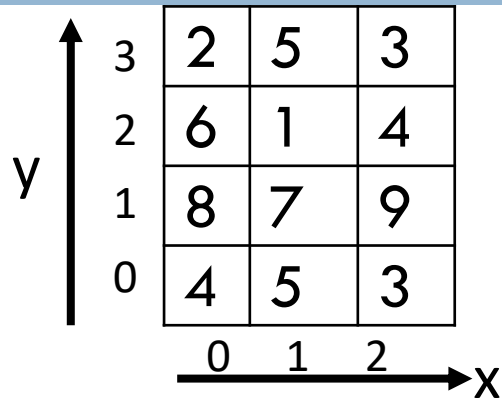
$$y' = 2\sin 30^\circ + 0\cos 30^\circ = 1$$

Max x'

$$x' = 2\cos 30^\circ - 3\sin 30^\circ = 0.23 = 0$$

$$y' = 2\sin 30^\circ + 3\cos 30^\circ = 3.6 = 4$$

Max y'



$$x = 1\cos 30 + 2\sin 30 = 1.87$$

$$y = -1\sin 30 + 2\cos 30 = 1.23$$

$$J=1, k=1, a=0.87, b=0.23$$

$$\begin{aligned} I'(x', y') &= \begin{bmatrix} .77 & .23 \end{bmatrix} \begin{bmatrix} 7 & 9 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} .13 \\ .87 \end{bmatrix} \\ &= \begin{bmatrix} .77 & .23 \end{bmatrix} \begin{bmatrix} 8.74 \\ 3.61 \end{bmatrix} = 7.56 = 8 \end{aligned}$$

$$j = \lfloor x \rfloor \quad k = \lfloor y \rfloor \quad a = x - j \quad b = y - k$$

$$I'(x', y') = [1 - b, b] \cdot \begin{bmatrix} I(j, k) & I(j + 1, k) \\ I(j, k + 1) & I(j + 1, k + 1) \end{bmatrix} \cdot \begin{bmatrix} 1 - a \\ a \end{bmatrix}$$

3	2	5	3
2	6	1	4
1	8	7	9
0	4	5	3

0 1 2  $x$

$y$

4	0	0	2	0	0
3	1	4	3	2	0
2	1	5	3	8	2
1	0	5	7	6	2
0	0	1	4	2	0

-2 -1 0 1 2  $x'$

$y'$

Final  
0 padding an bilinear intp.



## Computation in excel

[illegible]

excel

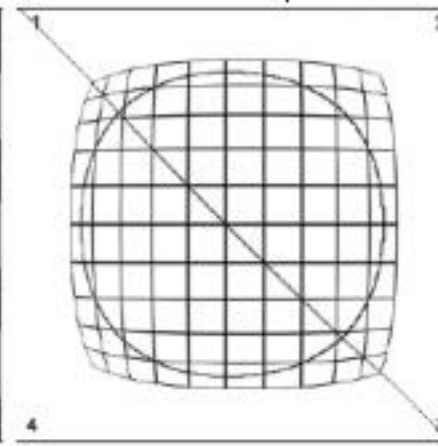
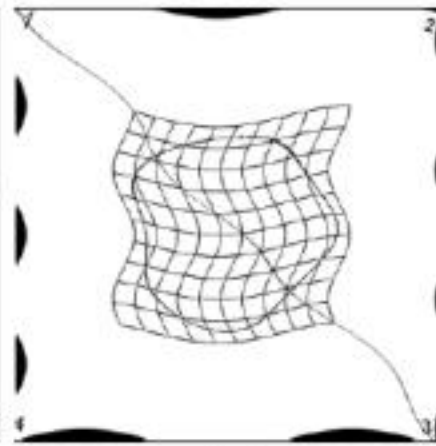
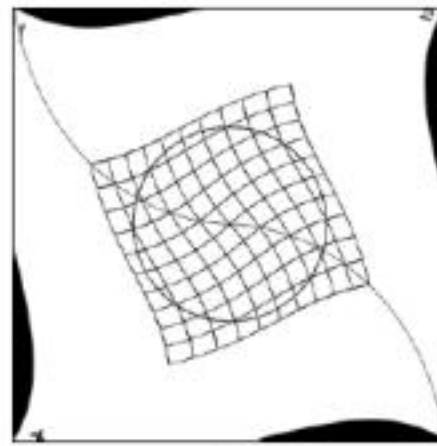
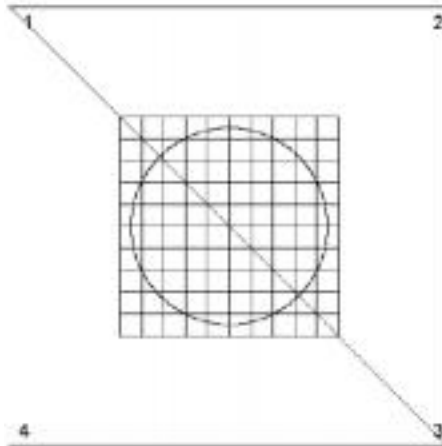
excel

# Non Linear 2D Transformations

- Non linear transformations do not necessarily preserve straight lines.
- Methods:
  - Piecewise linear transformations
  - Non linear parametric mapping



# Non Linear 2D Transformations



(a)

(b)

(c)



Original



(d)

Twirl



(e)

Ripple



(f)

Spherical

# Non Linear 2D Transformations

## □ Twirl:

- ❑ Rotate image by **angle  $\alpha$**  at center or **anchor point**  $(x_c, y_c)$
- ❑ Increasingly rotate image as radial distance  $r$  from center increases (up to  $r_{max}$ )
- ❑ Image unchanged outside radial distance  $r_{max}$

$$T_x^{-1} : x = \begin{cases} x_c + r \cdot \cos(\beta) & \text{for } r \leq r_{max} \\ x' & \text{for } r > r_{max}, \end{cases}$$

$$T_y^{-1} : y = \begin{cases} y_c + r \cdot \sin(\beta) & \text{for } r \leq r_{max} \\ y' & \text{for } r > r_{max}, \end{cases}$$

$$d_x = x' - x_c,$$

$$d_y = y' - y_c,$$

$$r = \sqrt{d_x^2 + d_y^2},$$

$$\beta = \text{Arctan}(d_y, d_x) + \alpha \cdot \left( \frac{r_{max} - r}{r_{max}} \right)$$

# Non Linear 2D Transformations

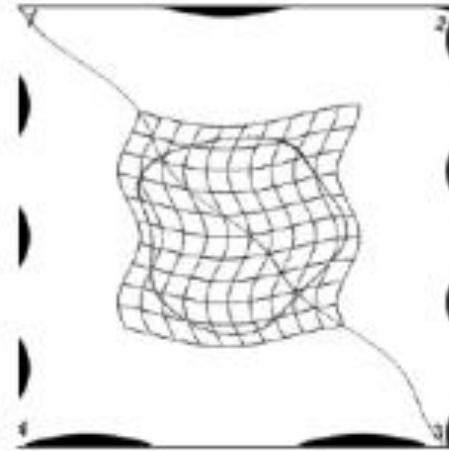
## Ripple

- Ripple causes wavelike displacement of image along both the x and y directions

$$T_x^{-1} : x = x' + a_x \cdot \sin\left(\frac{2\pi \cdot y'}{\tau_x}\right),$$

$$T_y^{-1} : y = y' + a_y \cdot \sin\left(\frac{2\pi \cdot x'}{\tau_y}\right).$$

- Sample values for parameters (in pixels) are
  - $\tau_x = 120$
  - $\tau_y = 250$
  - $a_x = 10$
  - $a_y = 15$



(b)



(e)



# Non Linear 2D Transformations

## Spherical Transformation

- Imitates viewing image through a lens placed over image
- Lens parameters: center  $(x_c, y_c)$ , lens radius  $r_{max}$  and refraction index  $\rho$
- Sample values  $\rho = 1.8$  and  $r_{max} = \text{half image width}$

$$T_x^{-1}: x = x' - \begin{cases} z \cdot \tan(\beta_x) & \text{for } r \leq r_{max} \\ 0 & \text{for } r > r_{max}, \end{cases}$$

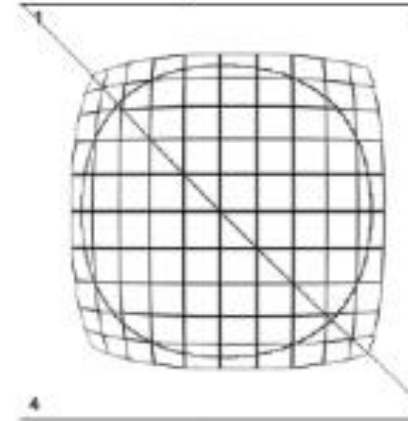
$$T_y^{-1}: y = y' - \begin{cases} z \cdot \tan(\beta_y) & \text{for } r \leq r_{max} \\ 0 & \text{for } r > r_{max}, \end{cases}$$

$$d_x = x' - x_c, \quad r = \sqrt{d_x^2 + d_y^2},$$

$$d_y = y' - y_c, \quad z = \sqrt{r_{max}^2 - r^2},$$

$$\beta_x = \left(1 - \frac{1}{\rho}\right) \cdot \sin^{-1}\left(\frac{d_x}{\sqrt{d_x^2 + z^2}}\right),$$

$$\beta_y = \left(1 - \frac{1}{\rho}\right) \cdot \sin^{-1}\left(\frac{d_y}{\sqrt{d_y^2 + z^2}}\right).$$



(c)



(f)

# Non Linear 2D Transformations

**Radial wave transformation:** This transformation simulates an omni-directional wave which originates from a fixed center point  $\mathbf{x}_c$  (see Fig. 21.11(b)). The inverse transformation (applied to a target image point  $\mathbf{x}' = (x', y')$ ) is

$$T^{-1} : \mathbf{x} = \begin{cases} \mathbf{x}_c & \text{for } r = 0, \\ \mathbf{x}_c + \frac{r+\delta}{r} \cdot (\mathbf{x}' - \mathbf{x}_c) & \text{for } r > 0, \end{cases} \quad (21.74)$$

with  $r = \|\mathbf{x}' - \mathbf{x}_c\|$  and  $\delta = a \cdot \sin(2\pi r/\tau)$ . Parameter  $a$  specifies the *amplitude* (strength) of the distortion and  $\tau$  is the *period* (width) of the radial wave (in pixel units).



(b) Radial wave ( $a = 10.0$ ,  $\tau = 38$ )



# Non Linear 2D Transformations

**Clover** transformation: This transformation distorts the image in the form of a  $N$ -leafed clover shape (see Fig. 21.11(c)). The associated inverse transformation is the same as in Eqn. (21.74) but uses

$$\delta = a \cdot r \cdot \cos(N \cdot \alpha), \quad \text{with } \alpha = \angle(\mathbf{x}' - \mathbf{x}_c) \quad (21.75)$$

instead. Again  $r = \|\mathbf{x}' - \mathbf{x}_c\|$  is the radius of the target image point  $\mathbf{x}'$  from the designated center point  $\mathbf{x}_c$ . Parameter  $a$  specifies the amplitude of the distortion and  $N$  is the number of radial “leaves”.



(c) Clover ( $a = 0.2$ ,  $N = 8$ )

# Non Linear 2D Transformations

**Spiral** transformation: This transformation (see Fig. 21.11(d)) is similar to the *twirl* transformation in Eqns. (21.63)–(21.64), defined by the inverse transformation

$$T^{-1}: \mathbf{x} = \mathbf{x}_c + r \cdot \begin{pmatrix} \cos(\beta) \\ \sin(\beta) \end{pmatrix}, \quad (21.76)$$

with  $\beta = \angle(\mathbf{x}' - \mathbf{x}_c) + a \cdot r$  and  $r = \|\mathbf{x}' - \mathbf{x}_c\|$  denoting the distance from the target point  $\mathbf{x}'$  and the center point  $\mathbf{x}_c$ . The angle  $\beta$  increases linearly with  $r$ ; parameter  $a$  specifies the “velocity” of the spiral.



(d) Spiral ( $a = 0.01$ )



# Non Linear 2D Transformations

**Angular wave** transformation: This is another variant of the *twirl* transformation in Eqns. (21.63)–(21.64). Its inverse transformation is the same as for the spiral mapping in Eqn. (21.76), but in this case

$$\beta = \angle(\mathbf{x}' - \mathbf{x}_c) + a \cdot \sin\left(\frac{2\pi r}{\tau}\right). \quad (21.77)$$

Thus the angle  $\beta$  is modified by a sine function with amplitude  $a$  (see Fig. 21.11(e)).



(e) Angular wave ( $a = 0.1$ ,  $\tau = 38$ )

# Non Linear 2D Transformations

**Tapestry transformation:** In this case the inverse transformation of a target point  $\mathbf{x}' = (x', y')$  is

$$T^{-1}: \mathbf{x} = \mathbf{x}' + a \cdot \begin{pmatrix} \sin\left(\frac{2\pi}{\tau_x} \cdot (x' - x_c)\right) \\ \sin\left(\frac{2\pi}{\tau_y} \cdot (y' - y_c)\right) \end{pmatrix}, \quad (21.78)$$

again with the center point  $\mathbf{x}_c = (x_c, y_c)$ . Parameter  $a$  specifies the distortion's amplitude and  $\tau_x, \tau_y$  are the wavelengths (measured in pixel units) along the  $x$  and  $y$  axis, respectively (see [Fig. 21.11\(f\)](#)).



(f) Tapestry ( $a = 5.0$ ,  $\tau_x = \tau_y = 30$ )

# Polynomial Transformations

- It belongs to the class of nonlinear mappings.
- it can approximate wider group of transformations than for example the affine mappings.
- It can be defined as follows  $\mathbf{y} = \mathbf{W} \cdot \mathbf{P}(\mathbf{x})$

$\mathbf{x}$  denotes the input vector (a point),  $\mathbf{y}$  is an output vector,  
 $\mathbf{W}$  is a transformation matrix and  $\mathbf{P}$  denotes a polynomial on  $\mathbf{x}$ .

the second-order polynomial transformation for which  $\mathbf{W}$   
and  $\mathbf{P}$  can be stated as follows:

$$\mathbf{W} = \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} & W_{15} & W_{16} \\ W_{21} & W_{22} & W_{23} & W_{24} & W_{25} & W_{26} \end{bmatrix}_{2 \times 6}$$

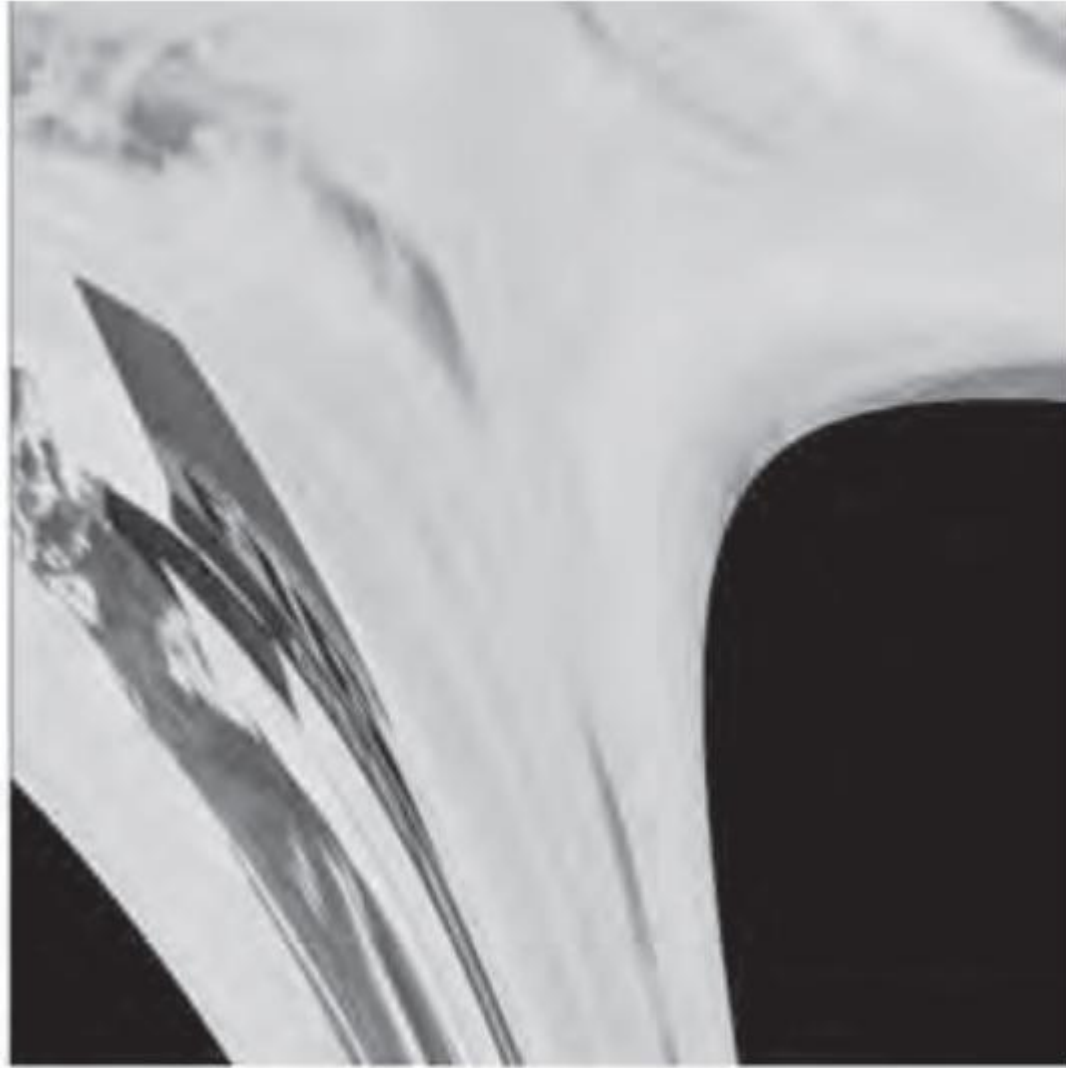
$$\mathbf{P}(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}_{6 \times 1} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

# Polynomial Transformations

- In contrast to the affine transformations – where elements of the transformation matrix can easily be found from given intuitive parameters, such as a rotation angle or scale
- The 12 parameters  $w_{ij}$  in **W** need to be determined for a mapping are usually unknown in analytical form.
- This can be achieved after manual (empirical) choice of the number of control points (at least six, since each point gives two equations) and their corresponding positions in the output image.
- Then the eqn is solved for W. For the more general case of more than six corresponding points this can be achieved by the least-squares method



# Polynomial Transformations - inverse warping



$W = [0, 1, 0, 0.001, -0.001, 0.001]$   
**83**  $[0, 0, 1, 0.001, -0.001, 0.001];$

$W = [0, 1, 0, 0, -0.001, 0]$   
 $[0, 0, 1, 0.001, -0.005, 0.001]$

- Digital Image Processing – Chapter 21
  - An Algorithmic Introduction Using Java
    - Wilhelm Burger • Mark J. Burge
- An Introduction to 3D Computer Vision Techniques and Algorithms – Chapter 12
  - Bogusław Cyganek, J. Paul Siebert
- Fundamentals of Digital Image Processing – Chapter 7
  - Chris Solomon, Toby Breckon
- Principles of Digital Image Processing, Core Algorithms– Chapter 10
  - Wilhelm Burger • Mark J. Burge