Computational Photography

* Study the basics of computation and its impact on the entire workflow of photography, from capturing, manipulating and collaborating on, and sharing photographs.

Georgia Tech College of Computing

Image Transformation

* Transformations applied to images

Lesson Objectives

1. Transform the image
2. Rigid Transformations: Translation, Rotations
3. Affine/Projective Transformation
4. Degrees of Freedom for different transformations

Image Transformations

image filtering:
change range of image

$$g(x) = T(f(x))$$

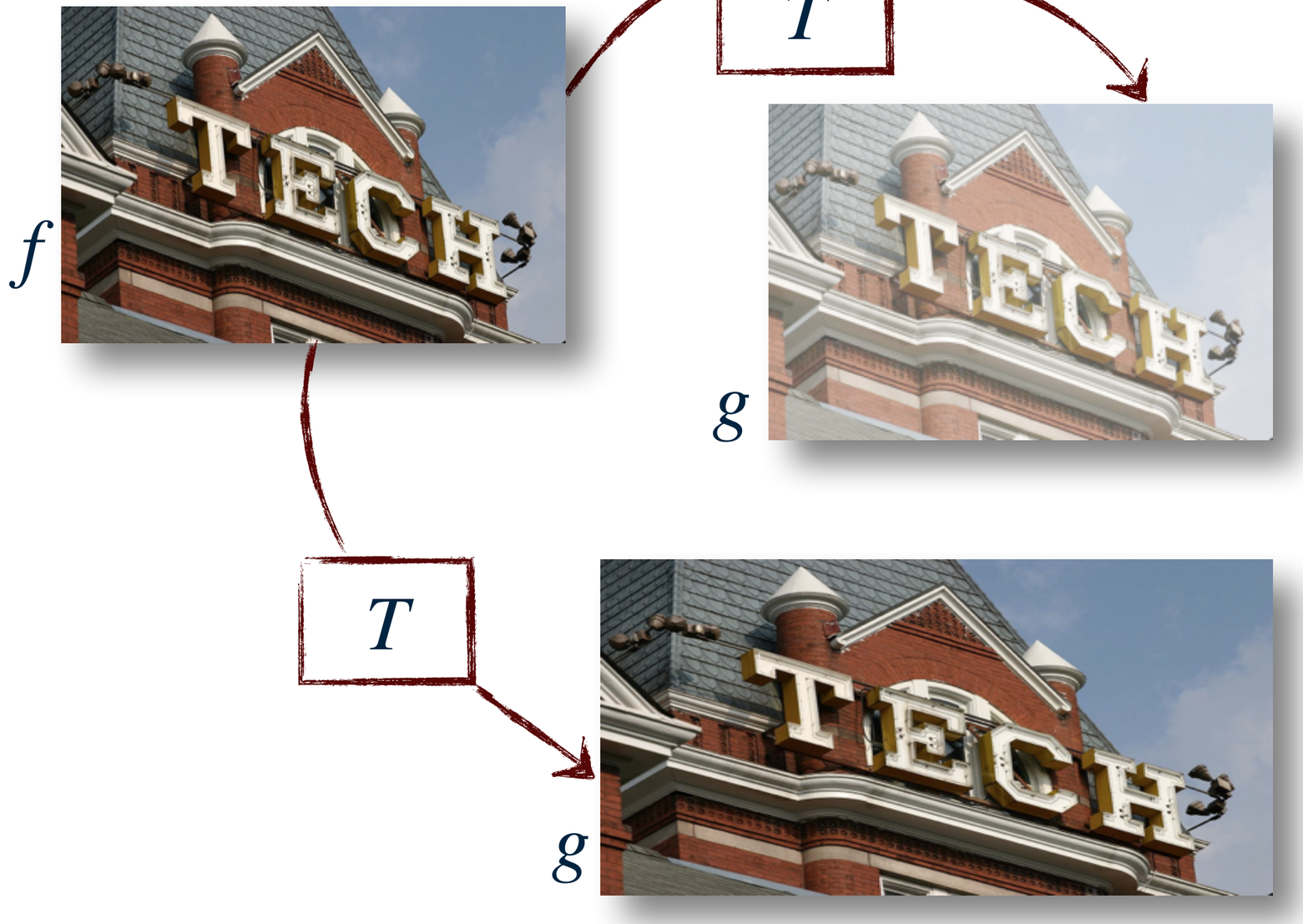


image warping:
change domain of image

$$g(x) = f(T(x))$$

Parametric Global Warping

translation

rotation

Parametric Global Warping

aspect

scale

Parametric Global Warping

affine

perspective

Parametric Global Warping

T

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Transformation T

$$p' = T(p)$$

- A global and parametric T
  - Same for any p
  - A few numbers (parameters)
- As a matrix transform

$$p' = \mathbf{M}p$$

Image Scaling (2D)

* Multiply each components by a scalar
* Uniform scaling: Scalar same for x, y
* Non-uniform: Not same

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Image Transformations

$$\mathbf{M} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

scale around (0,0)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

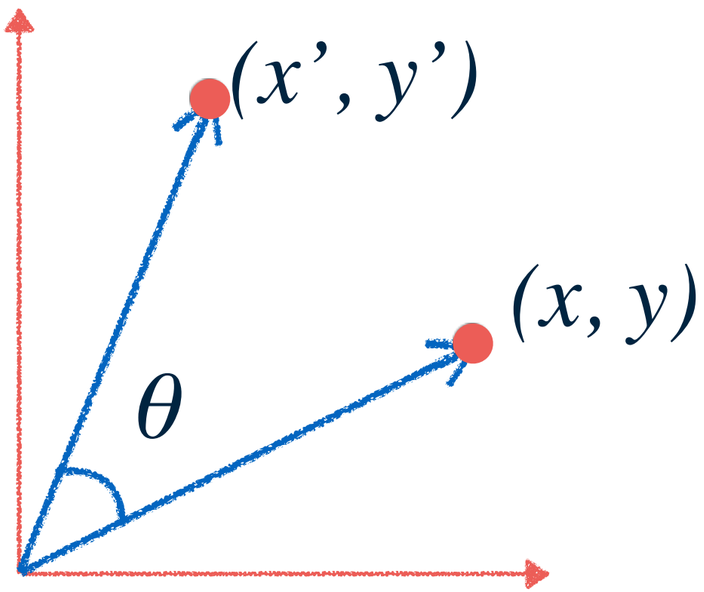
$$\mathbf{M} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

mirror over y-axis

$$\mathbf{M} = \begin{bmatrix} 1 & sh_x \\ sh_y & 1 \end{bmatrix}$$

shear

2D Rotation



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

$$
\mathbf{M} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}
$$

2D Linear Transformations

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} \Longleftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Only LINEAR COMBINATIONS

Linear transformations are combinations of

Scale,

Rotation,

Shear, and

Mirror

2D Linear Transformations

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} \Longleftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Only LINEAR COMBINATIONS

Properties of linear transformations:

Origin maps to origin

Lines map to lines

Parallel lines remain parallel

Ratios are preserved

2D Translation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} \Longleftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

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

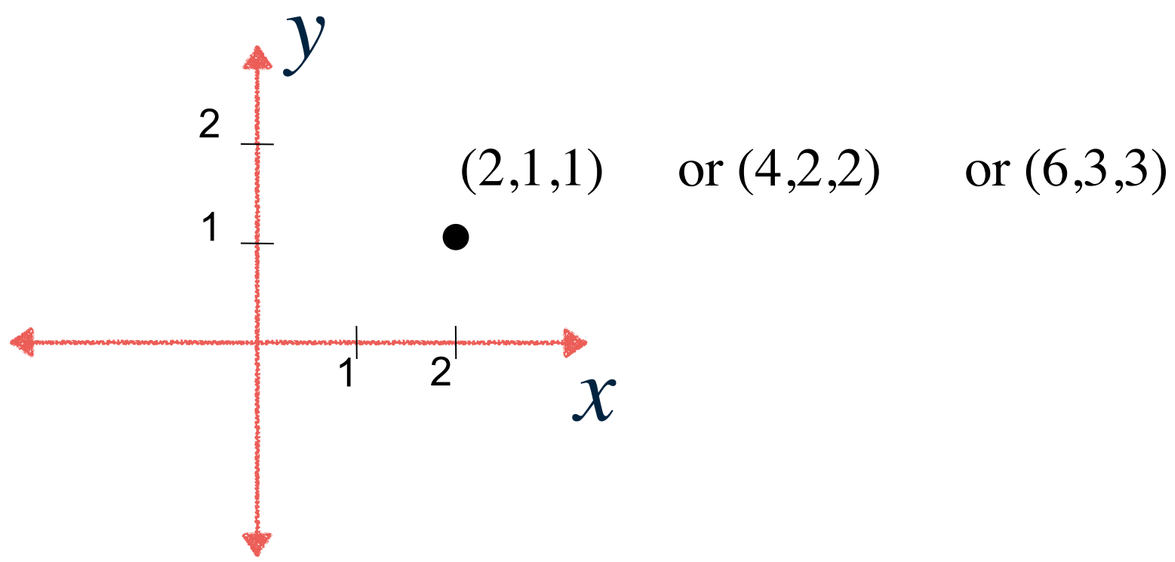
translation in x, y

Homogeneous Coordinates

* Represent coordinates in 2 dimensions with a 3-vector

$$\begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

* Add a 3rd coordinate to every 2D point
  * (x, y, w) ⇒ (x/w, y/w)
  * (x, y, 0) ⇒ infinity
  * (0, 0, 0) is not allowed

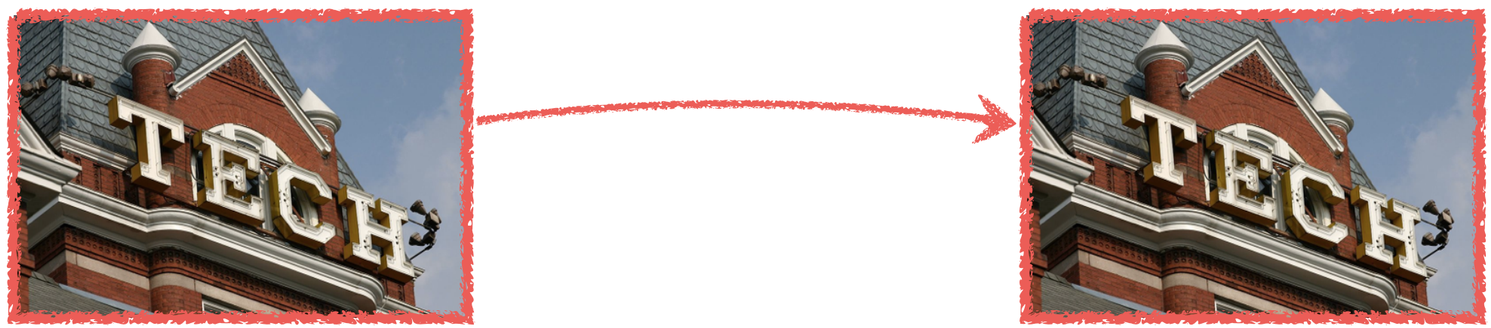


Basic 2D Transformation

$$p' = \mathbf{M}p$$

Translation

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

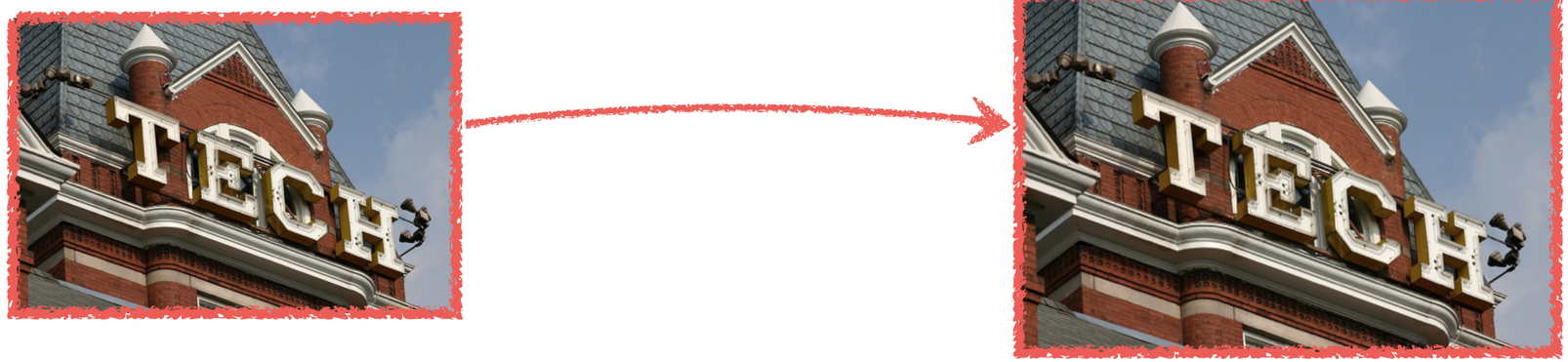


Basic 2D Transformation

$$p' = \mathbf{M}p$$

Scale

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

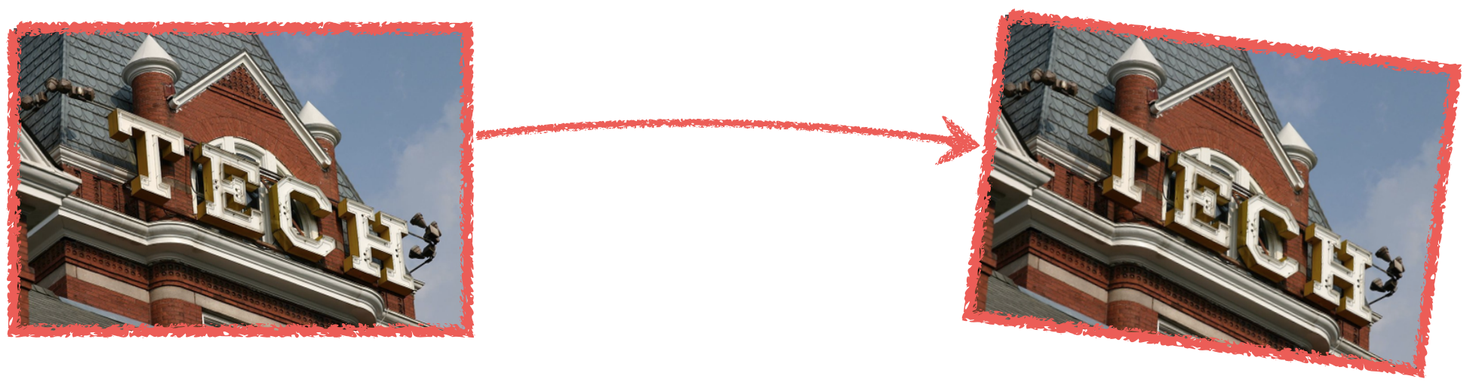


Basic 2D Transformation

$$p' = \mathbf{M}p$$

Rotation

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

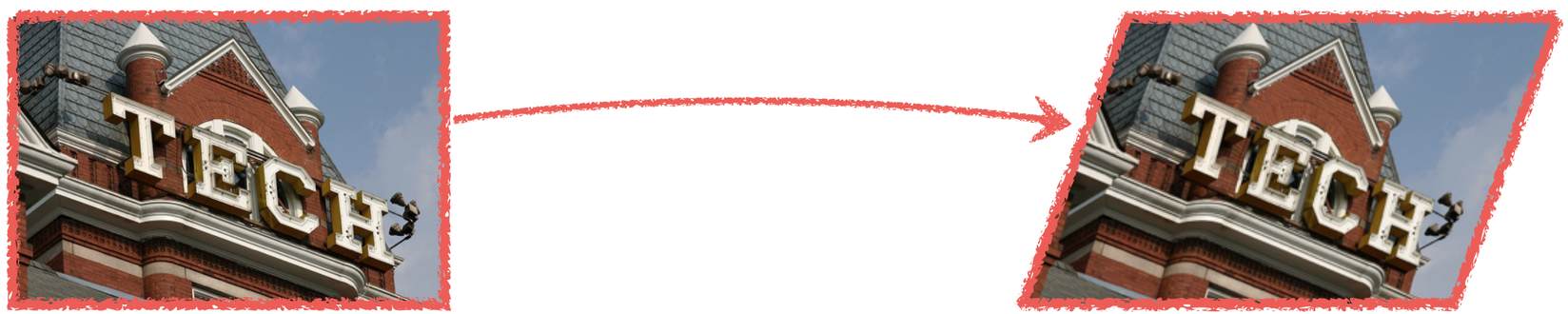


Basic 2D Transformation

$$p' = \mathbf{M}p$$

Shear

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



Basic 2D Transformation

Transformations can be combined

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \left\{ \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

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

Affine Transformations

- Combines linear transformations, and Translations
- Properties
- Origin does not necessarily map to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved

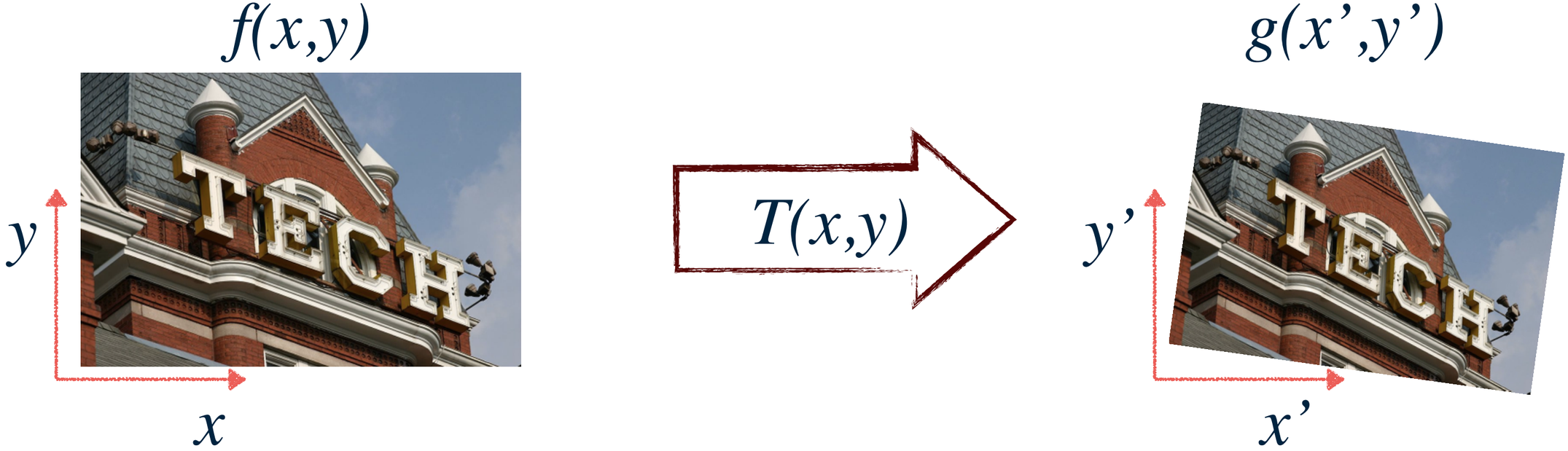
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Projective Transformations

* Combination of Affine transformations, and Projective warps
* Properties:
  * Origin does not necessarily map to origin
  * Lines map to lines
  * Parallel lines do not necessarily remain parallel
  * Ratios are not preserved

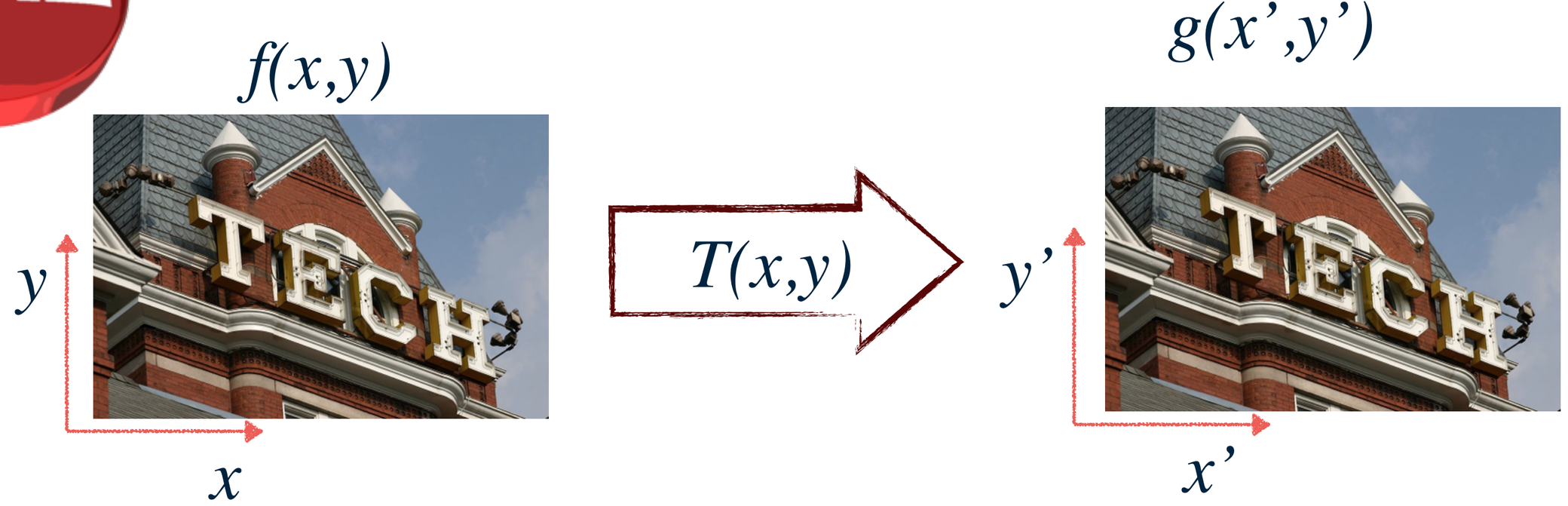
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Recovering Transformations



- What if we know f and g and want to recover the transform T?
- How many points correspondences would we need?
- How many Degrees of Freedom?

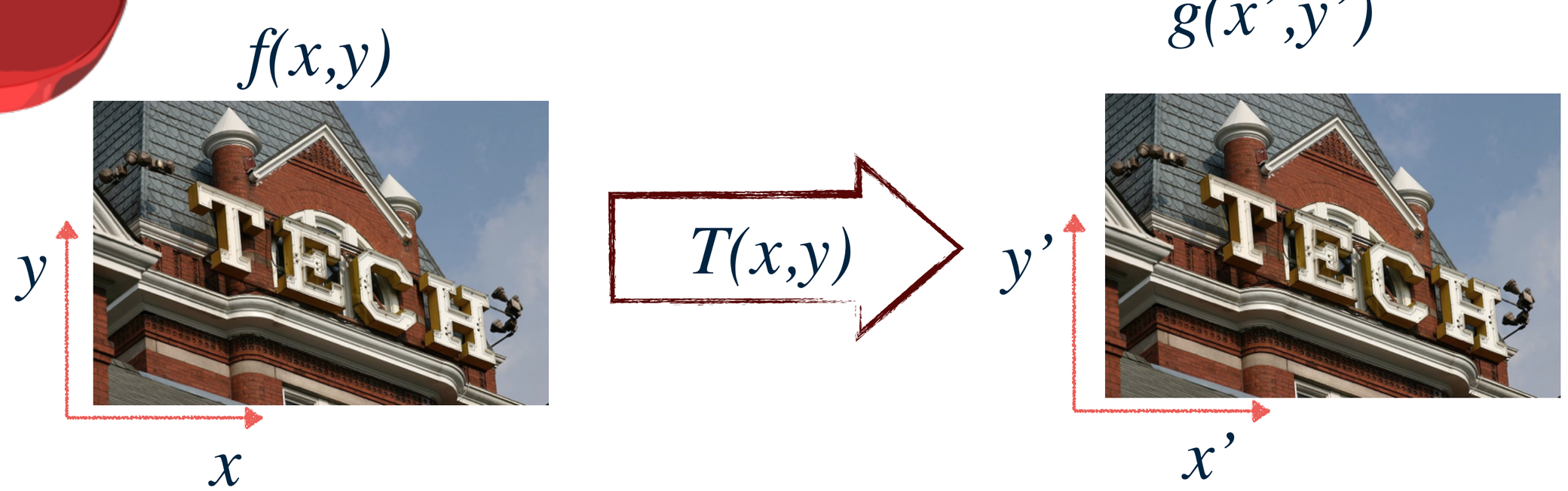
Translation



* How many correspondences needed for translation?

* How many Degrees of Freedom?

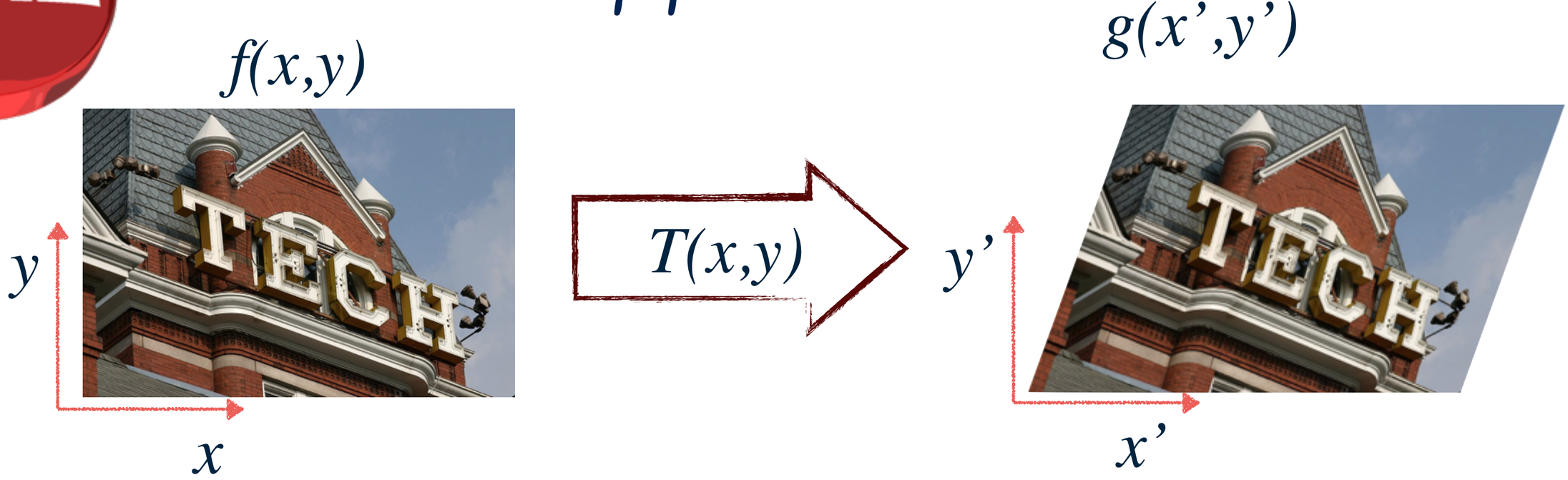
Rotation - Euclidean



* How many correspondences needed for translation?

* How many Degrees of Freedom?

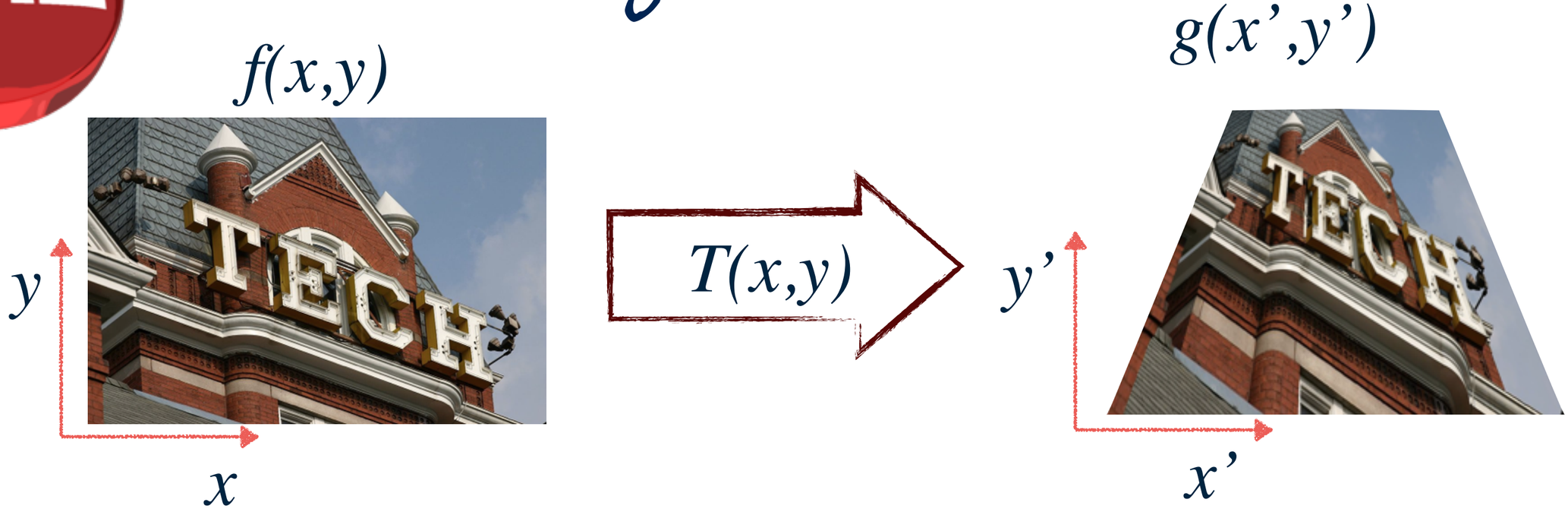
Affine



* How many correspondences needed for translation?

* How many Degrees of Freedom?

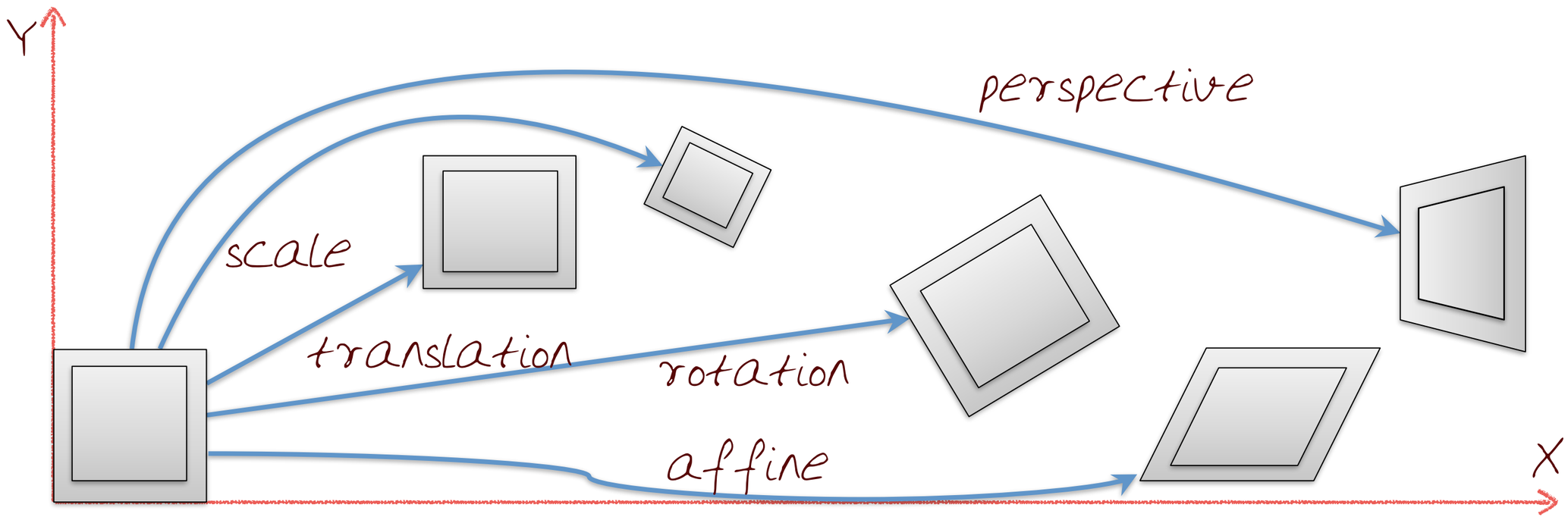
Projective



* How many correspondences needed for translation?

* How many Degrees of Freedom?

2D image transformations



Szeliski 2010

	Icon	DOF	3 X 3	Preserves
Translation		2		orientation
Euclidean		3		lengths
Similarity		4		angles
Affine		6		parallelism
Projective		8		straight lines

Szeliski 2010

DEMO

Translation

```
import cv2
import numpy as np

# Read image
img = cv2.imread('tech.png')
height, width = img.shape[:2]
cv2.imshow("Original", img)

# Translation
M_trans = np.float32(
    [[1, 0, 100],
     [0, 1,  50]])
print "Translation matrix:"
print M_trans
img_trans = cv2.warpAffine(img, M_trans, (width, height))  # note: last arg is size of output image
(width, height)
cv2.imshow("Translation", img_trans)
```

Rotation

```
import cv2
import numpy as np

# Read image
img = cv2.imread('tech.png')
height, width = img.shape[:2]
cv2.imshow("Original", img)

# Rotation around origin (0, 0)
M_rot = cv2.getRotationMatrix2D((0, 0), 45, 1)  # 45 deg, scale = 1
print "Rotation matrix (around origin):"
print M_rot
img_rot = cv2.warpAffine(img, M_rot, (width, height))
cv2.imshow("Rotation around origin", img_rot)

# Rotation around center (i.e. translation + rotation)
M_rot_center = cv2.getRotationMatrix2D((width / 2, height / 2), -45, 1)  # -45 deg, scale = 1
print "\nRotation matrix (around center):"
print M_rot_center
img_rot_center = cv2.warpAffine(img, M_rot_center, (width, height))
cv2.imshow("Rotation around center", img_rot_center)
```

Shear

```
import cv2
import numpy as np

# Read image
img = cv2.imread('tech.png')
height, width = img.shape[:2]
cv2.imshow("Original", img)

# Scale (resize)
img_scaled = cv2.resize(img, None, fx=1.5, fy=1.5, interpolation=cv2.INTER_CUBIC)
# specify scaling factors OR: img_scaled = cv2.resize(img, (int(1.5 * width), int(1.5 * height)),
interpolation=cv2.INTER_CUBIC)  # specify target size
cv2.imshow("Scale", img_scaled)

# Shear or skew (horizontal only)
M_shear = np.float32(
    [[1, 0.5, 0],
     [0,   1, 0]])
print "Shear matrix:"
print M_shear
img_shear = cv2.warpAffine(img, M_shear, (int(width + 0.5 * height), height))  # output image needs
to be wider to accomodate stretched image
cv2.imshow("Shear", img_shear)
```

Affine

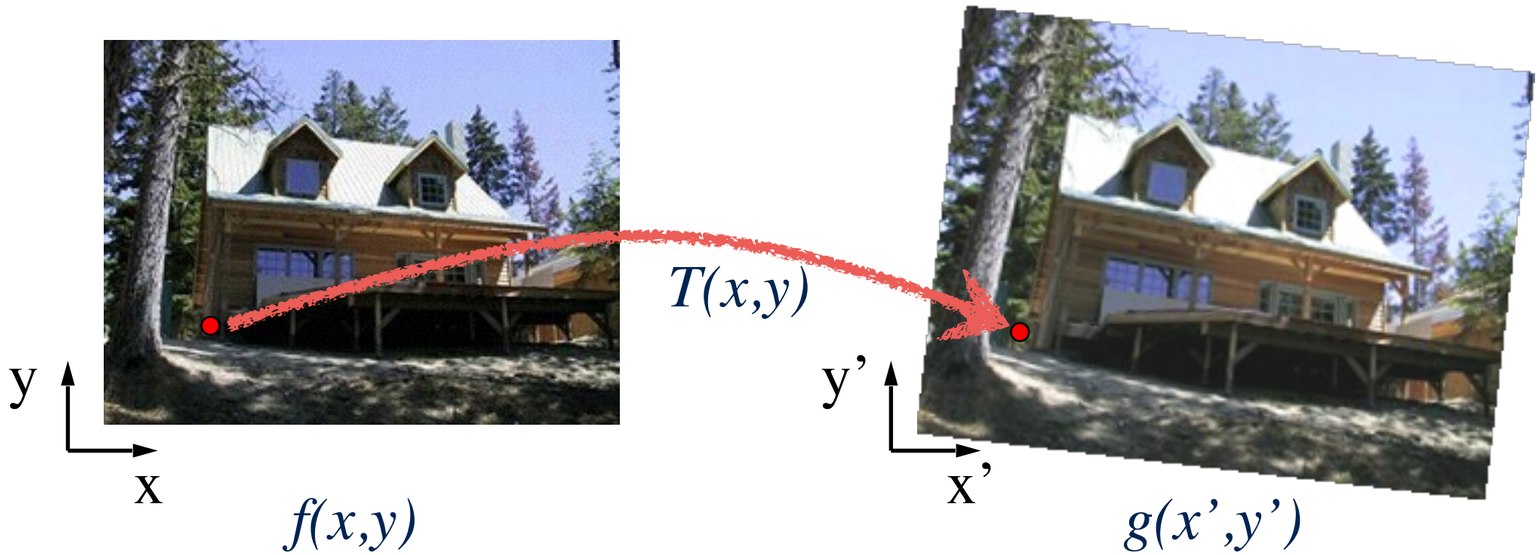
```
import cv2
import numpy as np

# Read image
img = cv2.imread('tech.png')
height, width = img.shape[:2]
cv2.imshow("Original", img)

# Affine transform (may include translation, rotation, scale)
pts1 = np.float32([[50, 50], [200, 50], [50, 200]])  # original point locations
pts2 = np.float32([[10, 100], [200, 50], [100, 250]])  # desired point locations after transform
(from feature matching, e.g.)

M_aff = cv2.getAffineTransform(pts1, pts2)  # compute affine transform (needs exactly 3
corresponding point pairs)
print "Computed affine transform matrix:"
print M_aff

img_aff = cv2.warpAffine(img, M_aff, (width, height))  # warp original image by applying it
cv2.imshow("Warped by affine transform", img_aff)
```

Projective

```
import cv2
import numpy as np

# Read image
img = cv2.imread('berlin-wall-03.png')
height, width = img.shape[:2]
cv2.imshow("Original", img)
```

```
# Perspective transform
pts1 = np.float32([[220, 85], [223, 414], [602, 190], [616, 323]])
pts2 = np.float32([[75, 75], [75, 225], [500, 75], [500, 225]])

M_persp = cv2.getPerspectiveTransform(pts1, pts2)  # compute perspective transform (needs exactly 4
points)
print "Computed perspective transform matrix:"
print M_persp

img_persp = cv2.warpPerspective(img, M_persp, (600, 300))  # apply transform; last arg is output
image size
cv2.imshow("Perspective transform", img_persp)
```

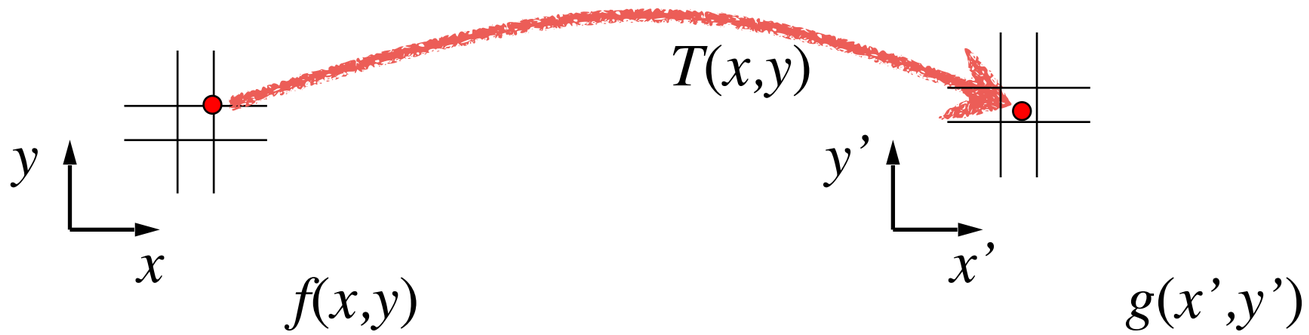
Forward warping



$f(x,y)$ $g(x',y')$

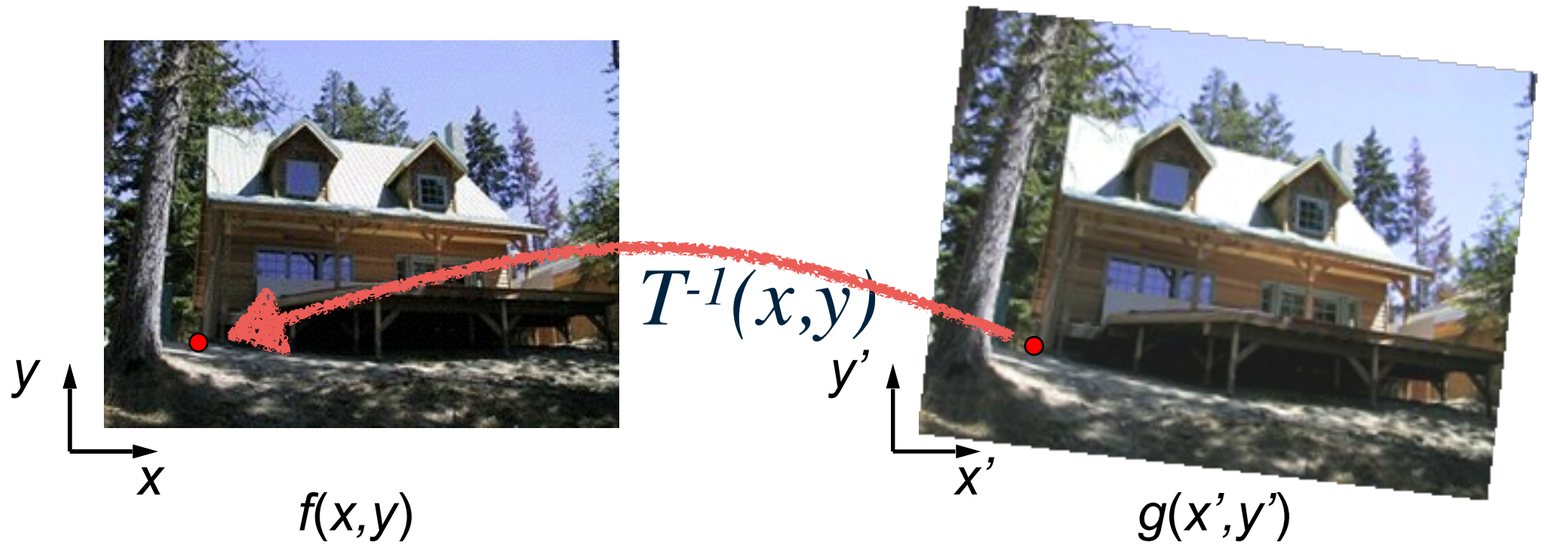
* Send each pixel $f(x,y)$ to its corresponding location
  * $(x',y') = T(x,y)$ in the second image
* Q: what if pixel lands "between" two pixels?

Forward warping



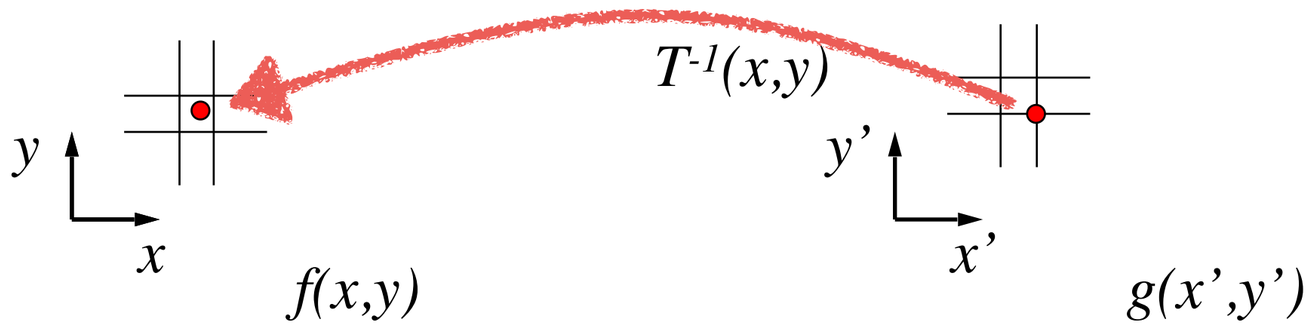
* Send each pixel $f(x,y)$ to its corresponding location
  * $(x',y') = T(x,y)$ in the second image
* Q: what if pixel lands "between" two pixels?
* A: distribute color among neighboring pixels (x',y')

Inverse warping



* Get each pixel $g(x',y')$ from its corresponding location
  * $(x,y) = T^{-1}(x',y')$ in the first image
* Q: what if pixel comes from "between" two pixels?

Inverse warping



* Get each pixel $g(x',y')$ from its corresponding location
  * $(x,y) = T^{-1}(x',y')$ in the first image
* Q: what if pixel comes from "between" two pixels?
* A: Interpolate color value from neighbors

Forward vs. inverse warping

* Q: which is better?

* A: usually inverse ⇒ eliminates holes

  * however, it requires an invertible warp function ⇒ not always possible...

Summary

* Learned about Image Transformations (besides image filtering)
* Studied details of Rigid to Projective Transformation of Images

Credits

* For more information, see:
  * Richard Szeliski (2010) Computer Vision: Algorithms and Applications, Springer (Chapter 2)
* Some concepts in slides motivated by similar slides by Aaron Bobick, James Hays and Greg Turk
* Additional list will be available on website

Computational Photography

* Study the basics of computation and its impact on the entire workflow of photography, from capturing, manipulating and collaborating on, and sharing photographs.

Georgia Tech College of Computing