# Image transformations

# Source of the slides

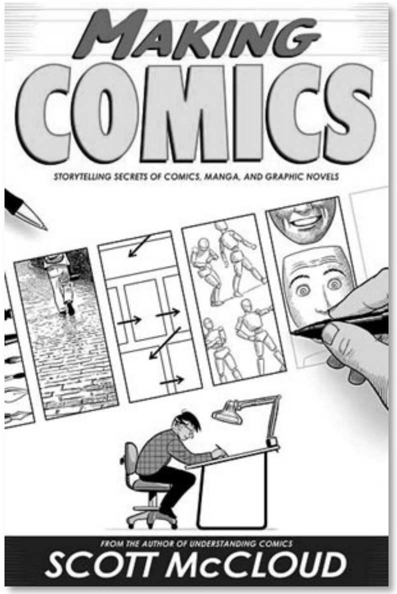
- Introduction to Computer Vision, CS5670, Spring 2024, Cornell Tech (*Image Transformations*) → Noah Snavely and others: Yung-Yu Chuang, Fredo Durand, Alexei Efros, William Freeman, James Hays, Svetlana Lazebnik, Andrej Karpathy, Fei-Fei Li, Srinivasa Narasimhan, Silvio Savarese, Steve Seitz, Richard Szeliski, and Li Zhang.

# Image alignment

# Image alignment

Why don’t these image line up exactly?

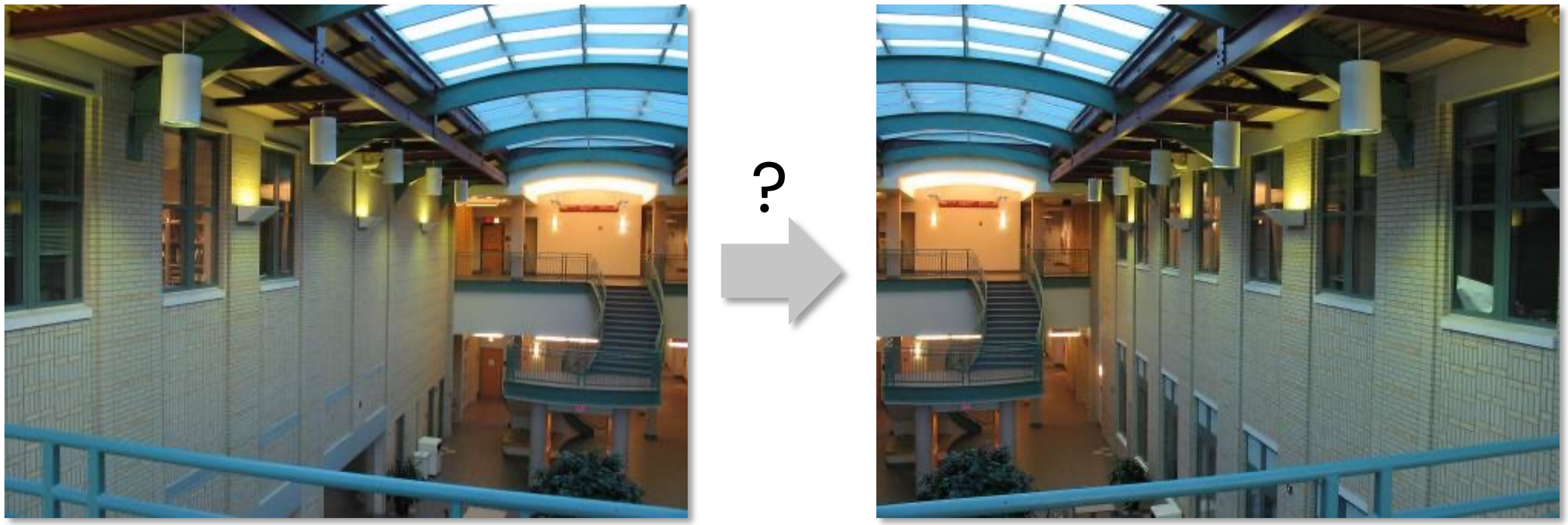
# What is the geometric relationship between these two images?




?


**Answer: Similarity transformation** (translation, rotation, uniform scale)

# What is the geometric relationship between these two images?



# What is the geometric relationship between these two images?

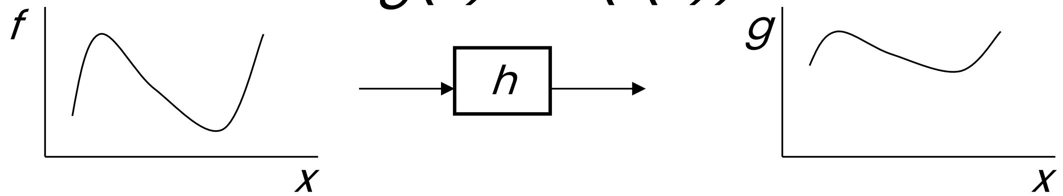
**Very important for creating mosaics!**

First, we need to know what this transformation is.

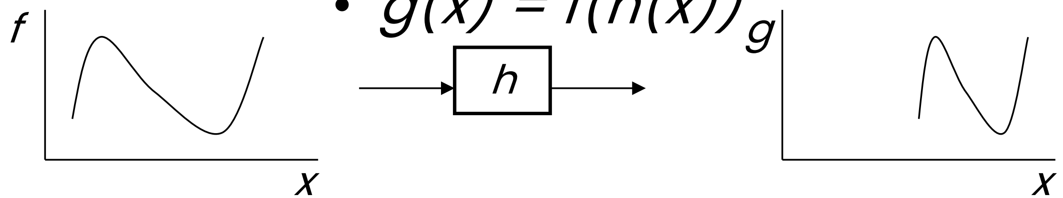
Second, we need to figure out how to compute it using feature matches.

# Image Warping

- image filtering: change *range* of image
  - $g(x) = h(f(x))$




- image warping: change *domain* of image
  - $g(x) = f(h(x))$

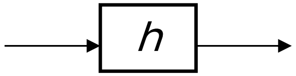


# Image Warping

- image filtering: change *range* of image

  - $g(x) = h(f(x))$

$f$

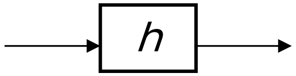



$g$

- image warping: change *domain* of image

  - $g(x) = f(h(x))$

$f$




$g$

# Parametric (global) warping

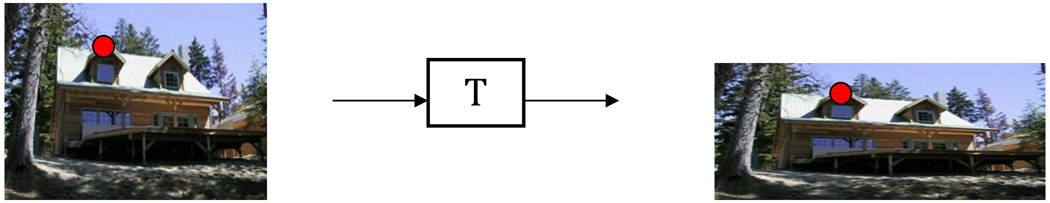
- Examples of parametric warps:

translation

rotation

aspect

# Parametric (global) warping




$\mathbf{p} = (x, y)$ $\mathbf{p'} = (x', y')$

- Transformation T is a coordinate-changing machine:

$$\mathbf{p'} = T(\mathbf{p})$$

- What does it mean that **T** is global?
  - Is the same for any point **p**
  - can be described by just a few numbers (parameters)
- Let's consider *linear* transforms (can be represented by a 2x2 matrix):

$$\mathbf{p}' = \mathbf{T}\mathbf{p} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix}$$

# Common linear transformations

- Uniform scaling by s

(0,0) 

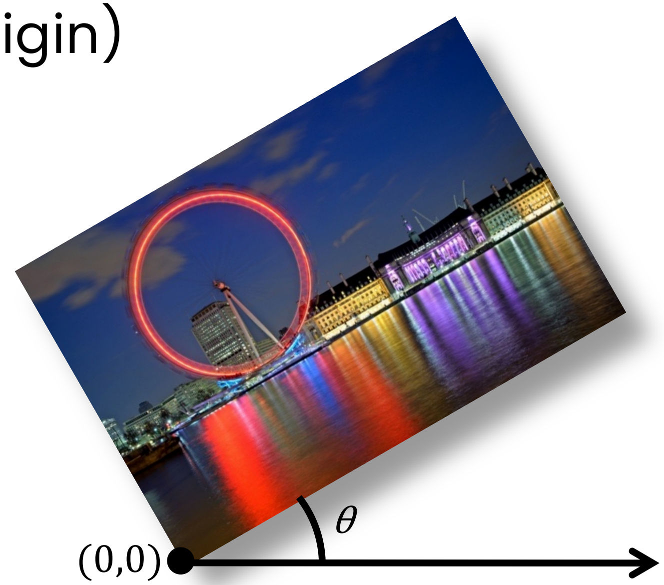
(0,0) 

$$\mathbf{S} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$$

What is the inverse?

# Common linear transformations

- Rotation by angle $\theta$ (about the origin)


$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

What is the inverse?

For rotations:

$$\mathbf{R}^{-1} = \mathbf{R}^T$$

# 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D mirror across Y axis?

$$\begin{aligned} x' &= -x \\ y' &= y \end{aligned}$$

2D mirror across line y = x?

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

# 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D mirror across Y axis?

$$\begin{aligned} x' &= -x \\ y' &= y \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

2D mirror across line y = x?

$$\begin{aligned} x' &= y \\ y' &= x \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

# 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Translation?

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

# 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Translation?

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

NO!

Translation is **not** a linear operation on 2D coordinates

# All 2D Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

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

- Properties of linear transformations:
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

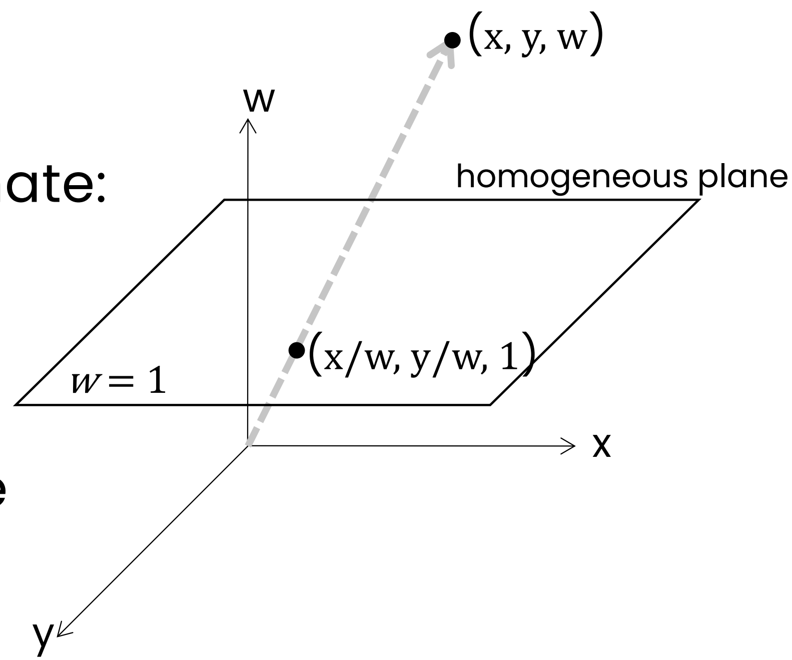
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

# Homogeneous coordinates

Trick: add one more coordinate:

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates




Converting *from* homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

# Translation

- Solution: homogeneous coordinates to the rescue

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

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

# Affine transformations

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

any transformation represented by a 3x3 matrix with last row [ 0 0 1 ] we call an *affine* transformation

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

# Basic affine transformations

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

**Translate**

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 & 0 \\ 0 & \boldsymbol{s}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

**Scale**

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

**2D *in-plane* rotation**

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh}_x & 0 \\ \boldsymbol{sh}_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

**Shear**

# Affine transformations

- Affine transformations are combinations of …
  - Linear transformations, and
  - Translations

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

- Properties of affine transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

# Is this an affine transformation?

# Where do we go from here?

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

affine transformation

← what happens when we mess with this row?

# Projective Transformations *aka* Homographies *aka* Planar Perspective Maps

$$\mathbf{H} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

Called a *homography* (or *planar perspective map*)

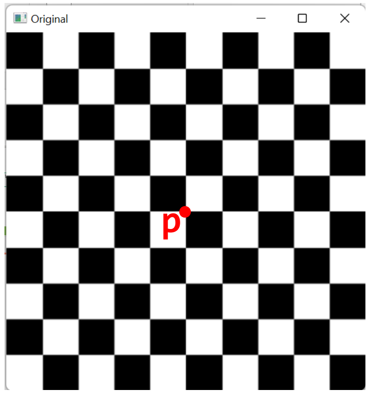
# Homographies

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

**What happens when the denominator is 0?**

$$\sim \begin{bmatrix} \frac{ax+by+c}{gx+hy+1} \\ \frac{dx+ey+f}{gx+hy+1} \\ 1 \end{bmatrix}$$

# What happens?




Projective Transformation (example code)
Input:

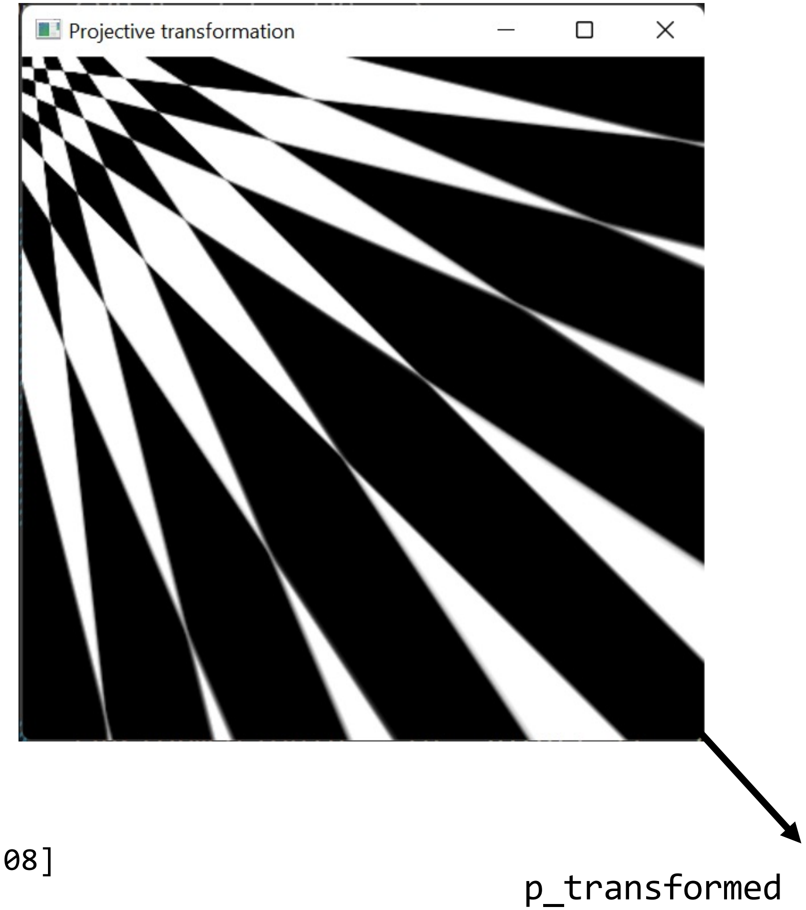
```
T = np.float32([
         [ 0.1200,  0.0000, 0.0000],
         [ 0.0000,  0.1200, 0.0000],
         [-0.0025, -0.0025, 1.0000]]
p = [200, 200, 1]
p_transformed = np.matmul(T,p)
```

Output p_transformed:

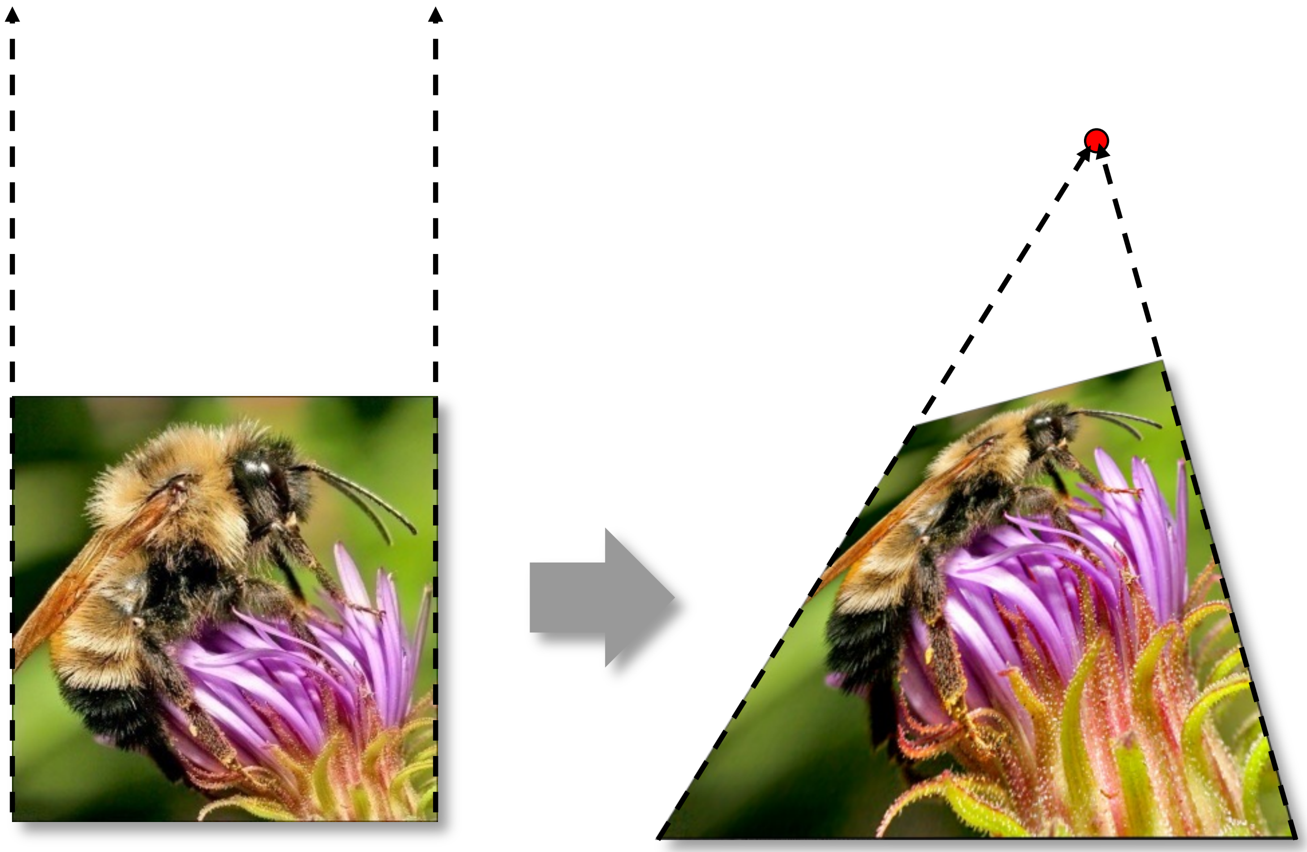
`[2.39999995e+01 2.39999995e+01 2.23517418e-08]`

`[24. 24.  0.]` (rounded)

`[1073741800.0 1073741800.0 1.0]` (as homogeneous coordinate)



# Points at infinity



# Homographies

# Homographies

- Homographies …
  - Affine transformations, and
  - Projective warps

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

- Properties of projective transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines do not necessarily remain parallel
  - Ratios are not preserved
  - Closed under composition

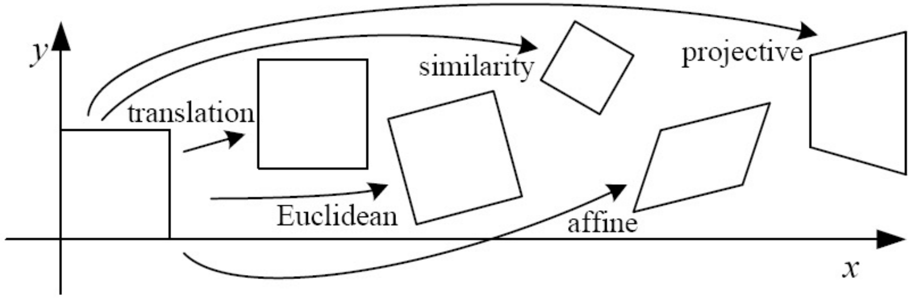
Key fact: homographies are only defined up to a scale factor (e.g., ***H*** and 2***H*** are equivalent homographies)

# Alternate formulation for homographies

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

where the length of the vector $[h_{00}\ h_{01} \dots h_{22}]$ is 1

# 2D image transformations



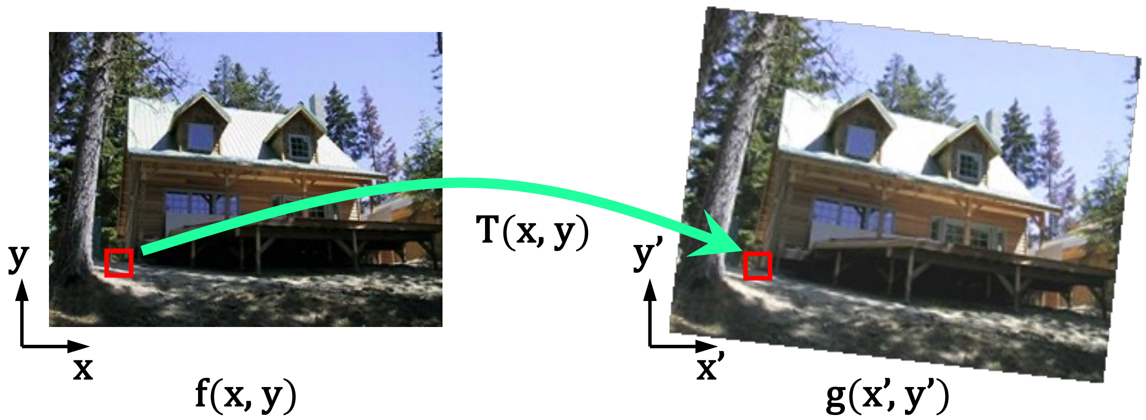

| Name | Matrix | # D.O.F. | Preserves: | Icon |
|---|---|---|---|---|
| translation | $\left[\, \boldsymbol{I} \mid \boldsymbol{t} \,\right]_{2\times 3}$ | 2 | orientation $+\cdots$ | |
| rigid (Euclidean) | $\left[\, \boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times 3}$ | 3 | lengths $+\cdots$ | |
| similarity | $\left[\, s\boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times 3}$ | 4 | angles $+\cdots$ | |
| affine | $\left[\, \boldsymbol{A} \,\right]_{2\times 3}$ | 6 | parallelism $+\cdots$ | |
| projective | $\left[\, \tilde{\boldsymbol{H}} \,\right]_{3\times 3}$ | 8 | straight lines | |

These transformations are a nested set of groups

- Closed under composition and inverse is a member

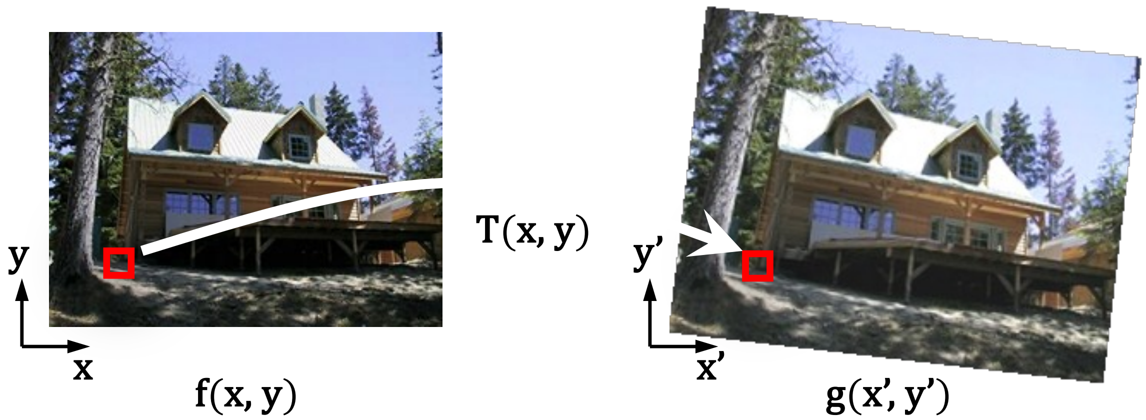
# Implementing image warping

- Given a coordinate transform $(\mathbf{x'}, \mathbf{y'}) = \mathbf{T}(\mathbf{x}, \mathbf{y})$ and a source image $\mathbf{f}(\mathbf{x}, \mathbf{y})$, how do we compute a transformed image $\mathbf{g}(\mathbf{x'}, \mathbf{y'}) = \mathbf{f}(\mathbf{T}(\mathbf{x}, \mathbf{y}))$?



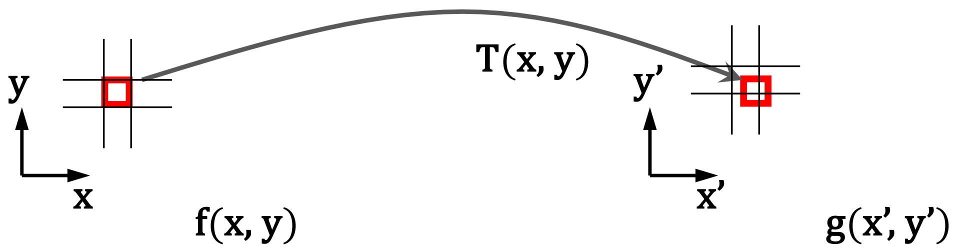
# Forward Warping

- Send each pixel $(\mathbf{x}, \mathbf{y})$ to its corresponding location $(\mathbf{x'}, \mathbf{y'}) = \mathbf{T}(\mathbf{x}, \mathbf{y})$ in $\mathbf{g}(\mathbf{x'}, \mathbf{y'})$
    - What if pixel lands “between” two pixels?



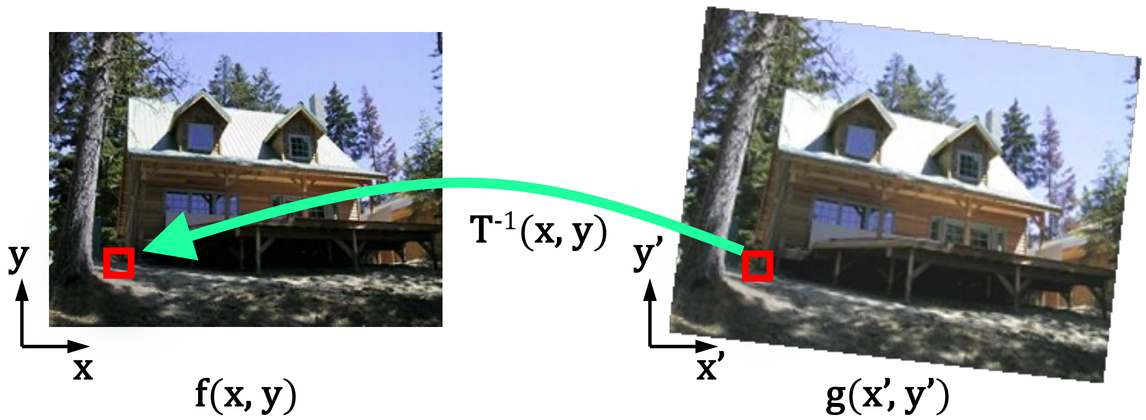
# Forward Warping

- Send each pixel $(x, y)$ to its corresponding location $(x', y') = T(x, y)$ in $g(x', y')$
    - What if pixel lands "between" two pixels?
    - Answer: add "contribution" to several pixels, normalize later (*splatting*)
    - Can still result in holes



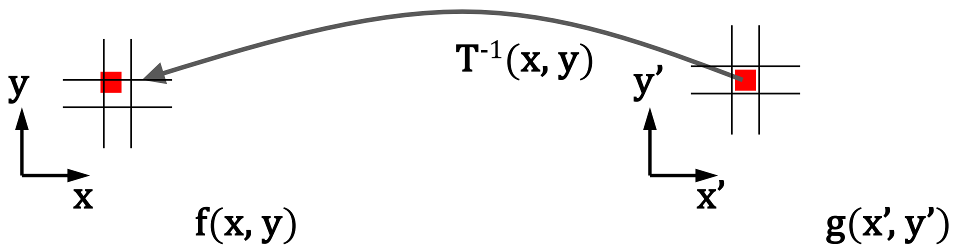
# Inverse Warping

- Get each pixel $g(x', y')$ from its corresponding location $(x, y) = T^{-1}(x, y)$ in $f(x, y)$
    - Requires taking the inverse of the transform
    - What if pixel comes from “between” two pixels?



# Inverse Warping

- Get each pixel $\mathbf{g}(\mathbf{x}', \mathbf{y}')$ from its corresponding location $(\mathbf{x}, \mathbf{y}) = \mathbf{T}^{-1}(\mathbf{x}, \mathbf{y})$ in $\mathbf{f}(\mathbf{x}, \mathbf{y})$
    - What if pixel comes from “between” two pixels?
    - Answer: *resample* color value from *interpolated (prefiltered)* source image



# Interpolation

- Possible interpolation filters:
  - nearest neighbor
  - bilinear
  - bicubic
  - sinc

- Needed to prevent “jaggies” and “texture crawl” (with prefiltering)