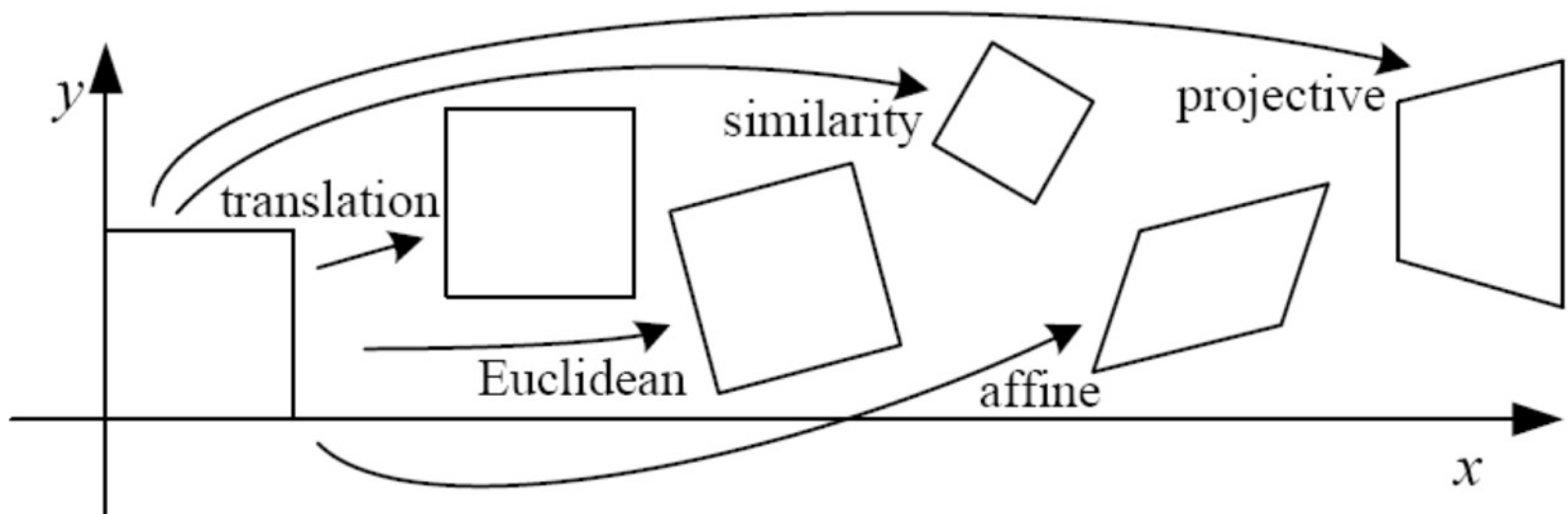


# Image Transformations

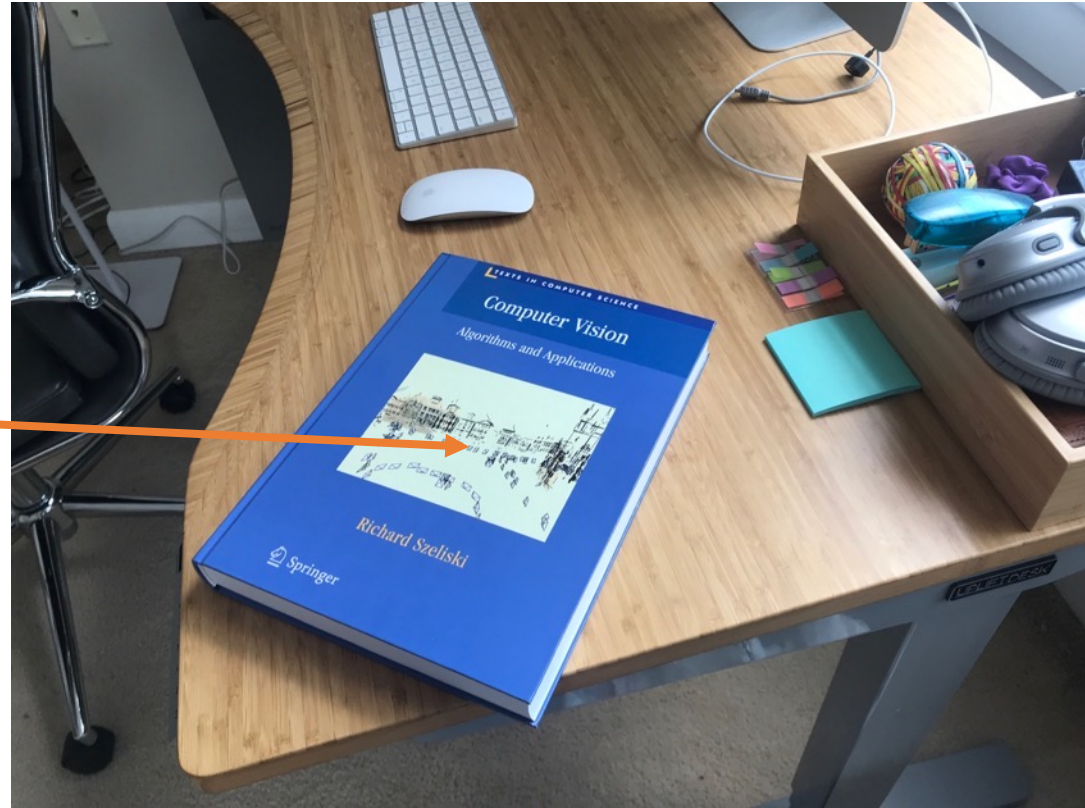
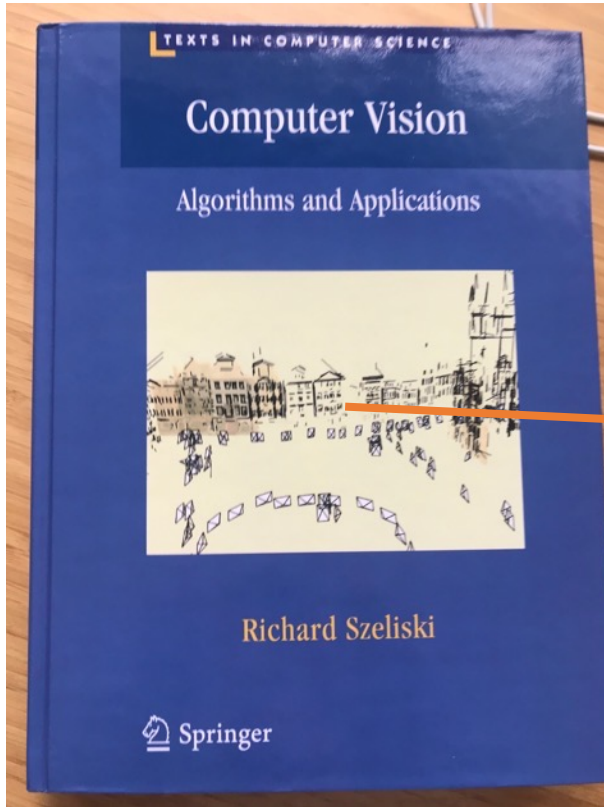
CS 482, Prof. Stein

Lecture 07

Today we're going to take a brief aside from talking about features to talk about how we transform images

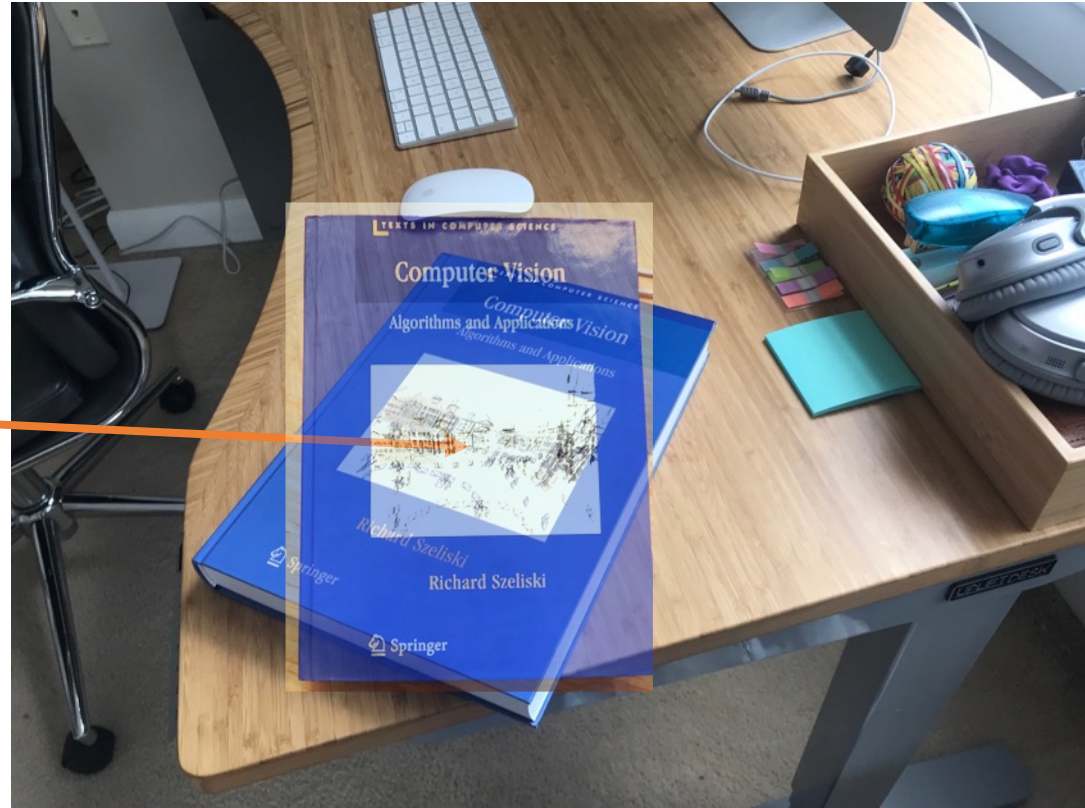
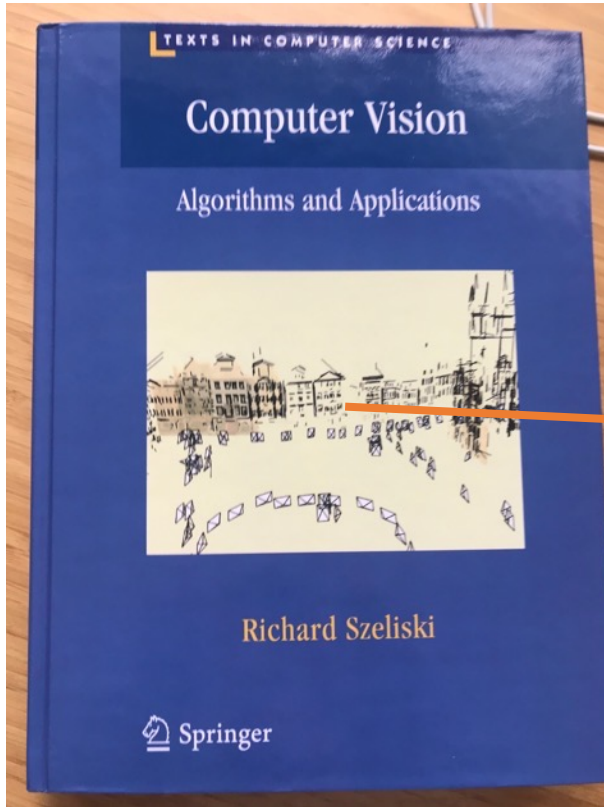


# Motivation: finding objects requires a mathematical definition of warping



Let's say we want to "align" these two images

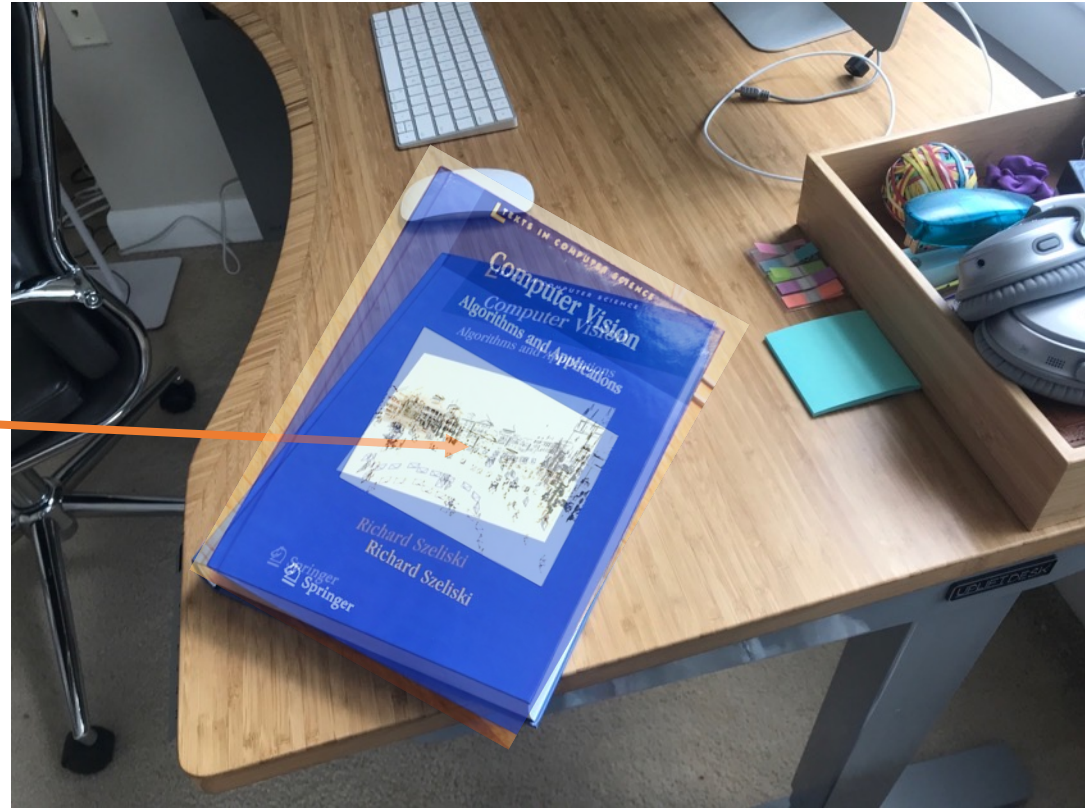
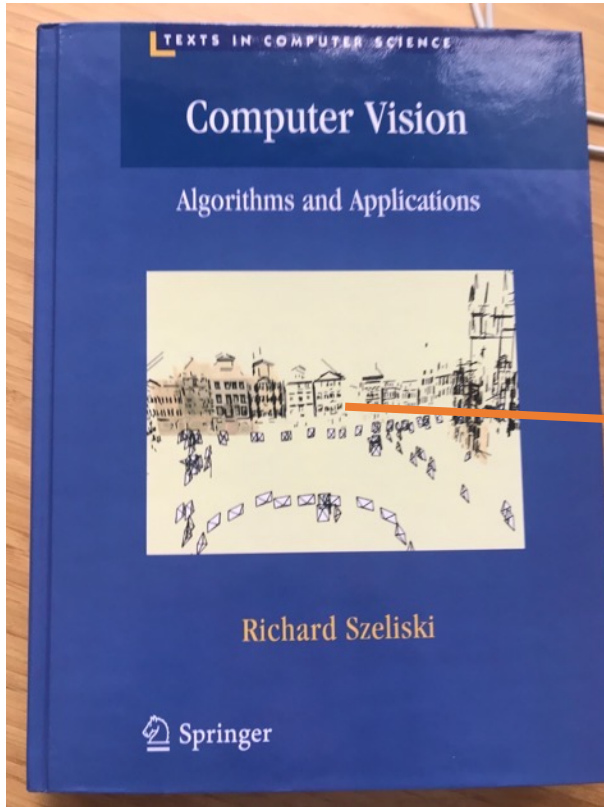
# Motivation: finding objects requires a mathematical definition of warping



Let's say we want to "align" these two images:  
Scale and Translation alone are *not enough* to match them

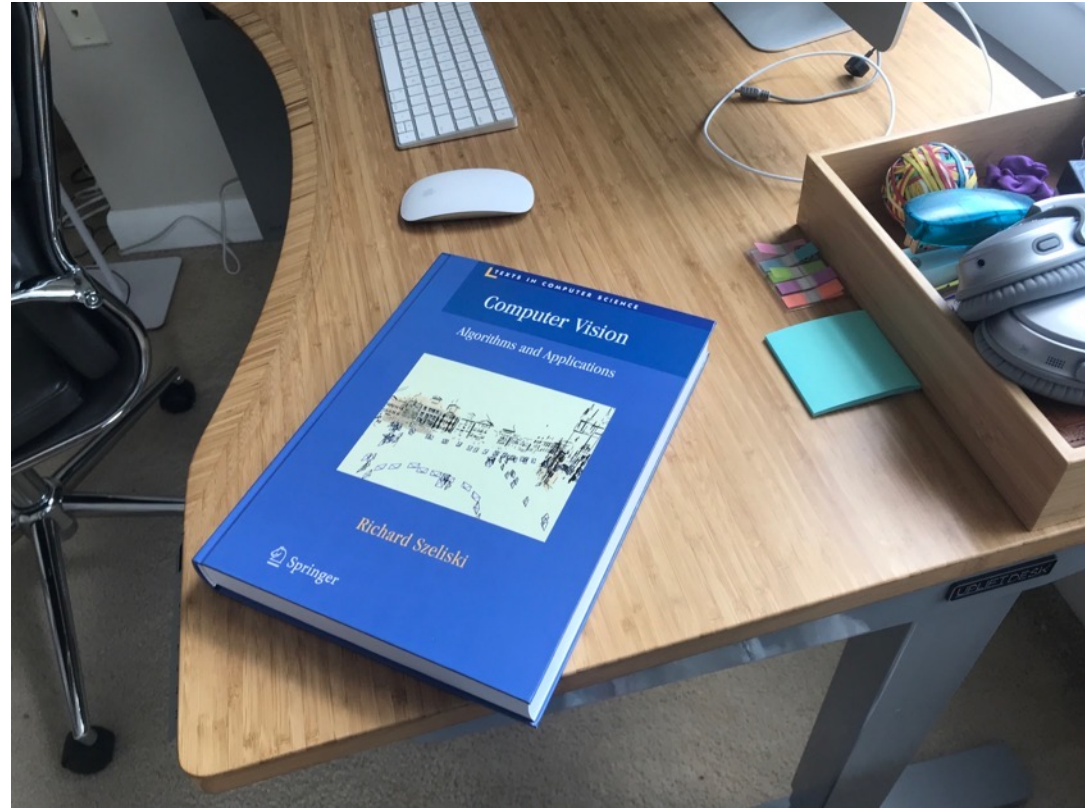
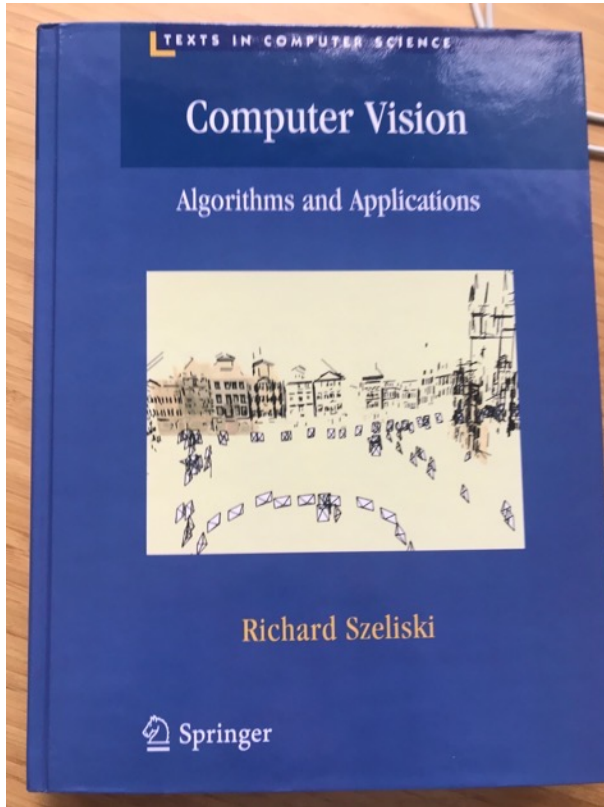


# Motivation: finding objects requires a mathematical definition of warping



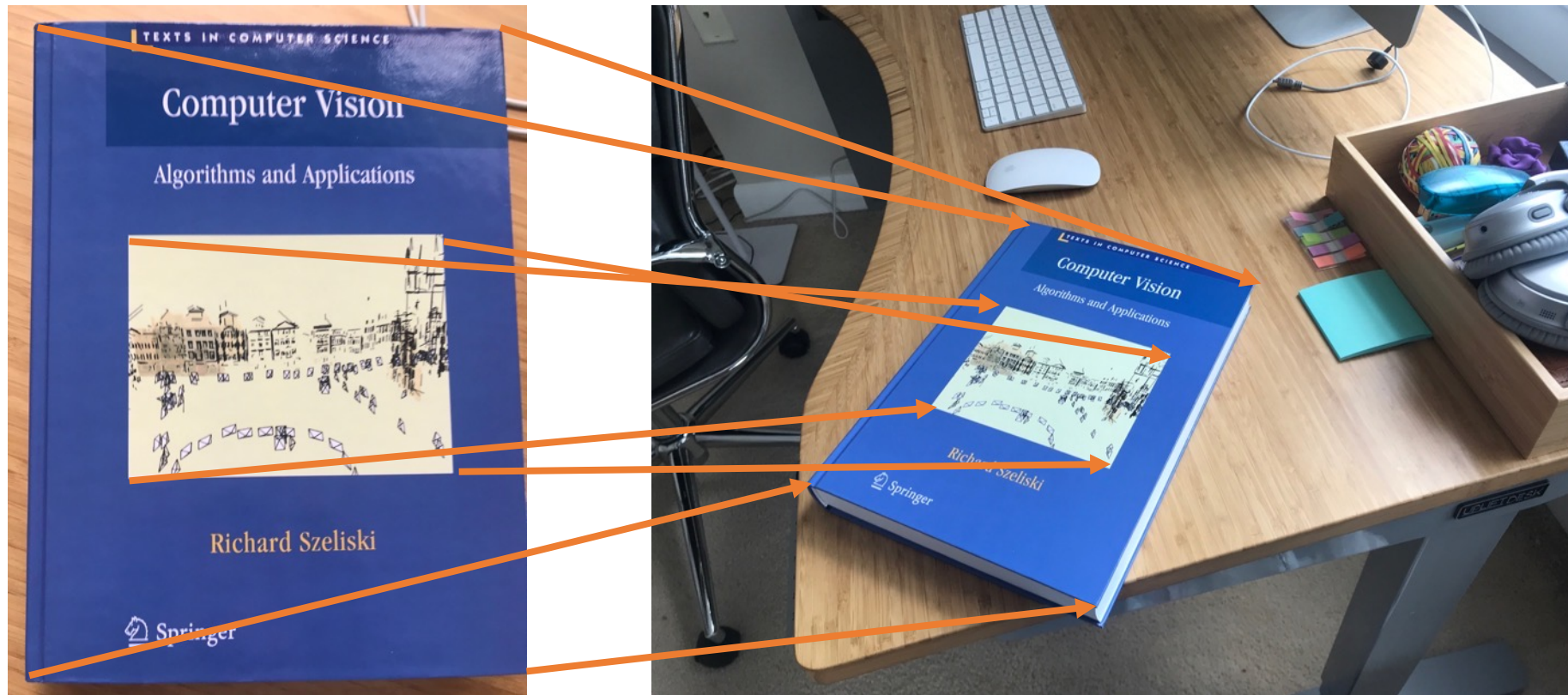
Let's say we want to "align" these two images:  
Adding rotation helps us get closer but is still insufficient.

# Motivation: we need a more general notion of *warping* to relate two images





# Motivation: we need a more general notion of *warping* to relate two images



By detecting and matching features common to the two images, we can estimate the transformation between them.

Motivation: we need a more general notion of *warping* to relate two images

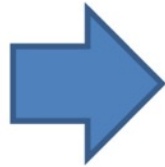




Motivation: we need a more general notion of *warping* to relate two images



# Motivation: we need a more general notion of *warping* to relate two images



Today, we'll set up the mathematical formalism that allows us to define and execute general image transformations.

Next week, we'll focus on how to automatically compute such transformations.

# Reading & Slide Credits

Readings:

- Szeliski: 3.6

Slide Credits (from which many of these slides are either directly taken or adapted)

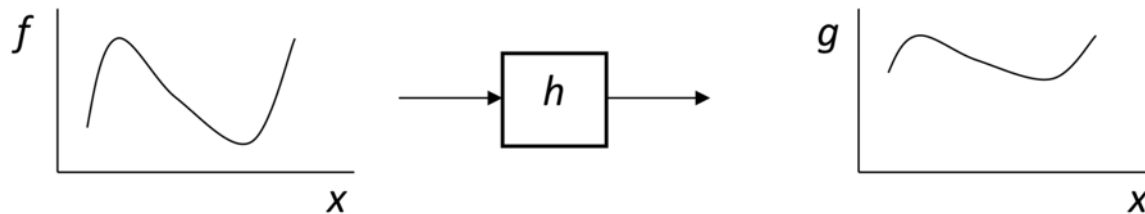
- [Cornell Tech Computer Vision Course](#) (Noah Snavely)



# Some review: filtering vs. warping

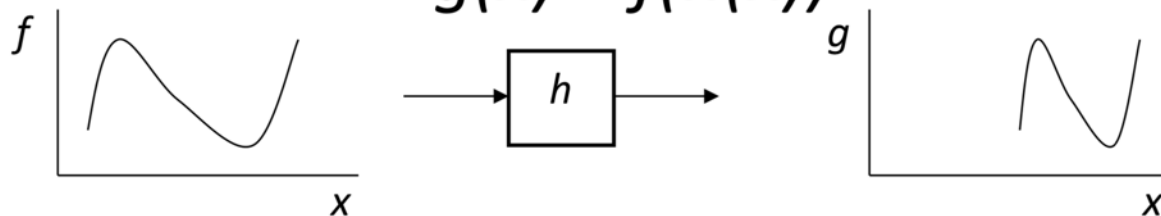
Filtering involves applying functions to change the *range* (output) of an image:

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



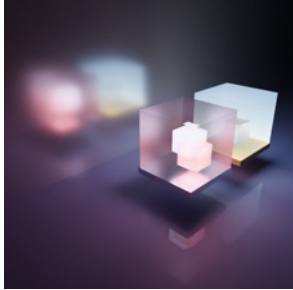
Warping instead modifies the *domain* (input) of an image:

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



# Warping

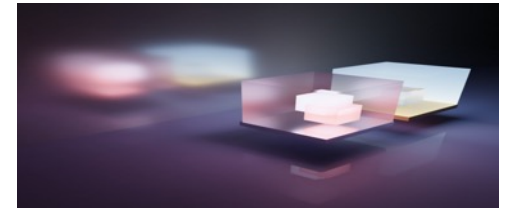
We've already talked about a few different types of transformations:



Scale



Rotation

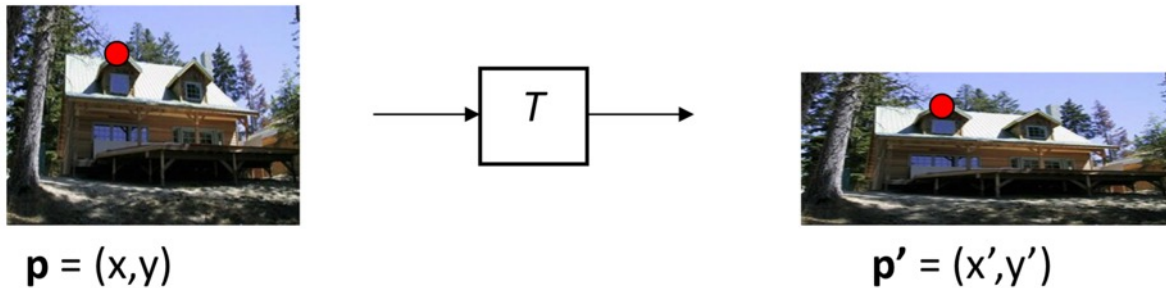


Aspect

And, of course, translations.

# Global Warping: apply a function to every coordinate.

The transformation  $T$  applies some fixed operation to the coordinates:



A “global” transformation treats every point the same.

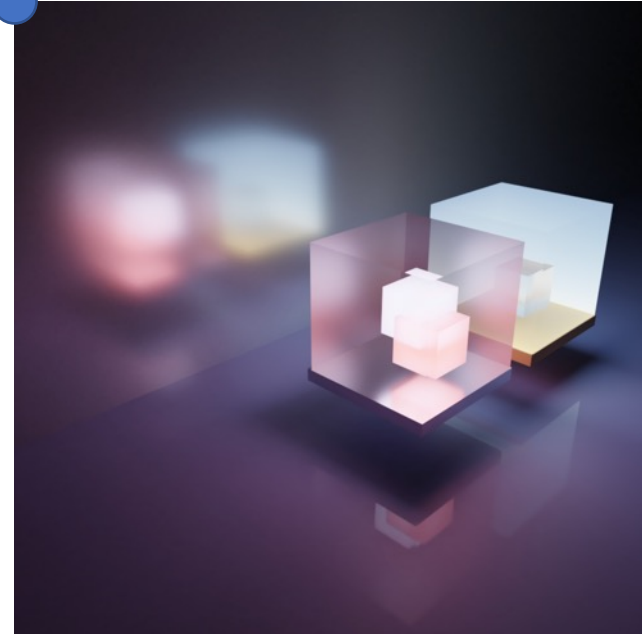
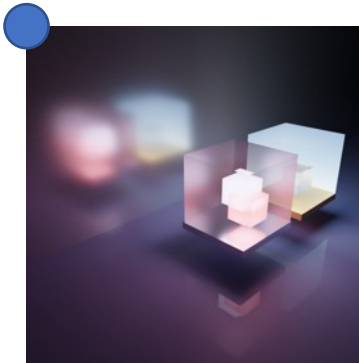
Commonly, these transformations are *linear*, i.e. they can be represented by matrix:

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



# Linear Warping: what operations can we represent by a 2x2 matrix?

Scale is a linear warp:



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

# Let's implement this!

Off to the Jupyter Notebook for today...

# Let's implement this!

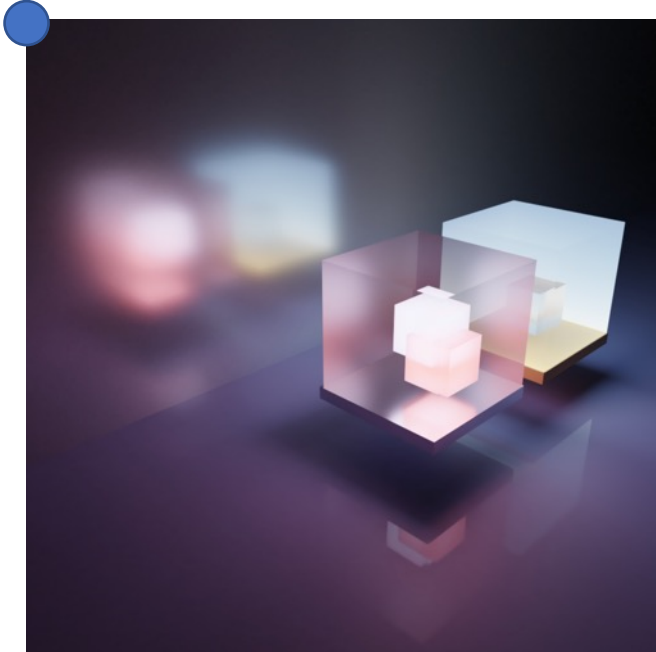
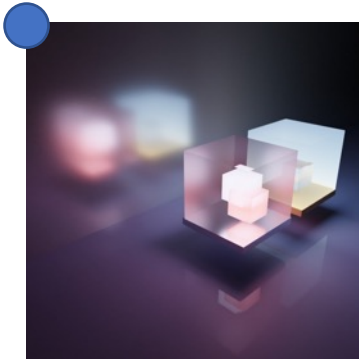
Off to the Jupyter Notebook for today...

```
1  ## Solution: 2D image coordinates
2
3  initial_image_coords = np.array([
4      [0, 0],
5      [0, 15],
6      [40, 0],
7      [40, 15],
8  ]).T
9
```



# Linear Warping: what operations can we represent by a 2x2 matrix?

Scale is a linear warp:



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

If  $s=2$ , does the image become larger or smaller?

# Linear Warping: Scale

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

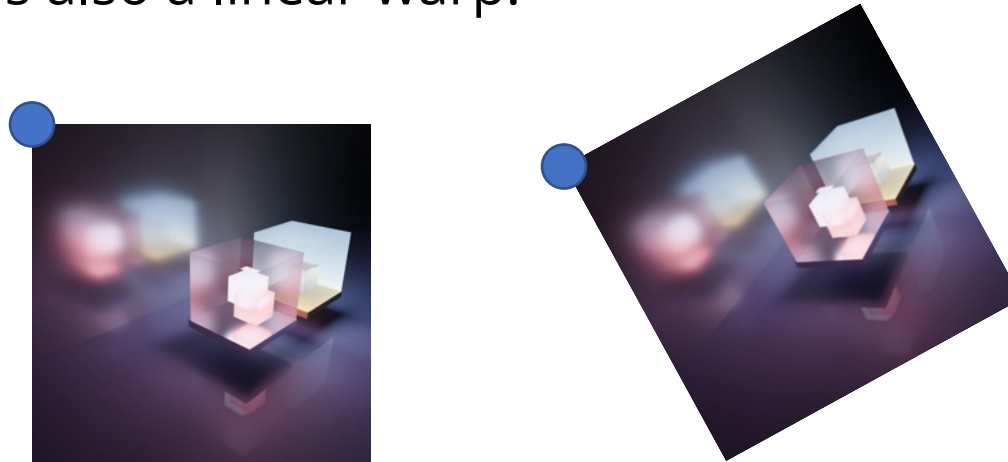
If  $s=2$ , does the image become larger or smaller? This depends on how you define the transformation operation.

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

Pixel  $[2, 2]$  in the transformed coordinates, correspond to  $[1, 1]$  in the original image: the image has become larger.

# Linear Warping: what operations can we represent by a 2x2 matrix?

Rotation is also a linear warp:



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

# Linear Warping: what operations can we represent by a 2x2 matrix?

So are reflection operations:

2D mirror about Y axis?

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

2D mirror across line  $y = x$ ?

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

# Linear Warping: what operations can we represent by a 2x2 matrix?

What about translation?

$$x' = x + t_x$$

$$y' = y + t_y$$

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



# Linear Warping: what operations can we represent by a 2x2 matrix?

What about translation?

$$x' = x + t_x$$

$$y' = y + t_y$$

No! Transformation cannot be represented as a 2x2 matrix operation (it isn't linear). [Think about what happens to the (0, 0) coordinate.]

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

# Homogeneous Coordinates:

# Homogeneous Coordinates: How we add translation

The trick is to add a third coordinate:

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

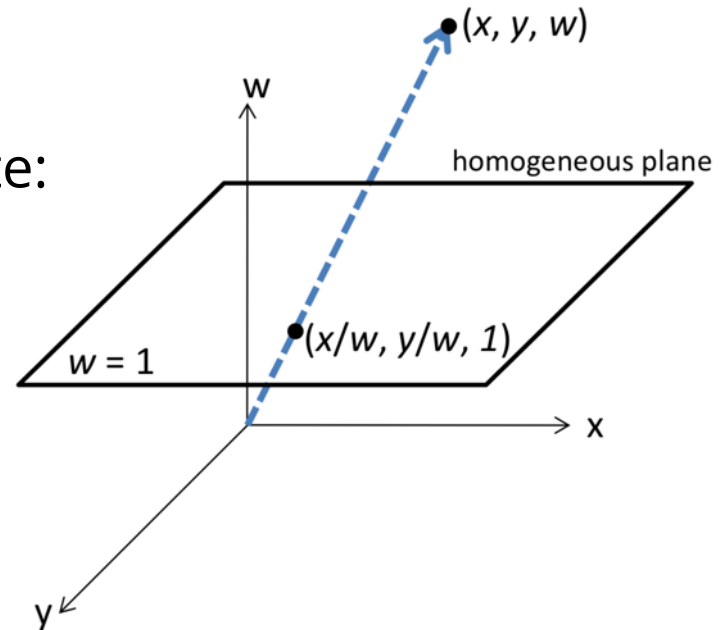
homogeneous image  
coordinates

# Homogeneous Coordinates: How we add translation

The trick is to add a third coordinate:

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

homogeneous image  
coordinates



After applying a transformation (now a 3x3 matrix), we can convert back to our “usual” 2D coordinates with:

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

# Homogeneous Coordinates: How we add translation

The trick is to add a third coordinate:

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

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



# Homogeneous Coordinates: 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

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

# Homogeneous Coordinates: 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

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

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

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

Shear

# Homogeneous Coordinates: 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

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

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

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

Shear

We'll discuss out-of-plane transformations later.

# Homogeneous Coordinates: Affine Transformations

Properties of Affine Transformations:

- The origin of the input image *does not* map to the origin in the target image (in general)
- Lines remain lines after transformations
- Lines that are parallel remain parallel

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

Aligning images typically requires an *out of plane* rotation of some kind

Not, in general, an affine transformation.





# Homographies

(Also known as projective transforms or planar perspective maps)

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

We add elements to this bottom row, allowing the image to rotate out of the plane.

# Homographies

(Also known as projective transforms or planar perspective maps)

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

We add elements to this bottom row, allowing the image to rotate out of the plane.



# Homographies

(Also known as projective transforms or planar perspective maps)

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

We add elements to this bottom row, allowing the image to rotate out of the plane.

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

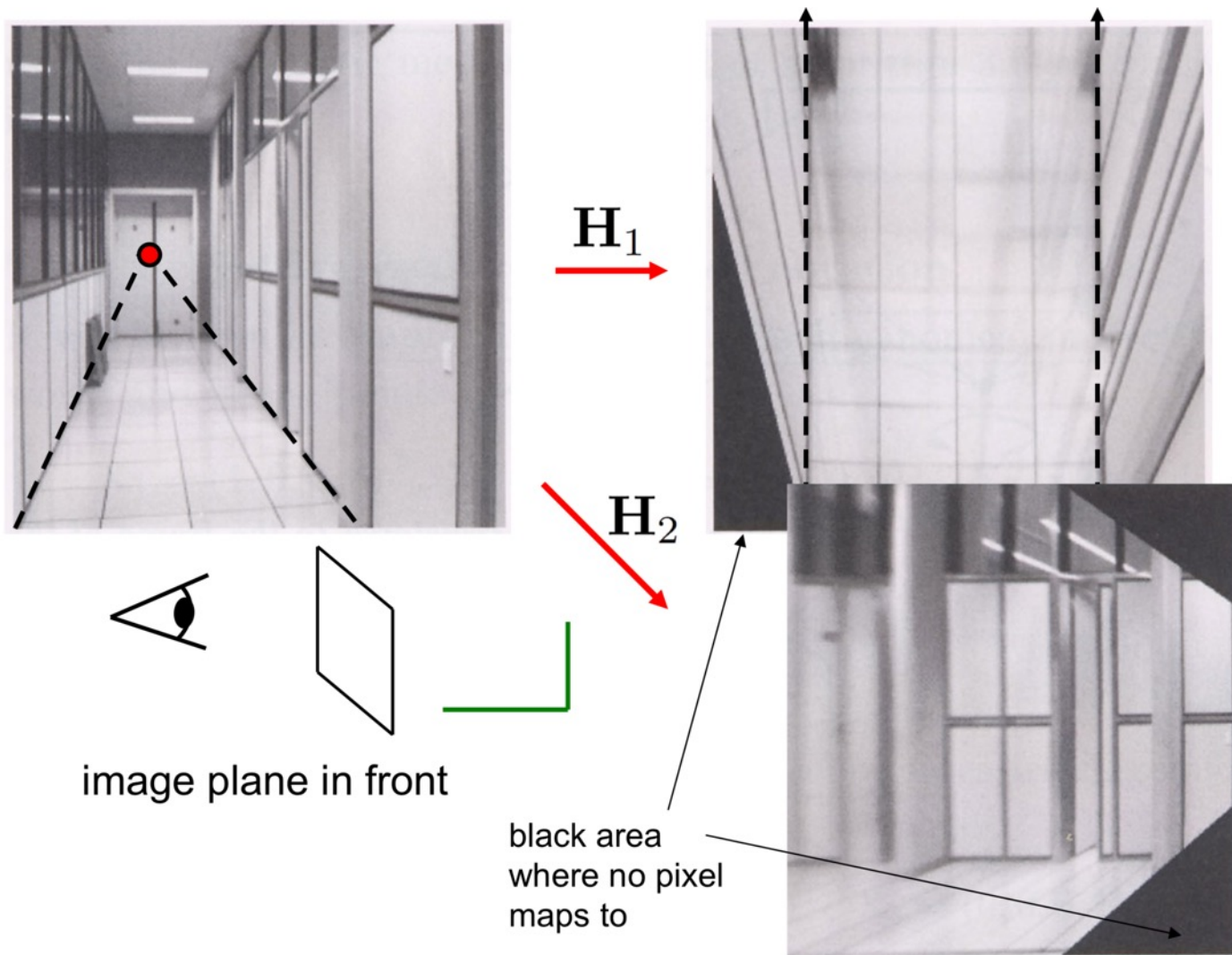


Homographies allow us to have *vanishing points* (at infinity)

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} \frac{ax+by+c}{gx+hy+1} \\ \frac{dx+ey+f}{gx+hy+1} \\ 1 \end{bmatrix}$$



# Homography Example





# Homography Example

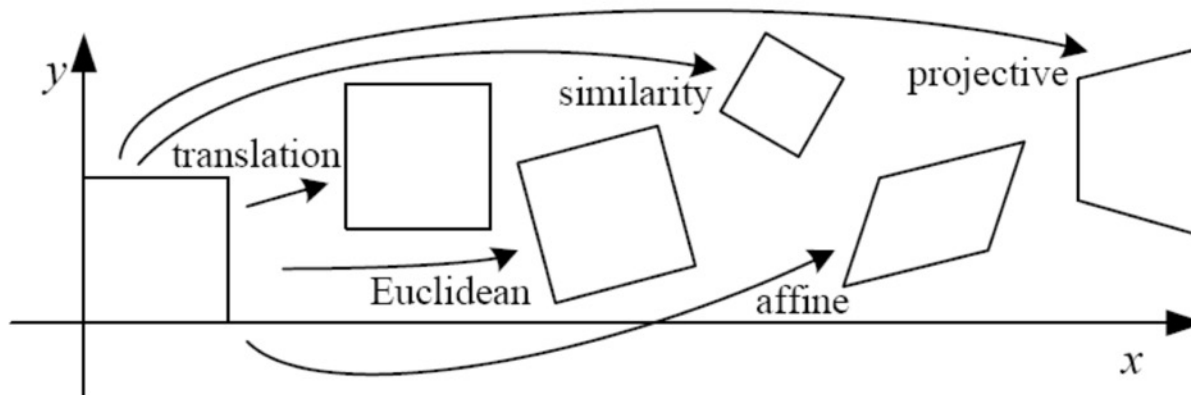







# Homographies Properties

Properties of Affine Transformations:

- The origin of the input image *does not* map to the origin in the target image (in general)
- Lines remain lines after transformations
- Lines that are parallel **do not** remain parallel

# Homographies Properties



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation + ...	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	3	lengths + ...	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	4	angles + ...	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2 \times 3}$	6	parallelism + ...	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3 \times 3}$	8	straight lines	

# Homographies: alternate formulation

We'll be discussing this formulation more next week:

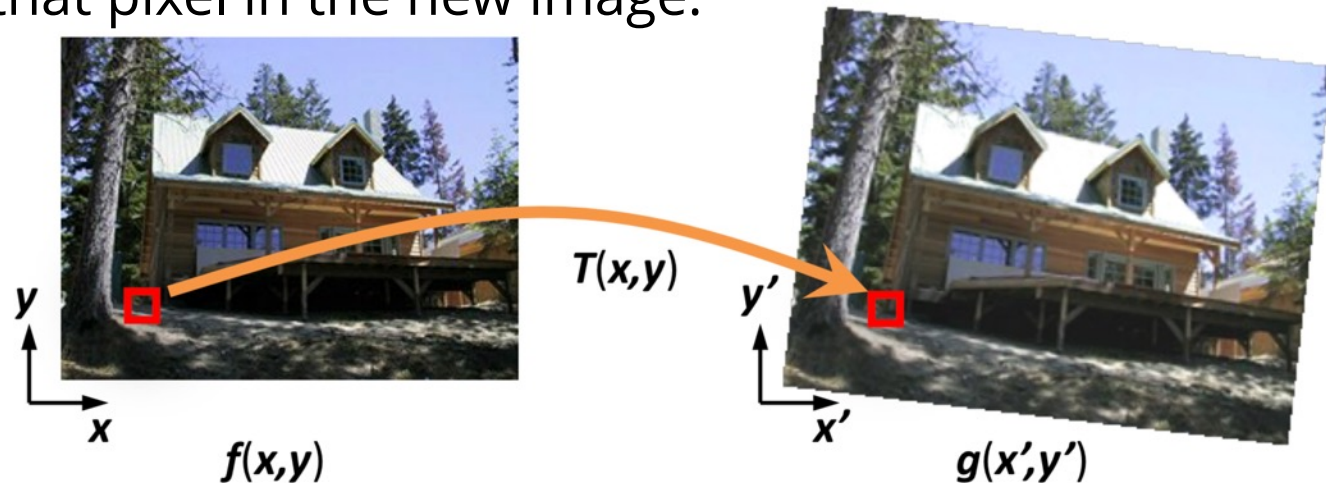
$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

(Subject to the constraint that the entries sum to 1)

# Implementing Image Warping

# The simplest thing you might do is to “forward transform” the original image

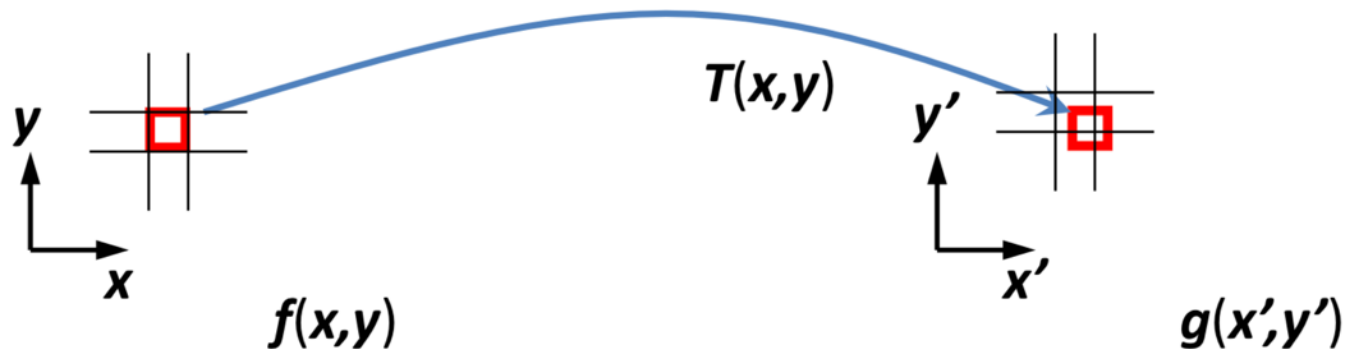
For every pixel in the original image, apply the transformation and set that pixel in the new image.



What happens when points land between pixels?

# The simplest thing you might do is to “forward transform” the original image

For every pixel in the original image, apply the transformation and set that pixel in the new image.

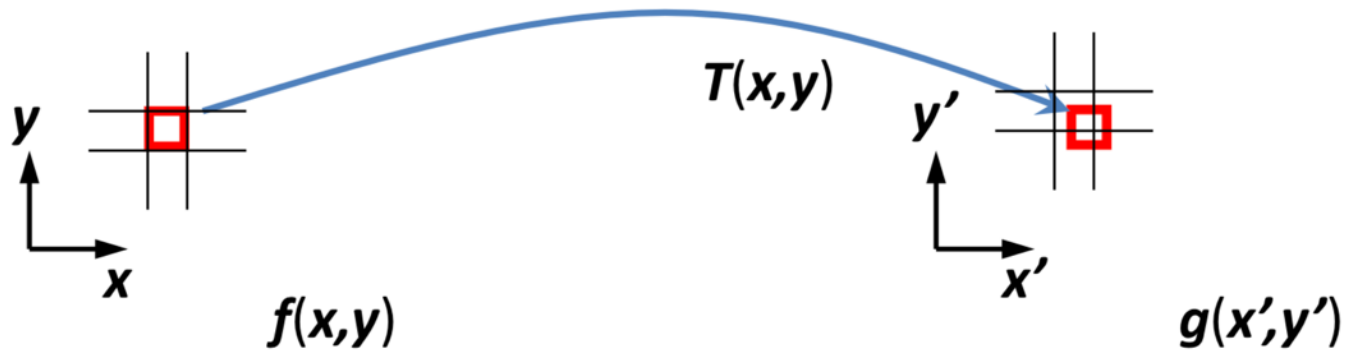


A technique known as “splatting” is used to fill in points between pixels: add a “contribution” to multiple nearby pixels.



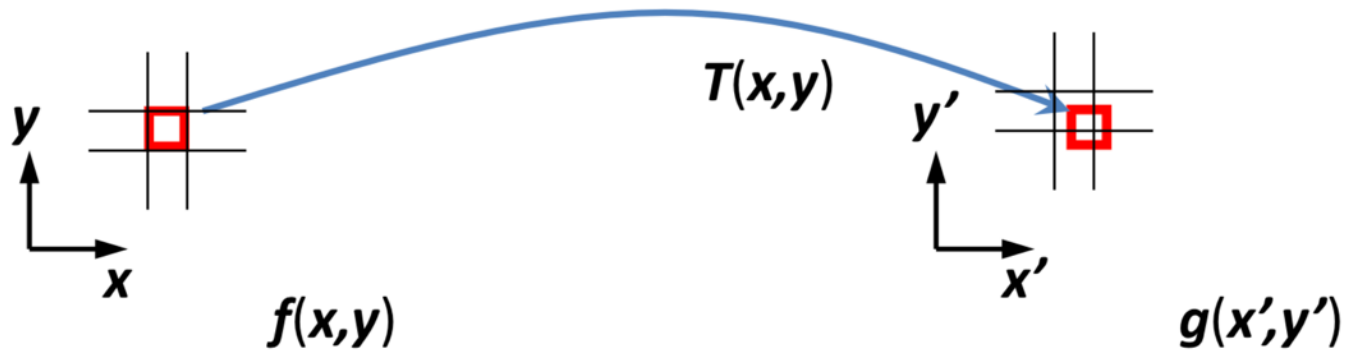
# Recap: remember interpolation?

When we did interpolation, what pixels did we loop through?  
The original image? The target image?



# Recap: remember interpolation?

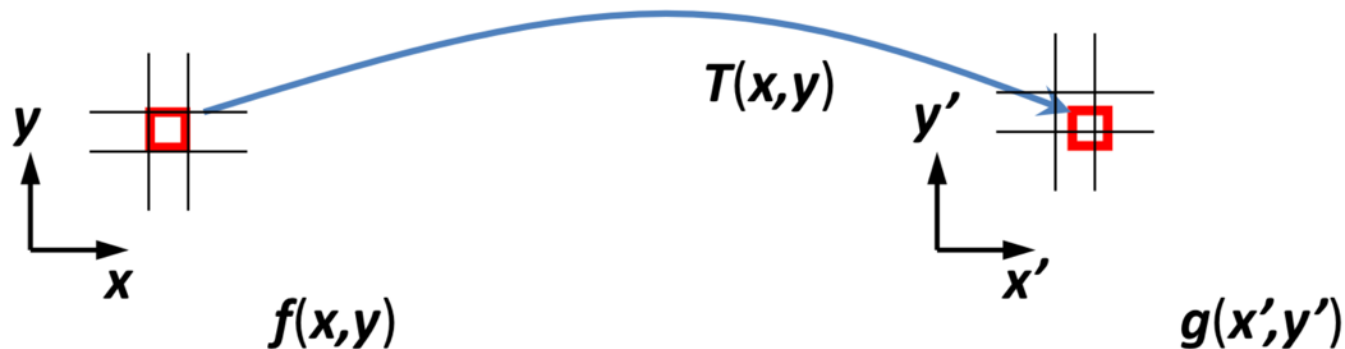
When we did interpolation, what pixels did we loop through?  
The original image? **The target image?**



Especially when we are changing the image size, the "splatting" approach below can leave big holes between pixel values.

# The simplest thing you might do is to “forward transform” the original image

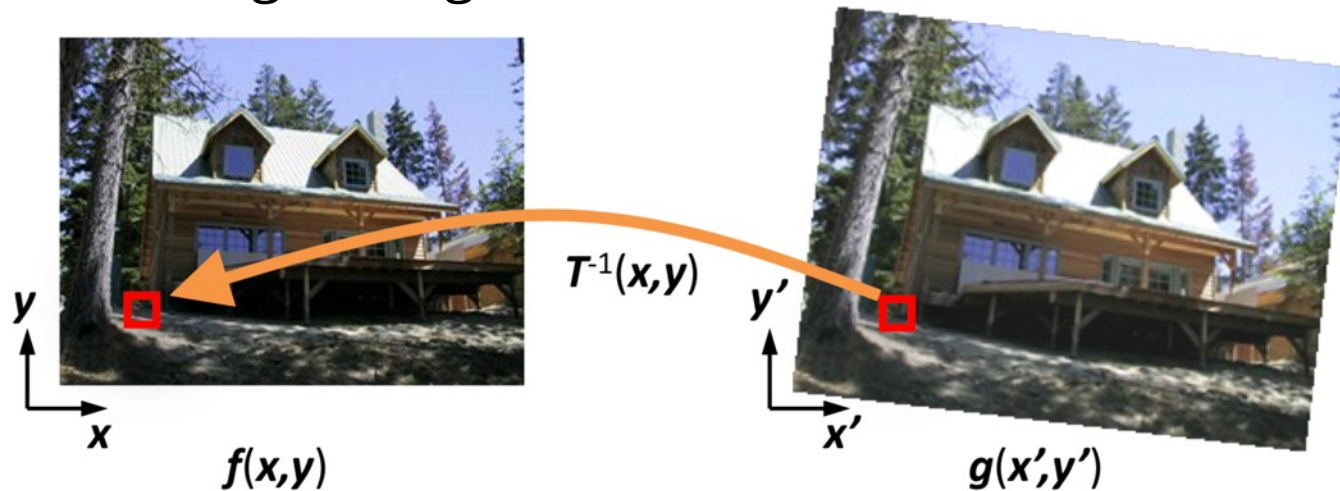
For every pixel in the original image, apply the transformation and set that pixel in the new image.



A technique known as “splatting” is used to fill in points between pixels: add a “contribution” to multiple nearby pixels. Still results in holes, that need to be interpolated over; actually rather mathematically tricky to deal with.

# Instead, it is common to use the inverse transformation:

We compute the inverse transformation for all the pixels we want at the target image:



Whenever the inverse-transformed points fall between pixels, we can interpolate over the original image (using whatever interpolation kernel you choose).

# Image Warping Procedure

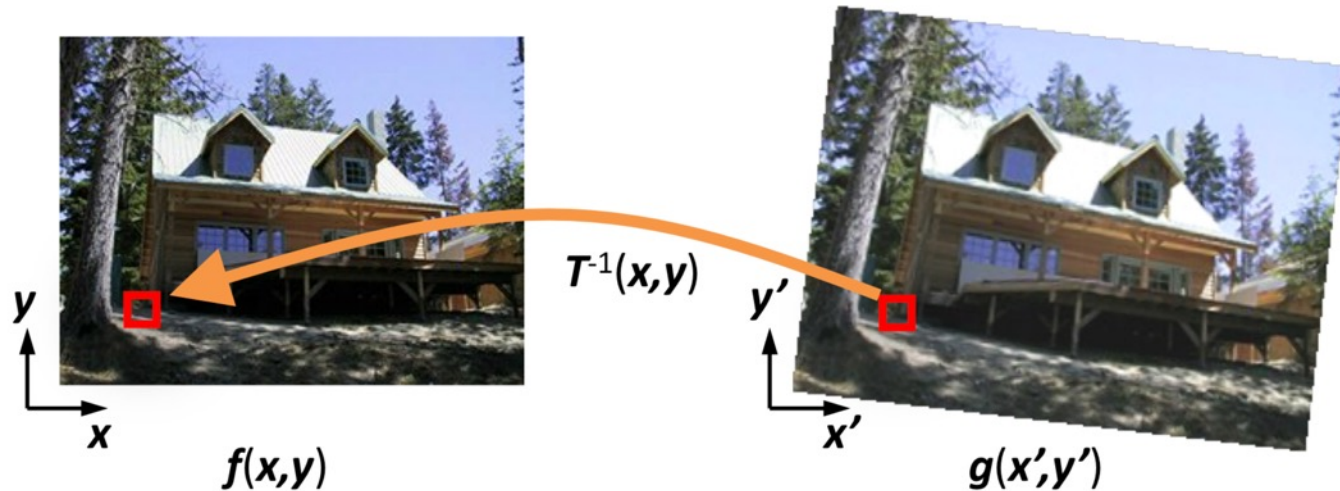
For every pixel in the target (warped) image:

1. Compute location of pixel in input image via the inverse transformation matrix and normalizing.
2. Use interpolation on initial image to compute pixel value (brightness and color).
3. Set pixel value in target image.

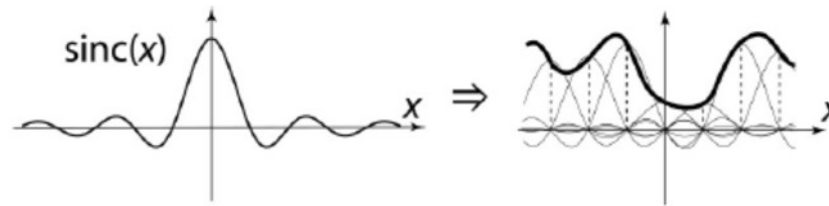
# Image Warping Procedure

For every pixel in the target (warped) image:

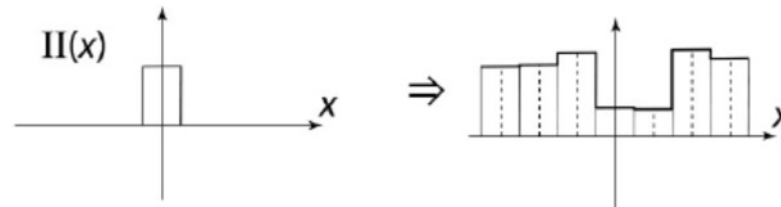
1. Compute location of pixel in input image via the inverse transformation matrix and normalizing.
2. Use interpolation on initial image to compute pixel value (brightness and color).
3. Set pixel value in target image.



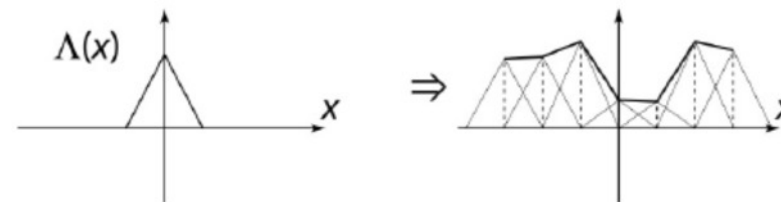
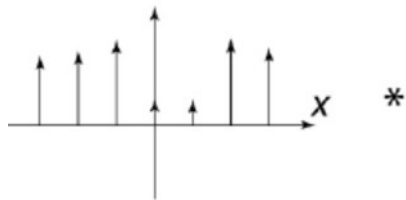
# Review: there are many choices of filter function for image interpolation



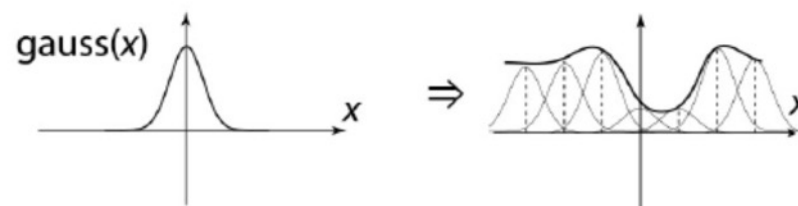
“Ideal” reconstruction  
Infinite Support



Nearest-neighbor  
interpolation  
Finite Support



Linear interpolation  
Finite Support



Gaussian reconstruction  
Infinite Support