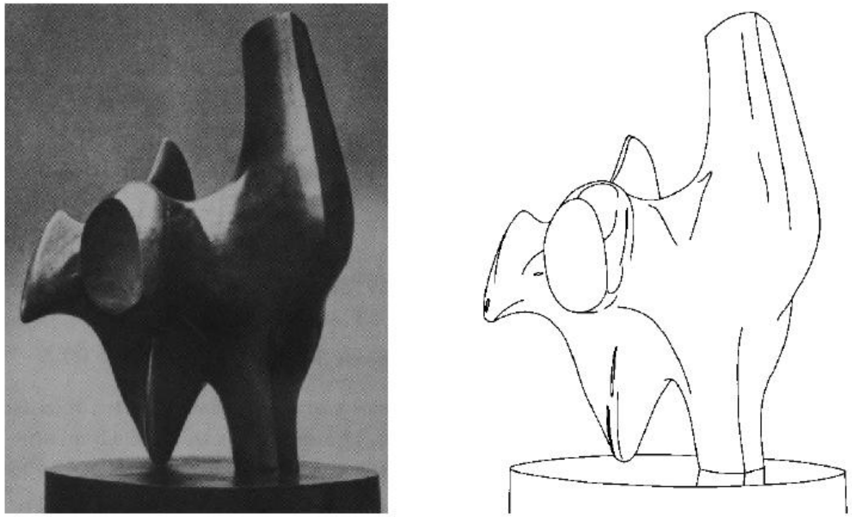
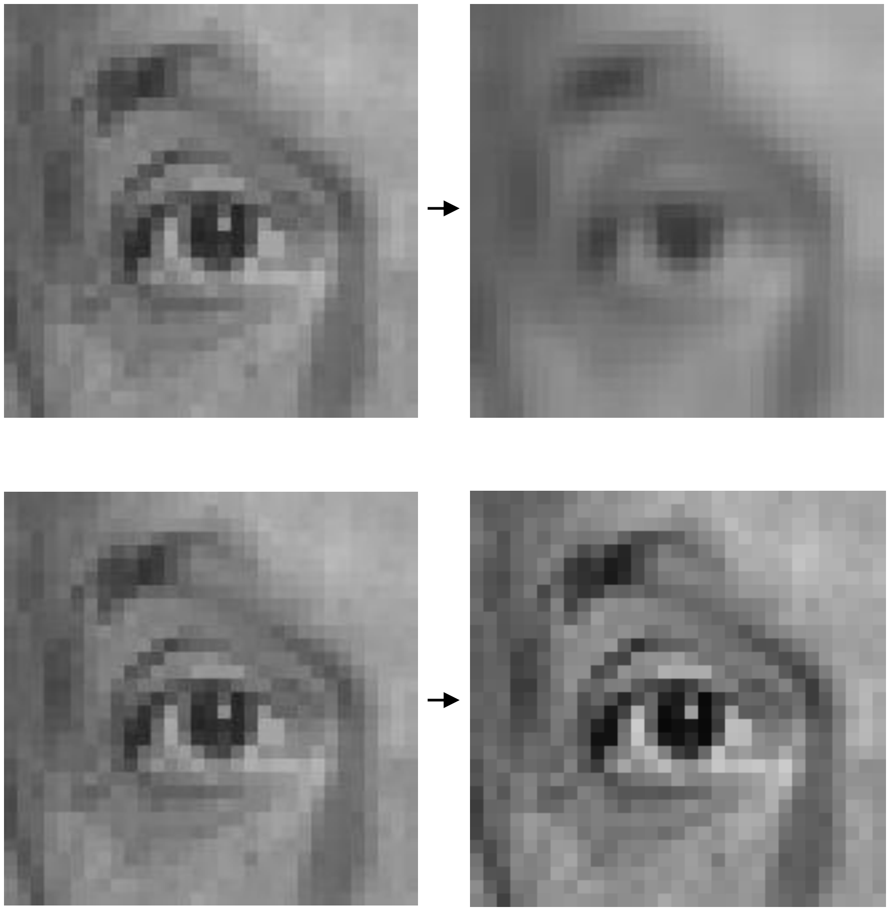
Last Lecture



Edge Detection



Filtering

Pyramid

Today

Motion Deblur

Image Transformation

SIGGRAPH2006

Removing Camera Shake from a Single Photograph

Rob Fergus, Barun Singh, Aaron Hertzmann,
Sam T. Roweis and William T. Freeman

http://people.csail.mit.edu/fergus/research/deblur.html

Massachusetts Institute of Technology
and
University of Toronto

Overview

Original

Our algorithm

Close-up

Original | Naïve Sharpening | Our algorithm

Let’s take a photo

Blurry result

Slow-motion replay

Slow-motion replay

Motion of camera

Image formation process

Blurry image

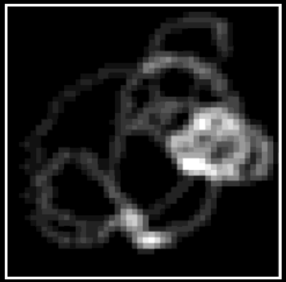
Input to algorithm

=

Sharp image

Desired output

⊗



Blur kernel

Convolution operator

Model is approximation

Why is this hard?

Simple analogy:

11 is the product of two numbers.

What are they?

No unique solution:

11 = 1 x 11

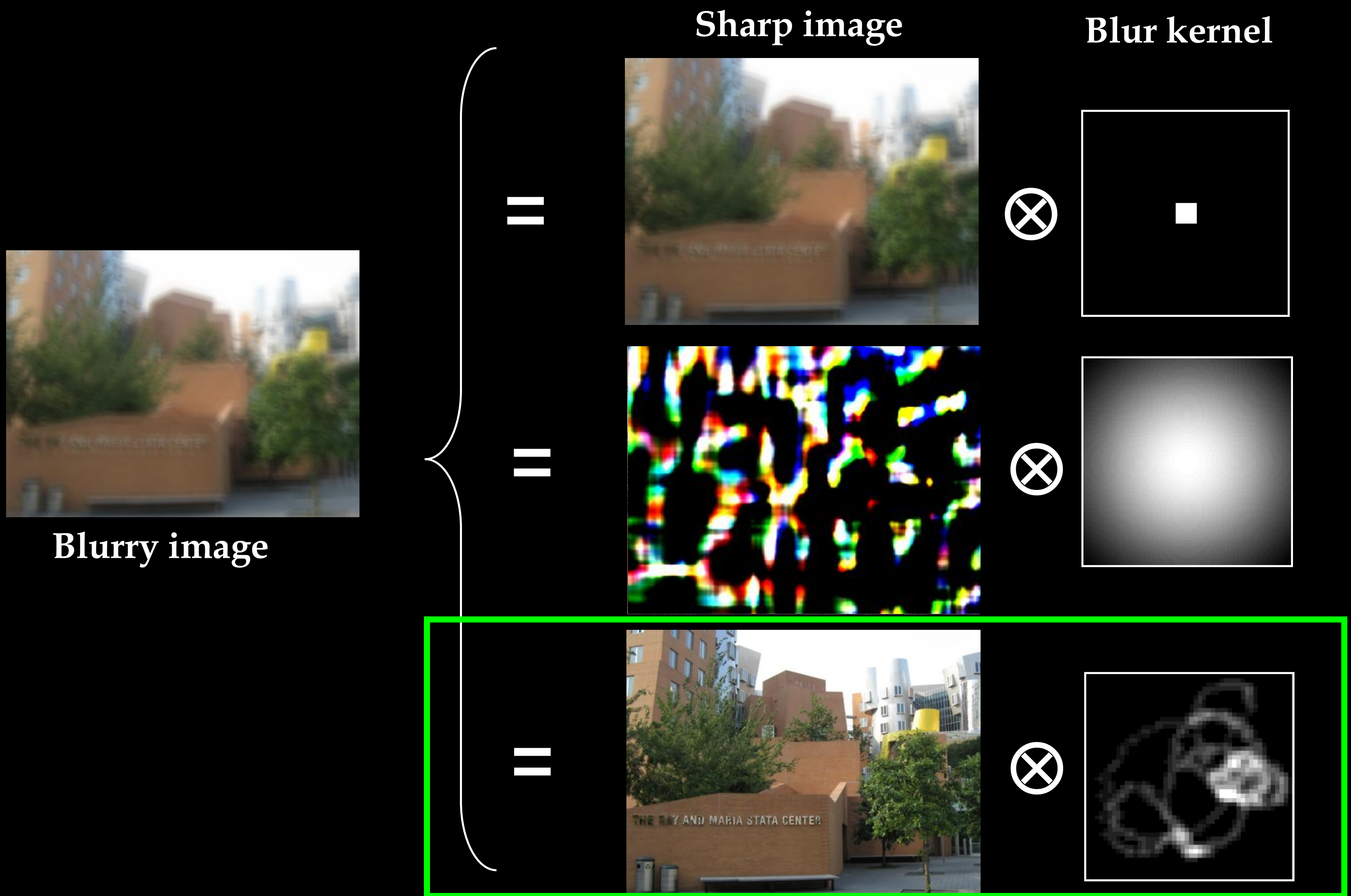
11 = 2 x 5.5

11 = 3 x 3.667

etc…..

Need more information !!!!

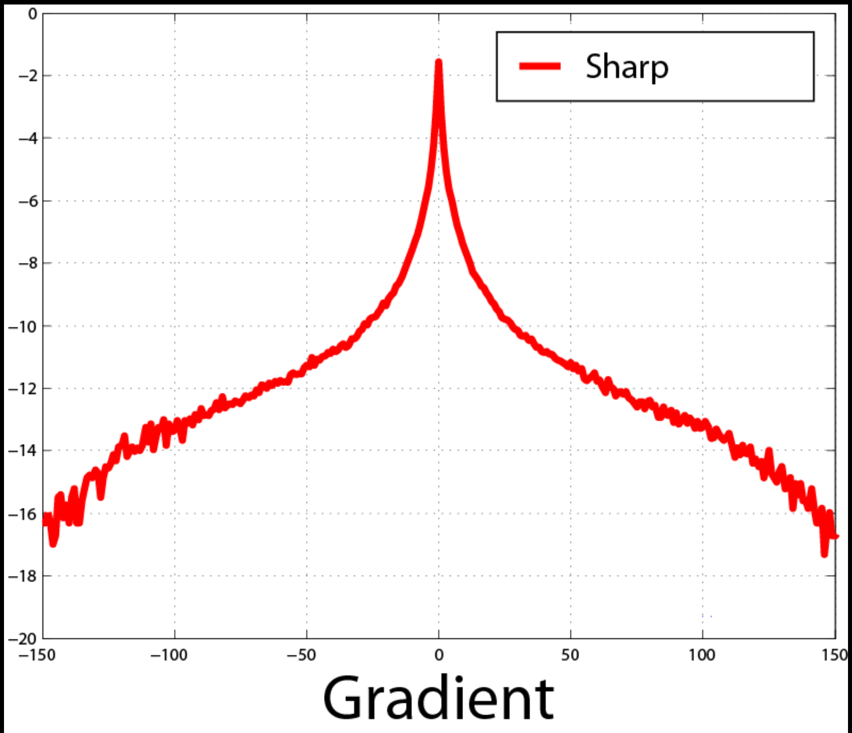
Multiple possible solutions



Natural image statistics

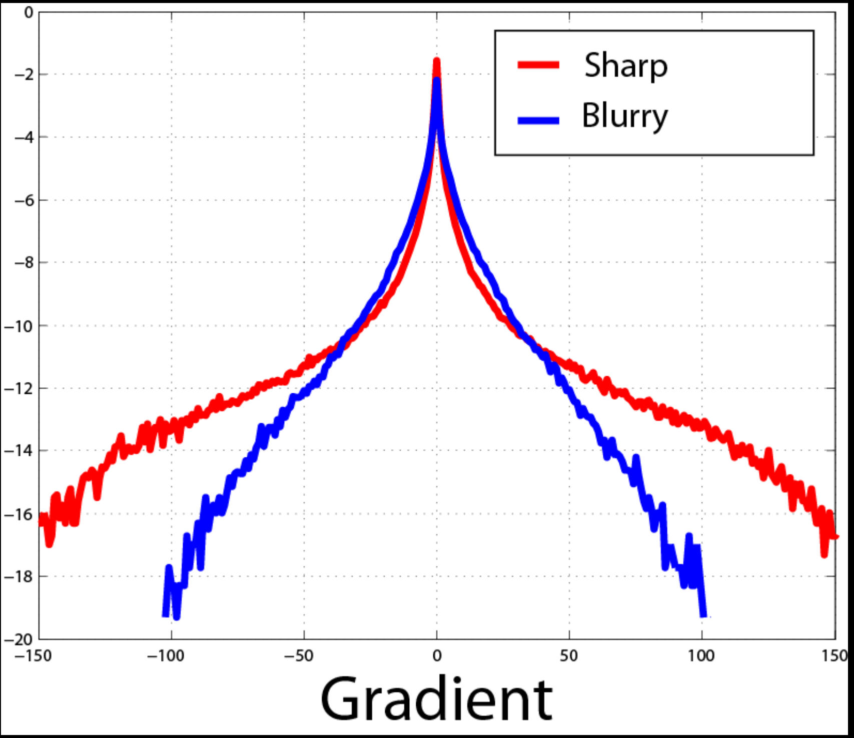
Characteristic distribution with heavy tails

Histogram of image gradients



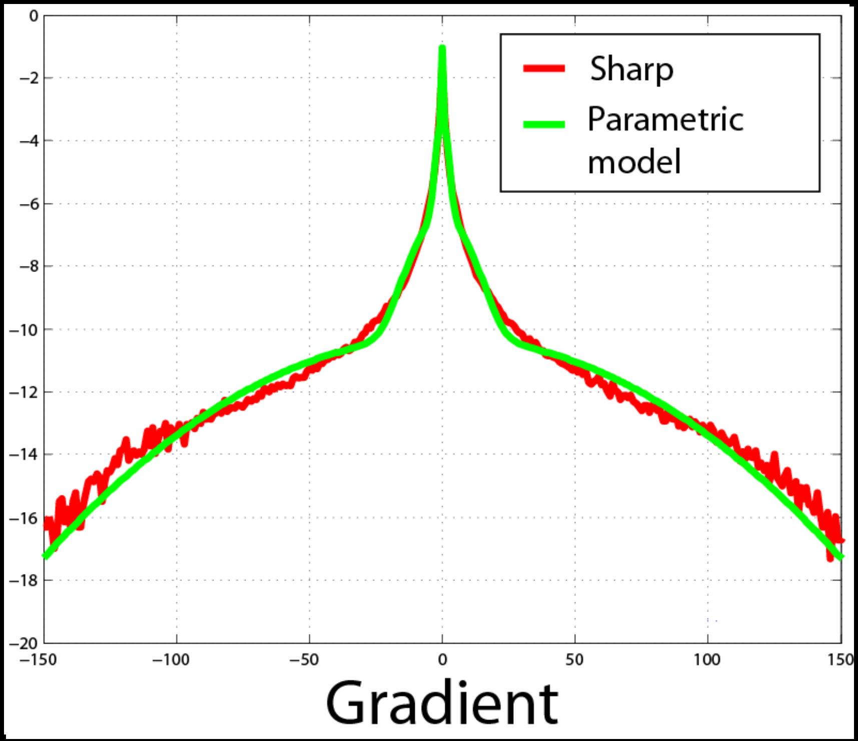
Blury images have different statistics

Histogram of image gradients



Parametric distribution

Histogram of image gradients



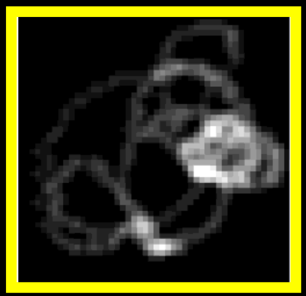
Use parametric model of sharp image statistics

Three sources of information

1. Reconstruction constraint:

Estimated sharp image

$\otimes$

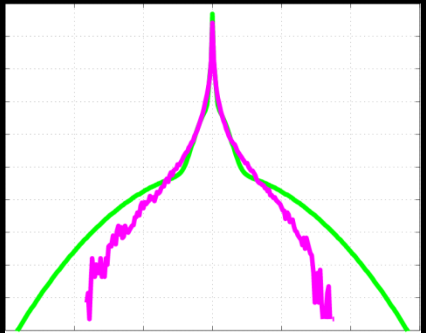


Estimated blur kernel

=

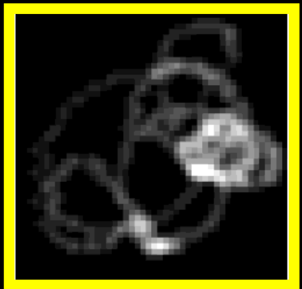
Input blurry image

2. Image prior:



Distribution of gradients

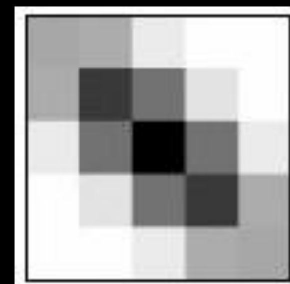
3. Blur prior:



Positive & Sparse

Variational Bayesian method

Based on work of Miskin & Mackay 2000

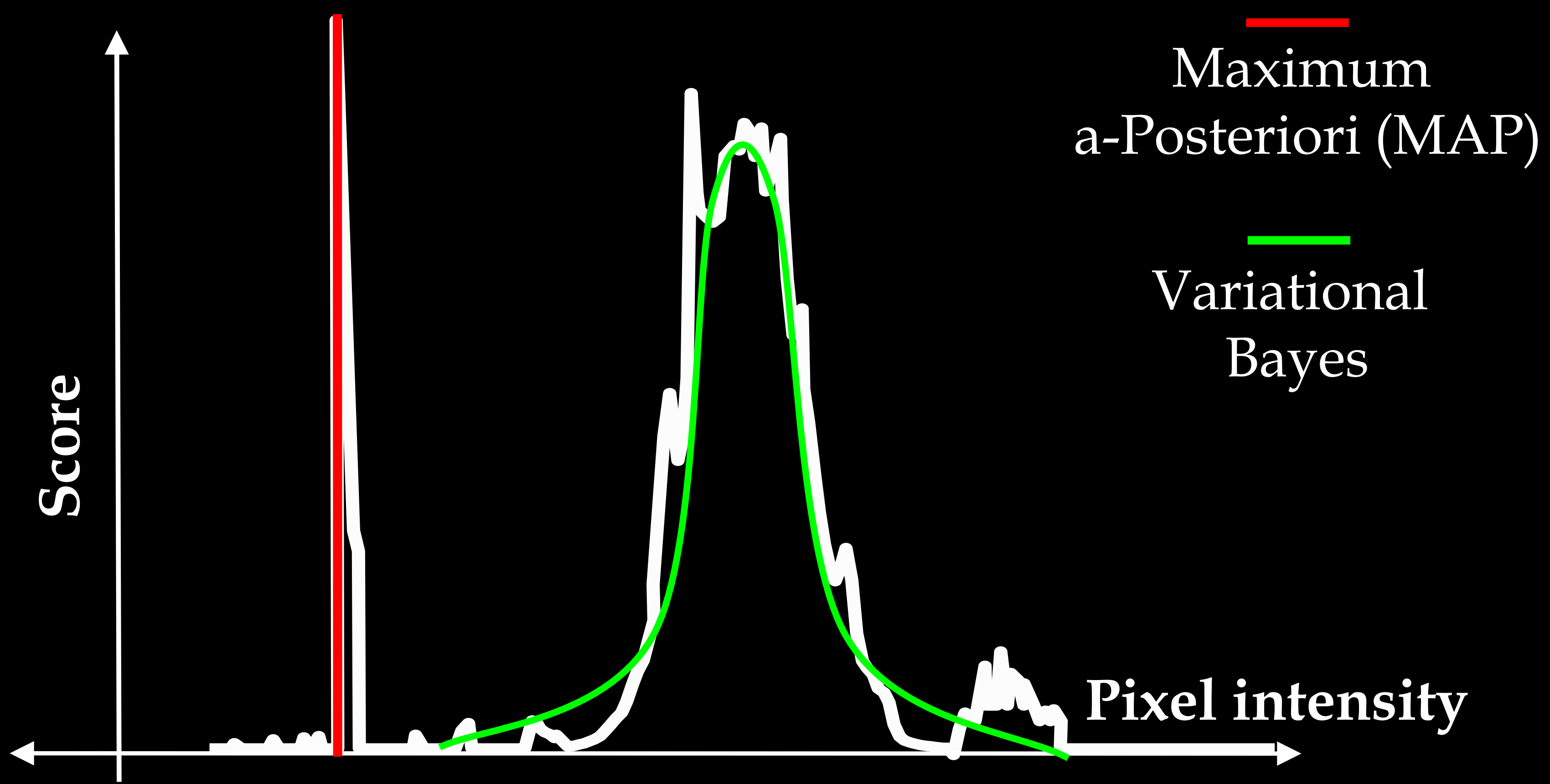


Keeps track of uncertainty in estimates of image and blur by using a distribution instead of a single estimate

Helps avoid local maxima and over-fitting

Variational Bayesian method

Objective function for a single variable



Overview of algorithm

Input image

1. Pre-processing

2. Kernel estimation
   - Multi-scale approach

3. Image reconstruction
   - Standard non-blind deconvolution routine

Preprocessing

Input image

Convert to grayscale → **Remove gamma correction** → **User selects patch from image**

Bayesian inference too slow to run on whole image

Infer kernel from this patch

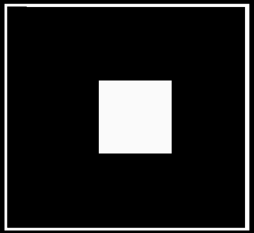
Initialization

Input image

Convert to grayscale → Remove gamma correction → User selects patch from image → **Initialize 3x3 blur kernel**

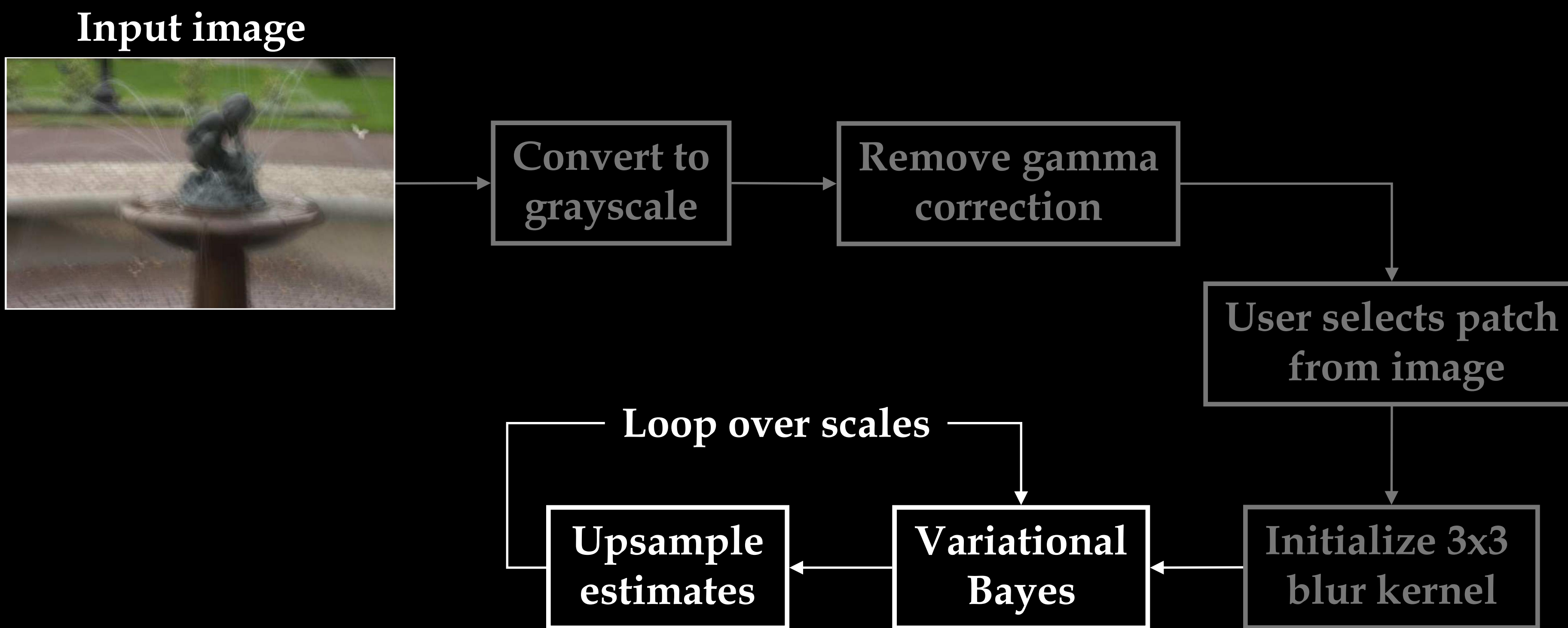
Blurry patch

Initial image estimate



Initial blur kernel

Inferring the kernel: multiscale method



Use multi-scale approach to avoid local minima:

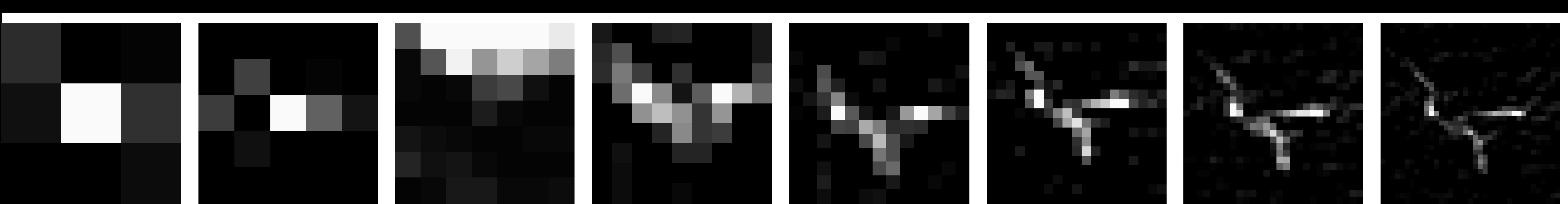
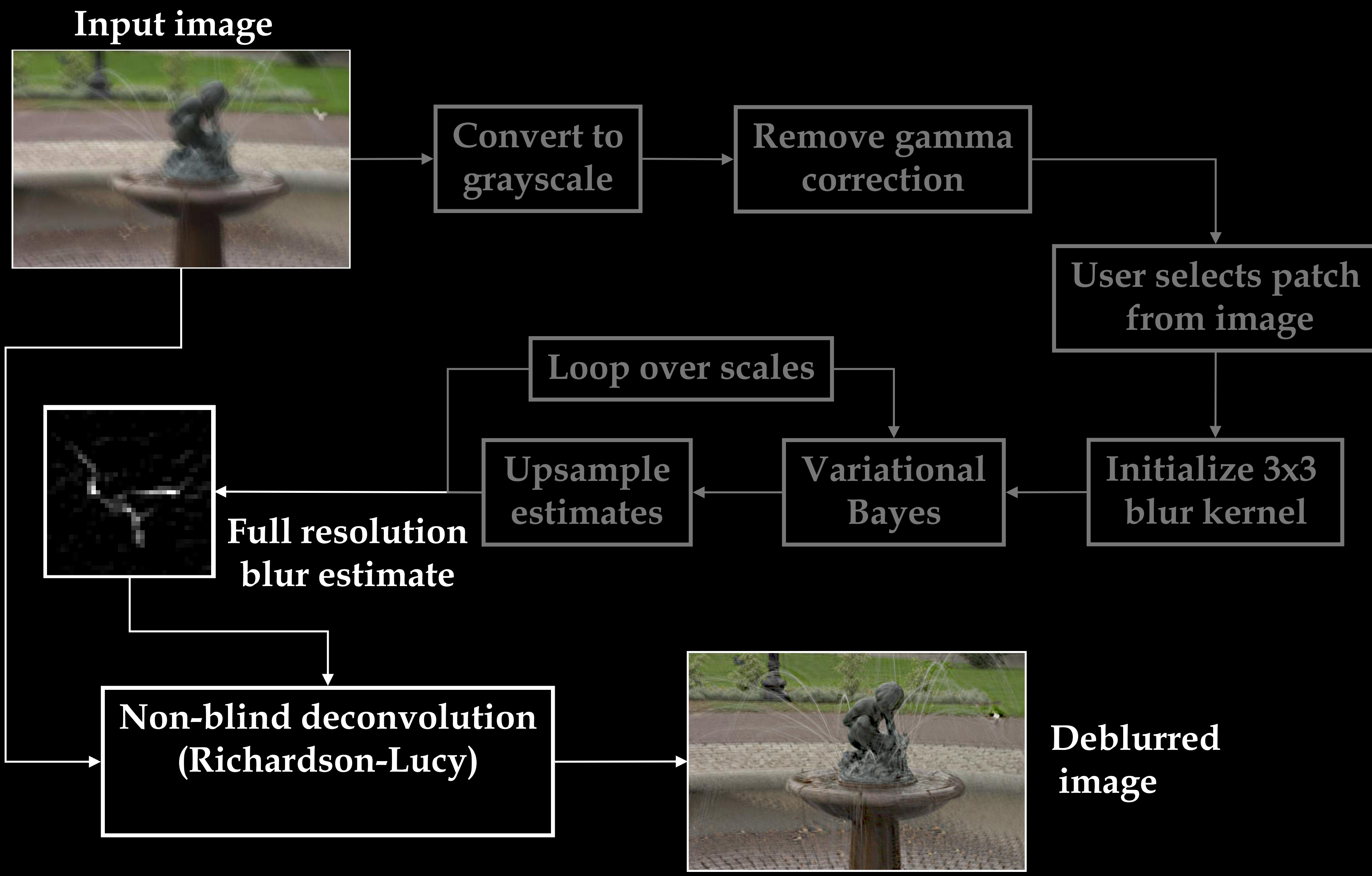


Image Reconstruction



Results on real images

Submitted by people from their own photo collections

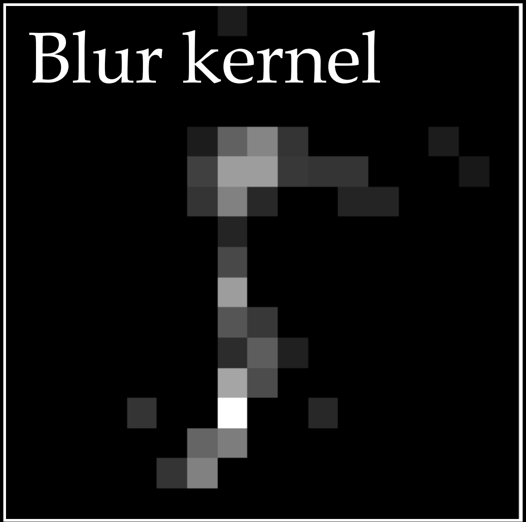
Type of camera unknown

Output does contain artifacts

- Increased noise
- Ringing

Compares well to existing methods

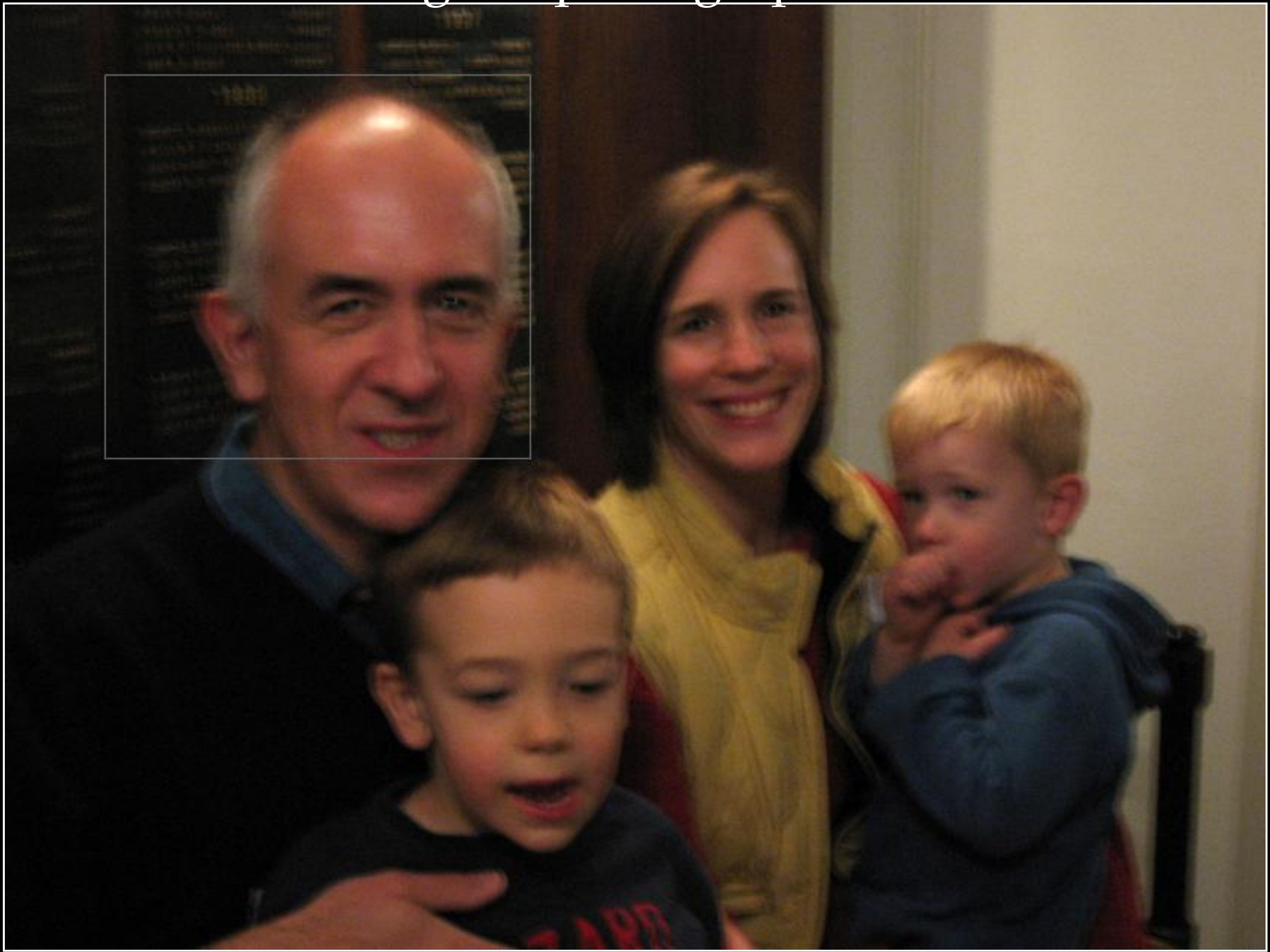
Original photograph


Blur kernel

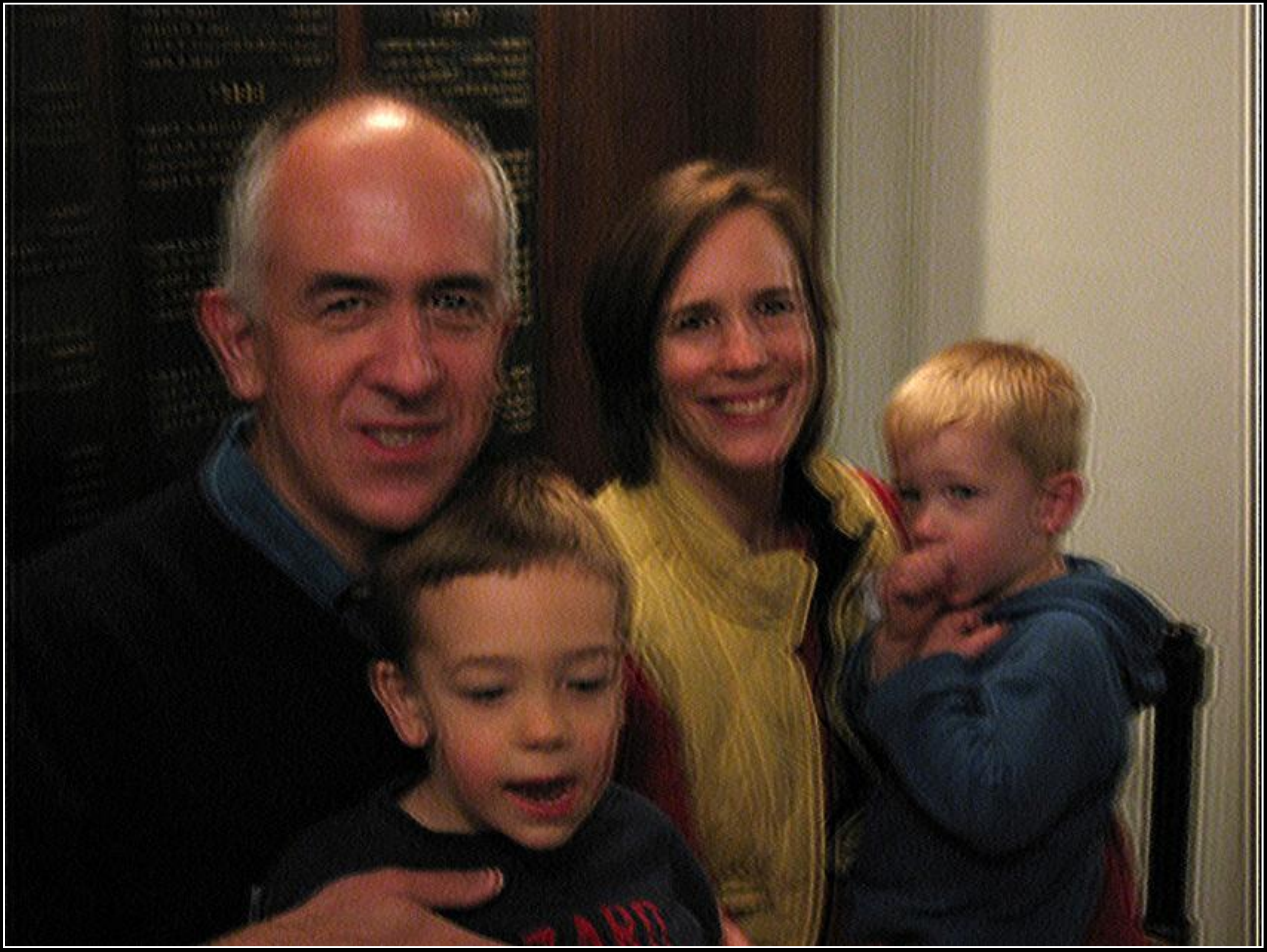

Our output

Matlab's `deconvblind`

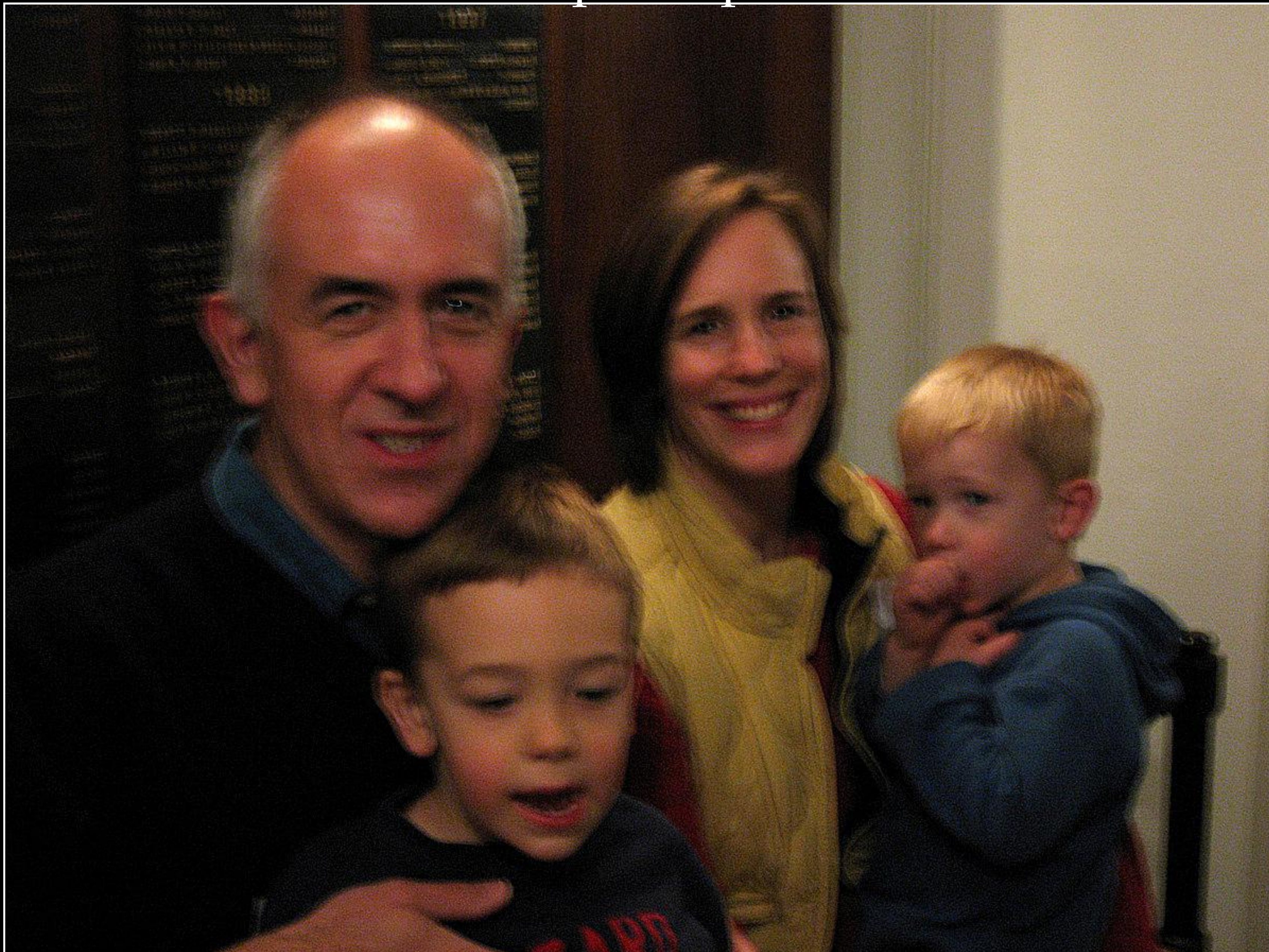
Original photograph



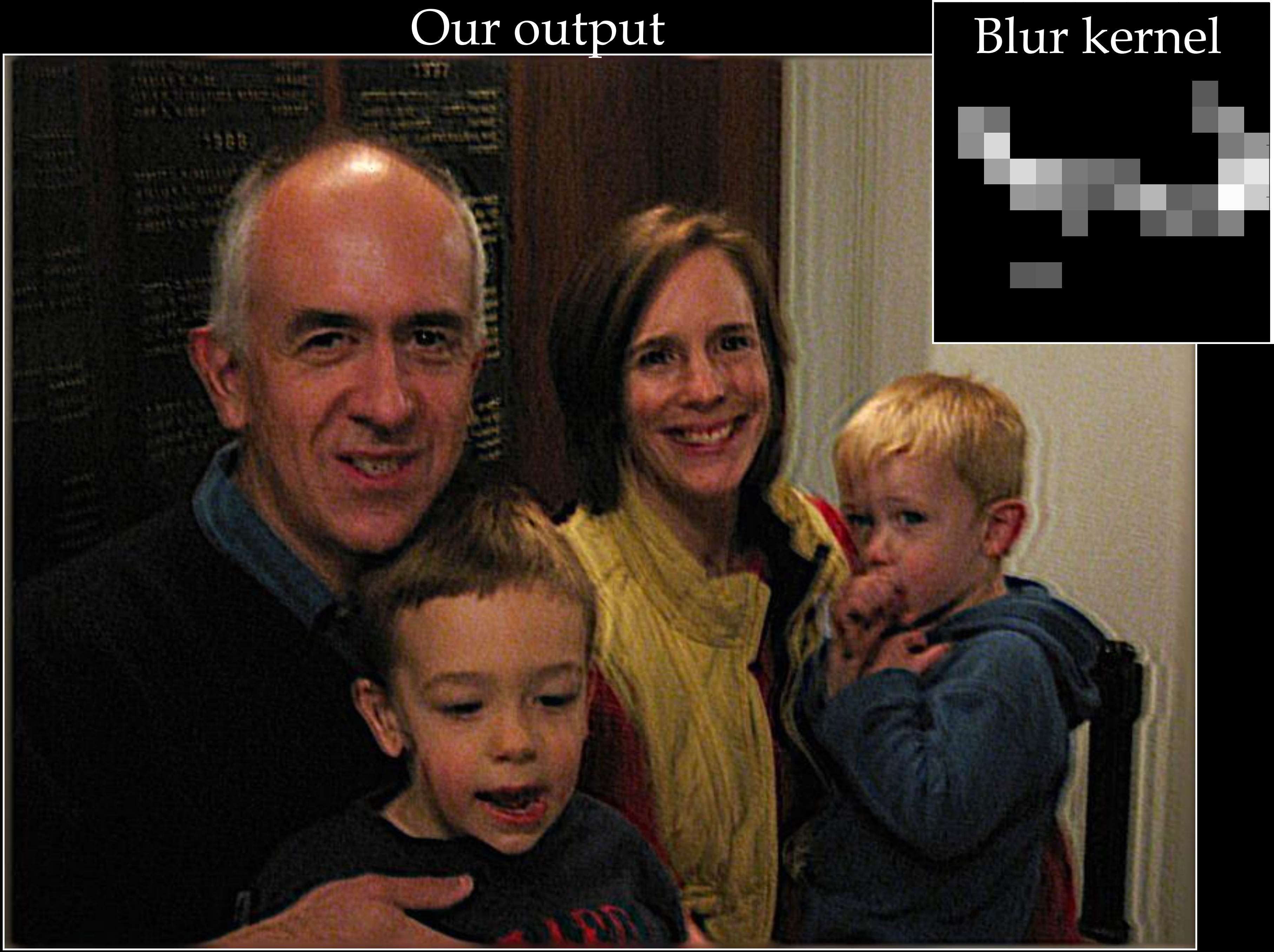
Matlab's `deconvblind`

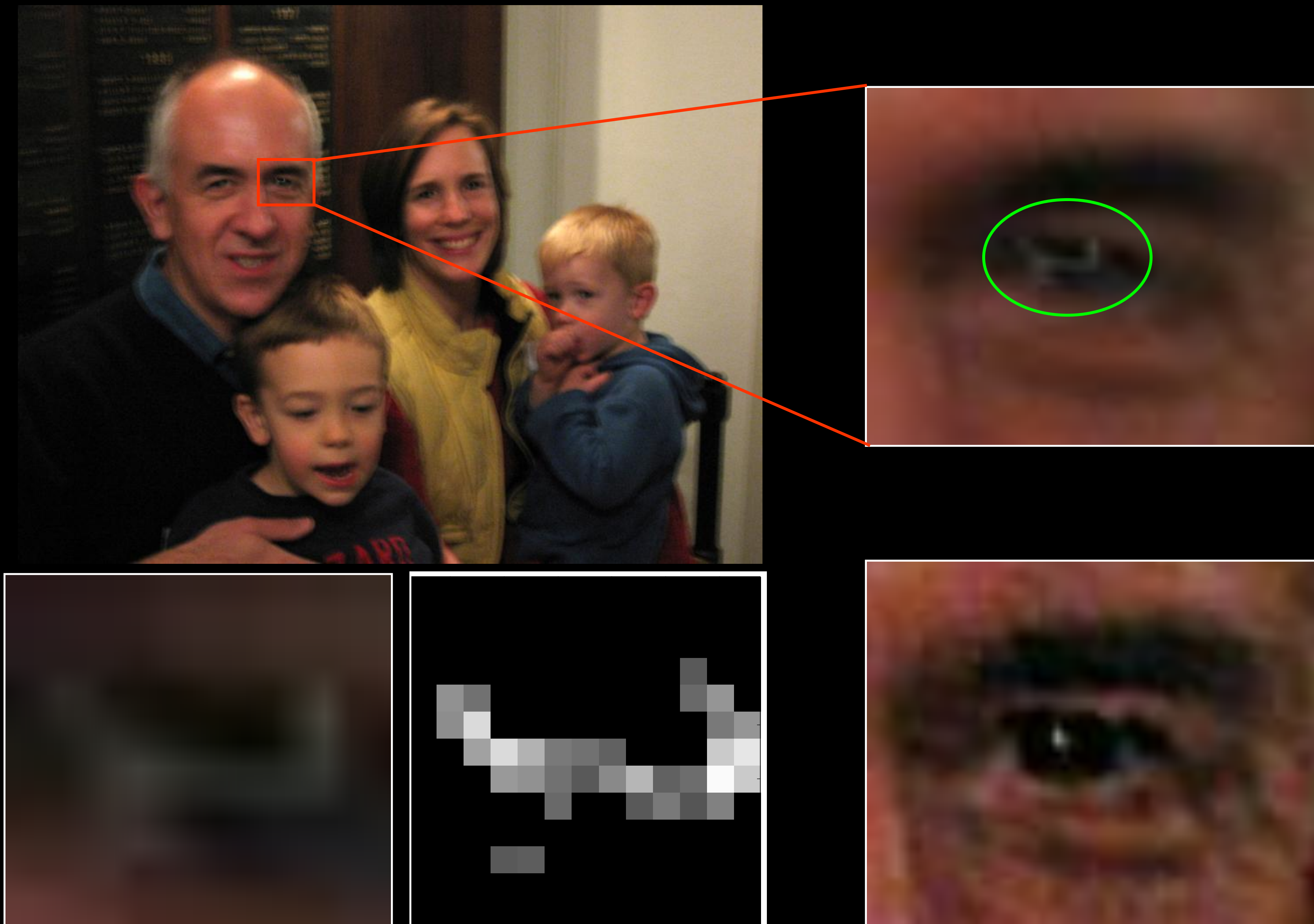


Photoshop sharpen more



Our output





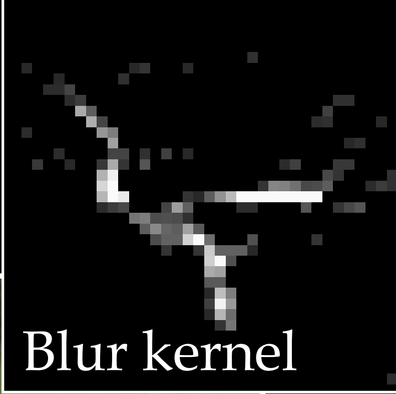
Original photograph

Our output

Blur kernel

Original photograph

Our output



Matlab's `deconvblind`

Original photograph

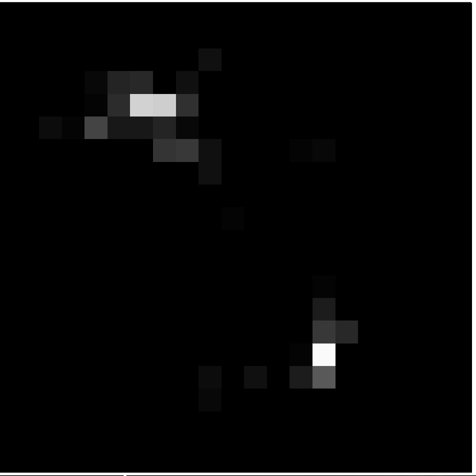

Our output
Blur kernel

Close-up of bird

Original

Our output


Original photograph



Blur kernel

Image artifacts & estimated kernels

Blur kernels

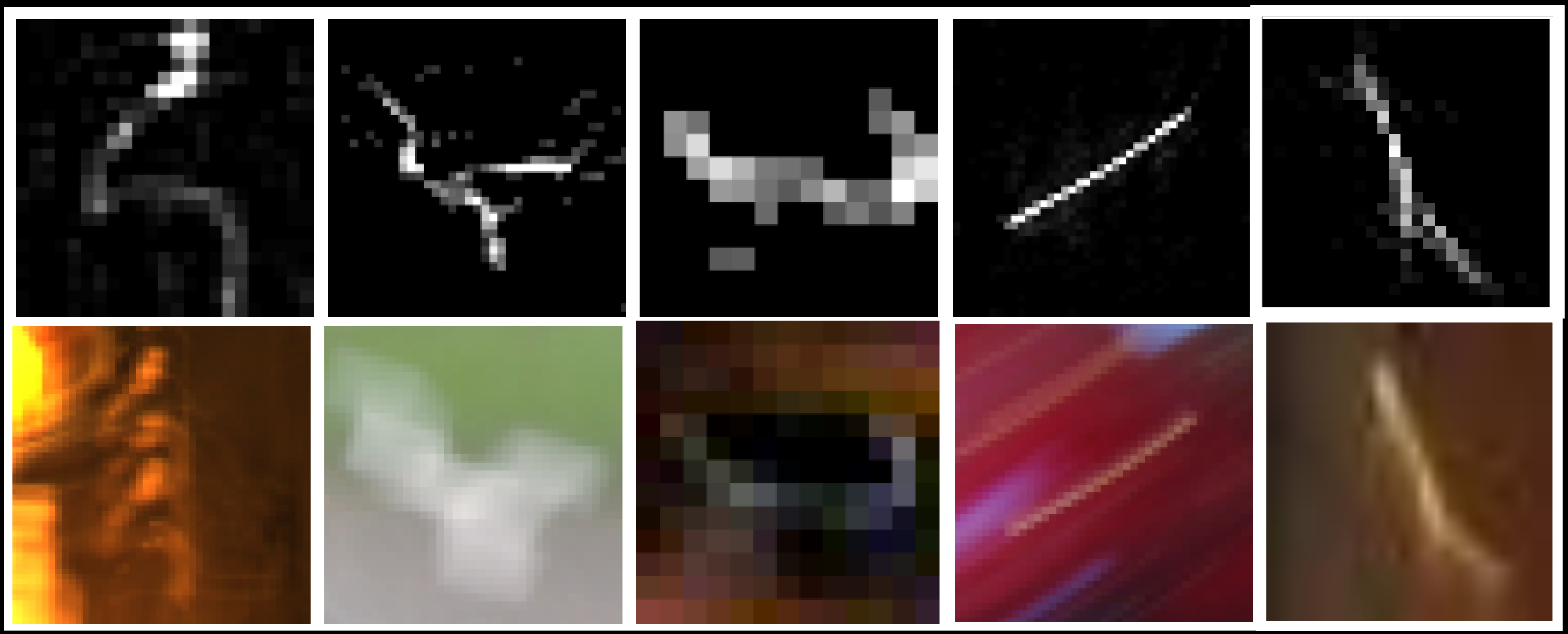


Image patterns

Note: blur kernels were inferred from large image patches, NOT the image patterns shown

Summary

Method for removing camera shake from real photographs

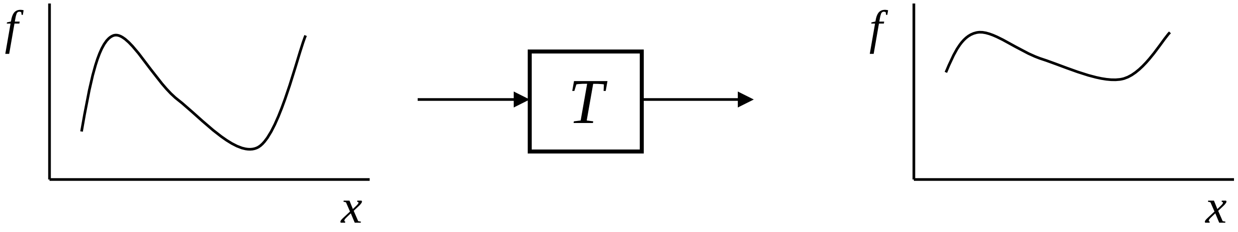
First method that can handle complicated blur kernels

Uses natural image statistics

Non-blind deconvolution currently simplistic

Image Warping

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



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

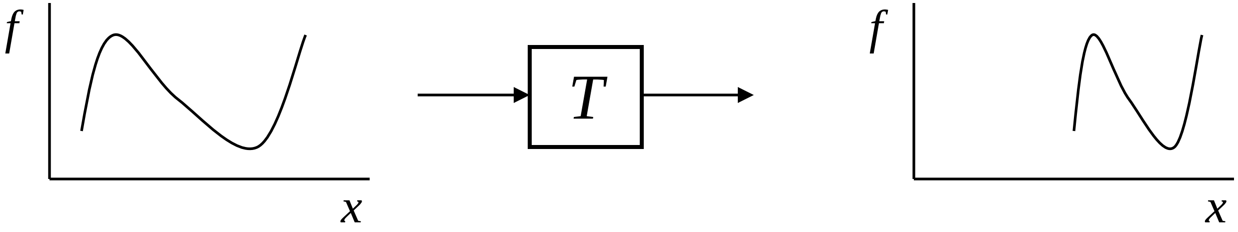
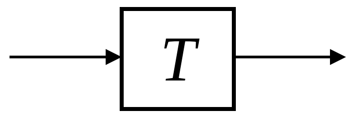
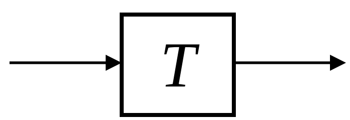


Image Warping

- 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

- Examples of parametric warps:

translation

rotation

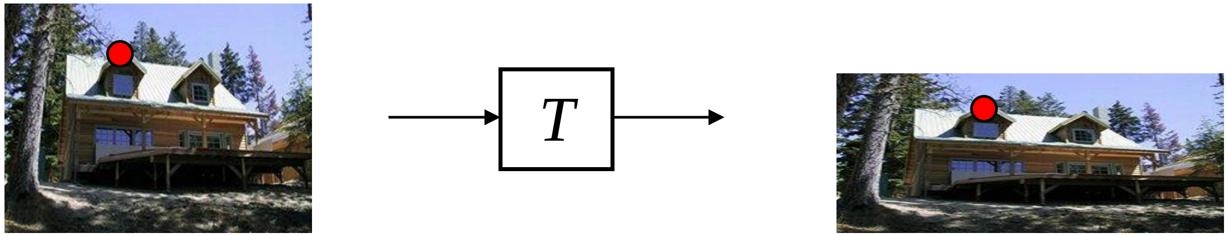
aspect

affine

perspective

cylindrical

Parametric (global) warping



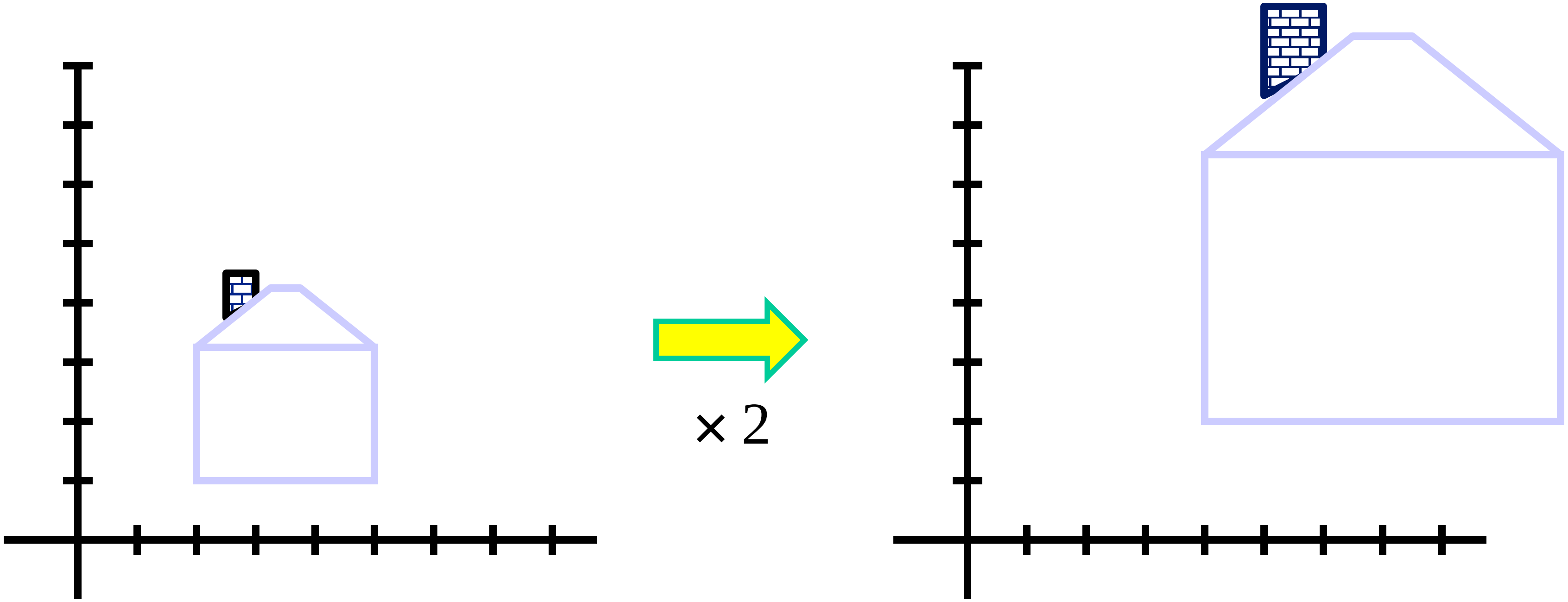
p = (x,y) **p'** = (x',y')

- Transformation T is a coordinate-changing machine:
- p' = *T*(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 represent *T* as a matrix:
- p' = **M**p

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

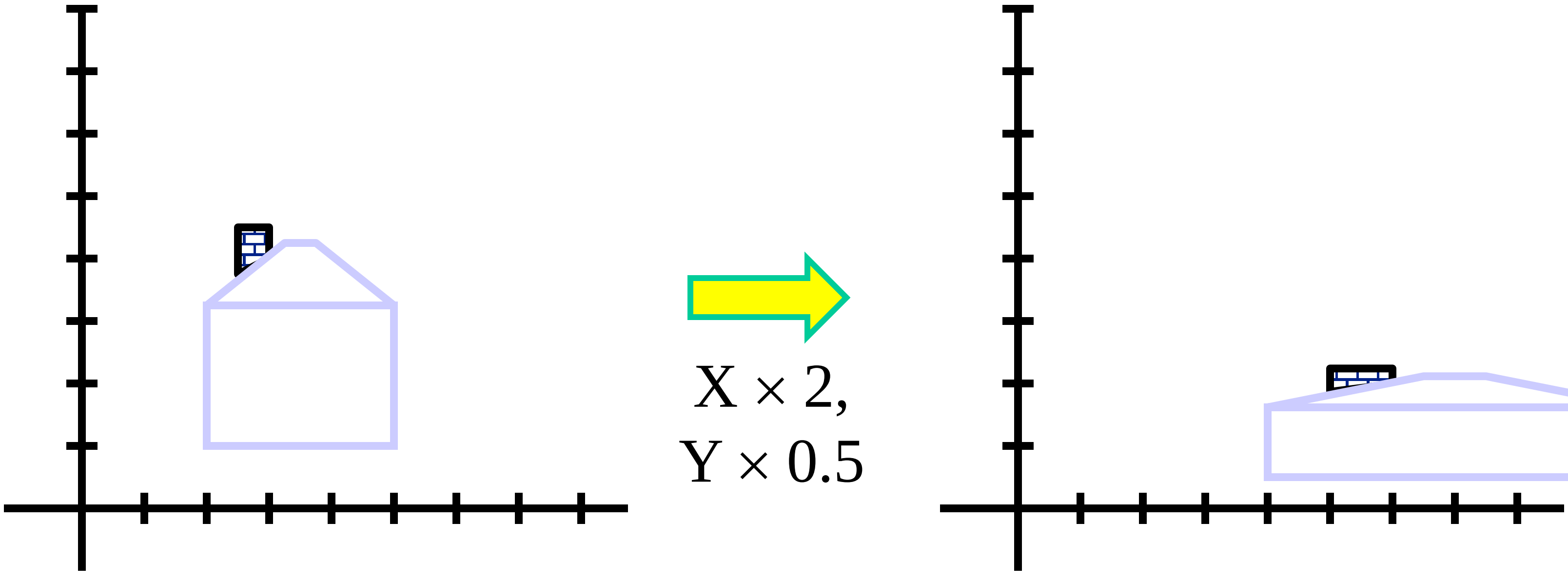
Scaling

- *Scaling* a coordinate means multiplying each of its components by a scalar
- *Uniform scaling* means this scalar is the same for all components:



Scaling

- *Non-uniform scaling*: different scalars per component:



Scaling

- Scaling operation:

$$x' = ax$$
$$y' = by$$

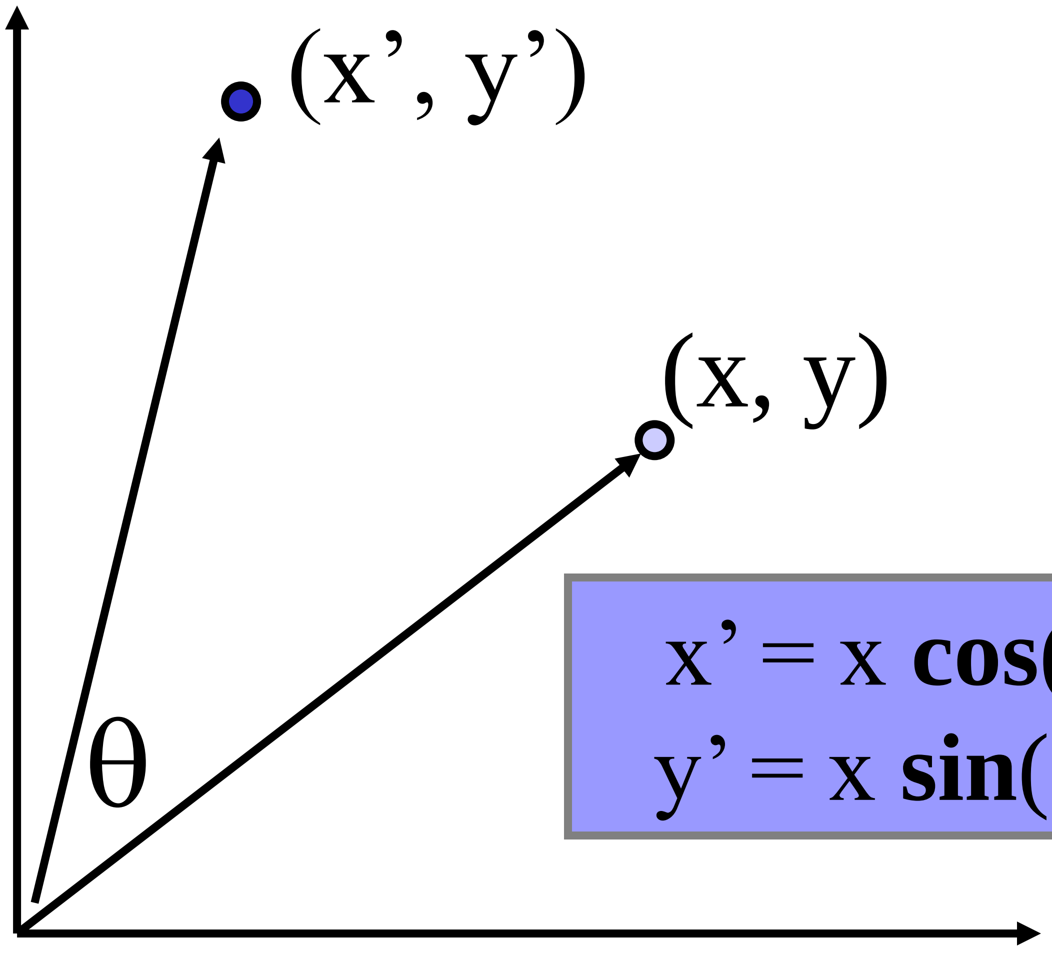
- Or, in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix } S} \begin{bmatrix} x \\ y \end{bmatrix}$$

scaling matrix S

What's inverse of S?

2-D Rotation



$$x' = x\,\mathbf{cos}(\theta) - y\,\mathbf{sin}(\theta)$$
$$y' = x\,\mathbf{sin}(\theta) + y\,\mathbf{cos}(\theta)$$

2-D Rotation

- This is easy to capture in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Even though sin(θ) and cos(θ) are nonlinear functions of θ,
  - ***x’ is a linear combination of x and y***
  - ***y’ is a linear combination of x and y***

- What is the inverse transformation?
  - Rotation by –θ
  - For rotation matrices

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

2x2 Matrices

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

2D Identity?

$$x' = x$$
$$y' = y$$

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

2D Scale around (0,0)?

$$\boldsymbol{x}' = \boldsymbol{s}_x * \boldsymbol{x}$$
$$\boldsymbol{y}' = \boldsymbol{s}_y * \boldsymbol{y}$$

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

2x2 Matrices

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

2D Rotate around (0,0)?

$$x' = \cos\Theta * x - \sin\Theta * y$$
$$y' = \sin\Theta * x + \cos\Theta * y$$

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

2D Shear?

$$\boldsymbol{x' = x + sh_x * y}$$
$$\boldsymbol{y' = sh_y * x + y}$$

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

2x2 Matrices

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

2D Mirror about Y axis?

$$x' = -x$$
$$y' = y$$

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

2D Mirror over (0,0)?

$$x' = -x$$
$$y' = -y$$

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

2x2 Matrices

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

2D Translation?

$$x' = x + t_x$$

$$y' = y + t_y$$

NO!

Only linear 2D transformations can be represented with a 2x2 matrix

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

- **Q: How can we represent translation as a 3x3 matrix?**

$$x' = x + t_x$$

$$y' = y + t_y$$

Homogeneous Coordinates

- ***Homogeneous coordinates***
  - represent coordinates in 2 dimensions with a 3-vector

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

Homogeneous Coordinates

- **Q: How can we represent translation as a 3x3 matrix?**

$$x' = x + t_x$$

$$y' = y + t_y$$

- A: Using the rightmost column:

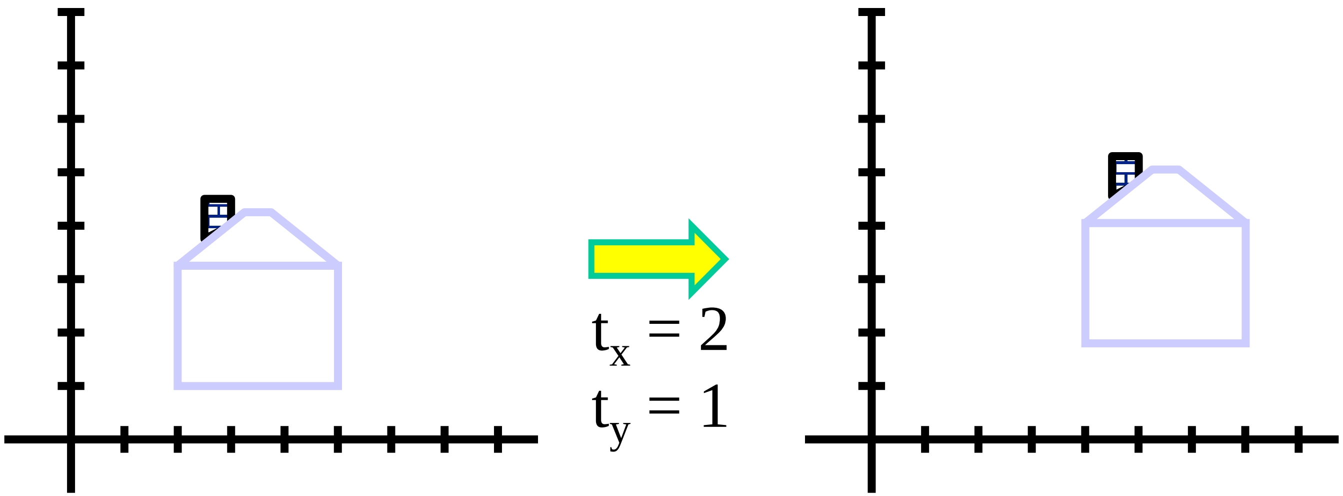
$$\mathbf{T}ranslation = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Translation

- Example of translation

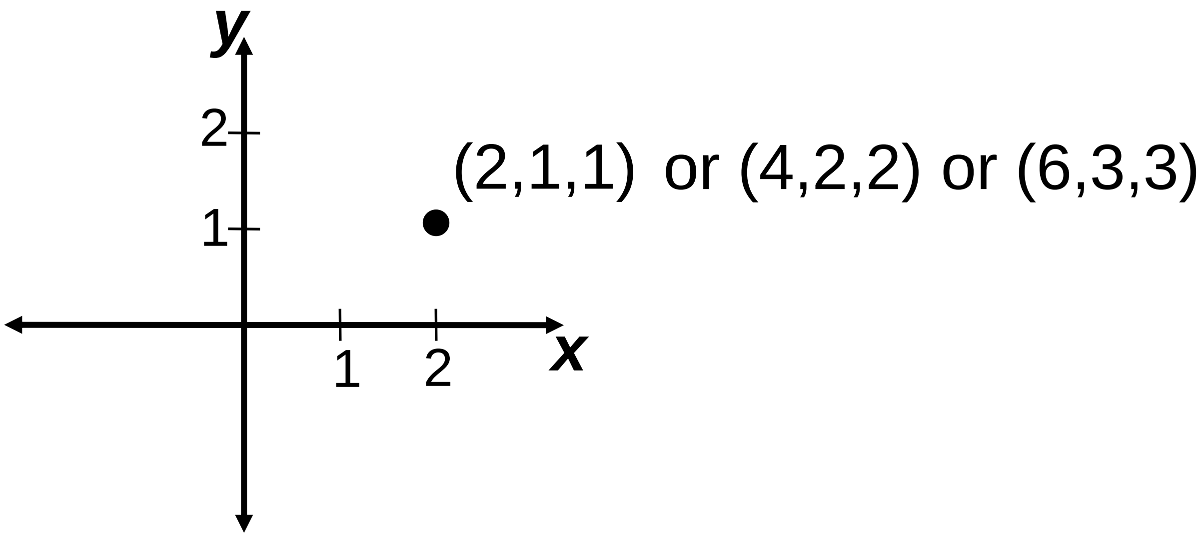
Homogeneous Coordinates

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



Homogeneous Coordinates

- Add a 3rd coordinate to every 2D point
  - (x, y, w) represents a point at location (x/w, y/w)
  - (x, y, 0) represents a point at infinity
  - (0, 0, 0) is not allowed



Convenient coordinate system to represent many useful transformations

Basic 2D Transformations

- Basic 2D transformations as 3x3 matrices

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

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

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

Rotate

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

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

Projective Transformations

- Projective transformations …
  - Affine transformations, and
  - Projective warps

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

- 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

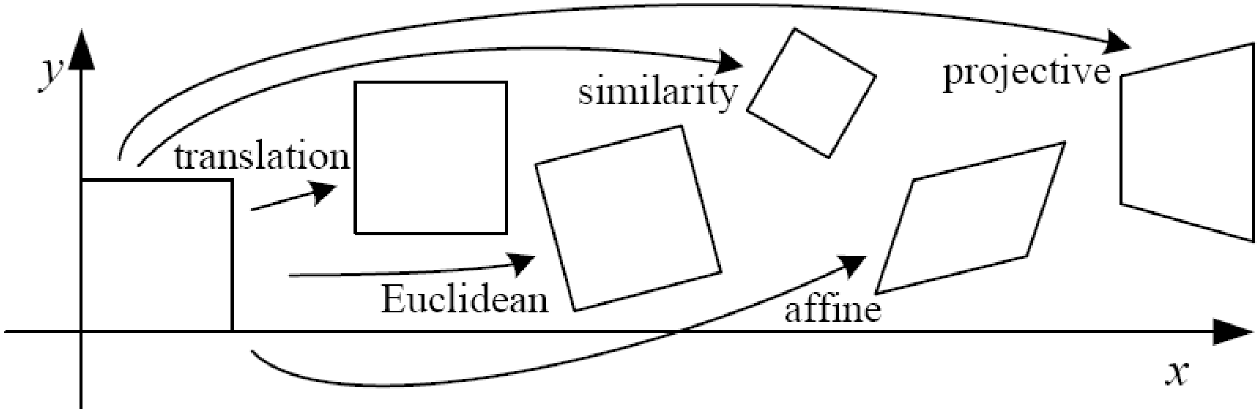
Matrix Composition

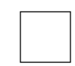
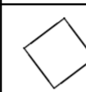
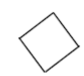
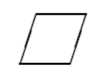
- Transformations can be combined by matrix multiplication

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

$$\mathbf{p'} = T(t_x, t_y) \quad R(\Theta) \quad S(s_x, s_y) \quad \mathbf{p}$$

2D image transformations

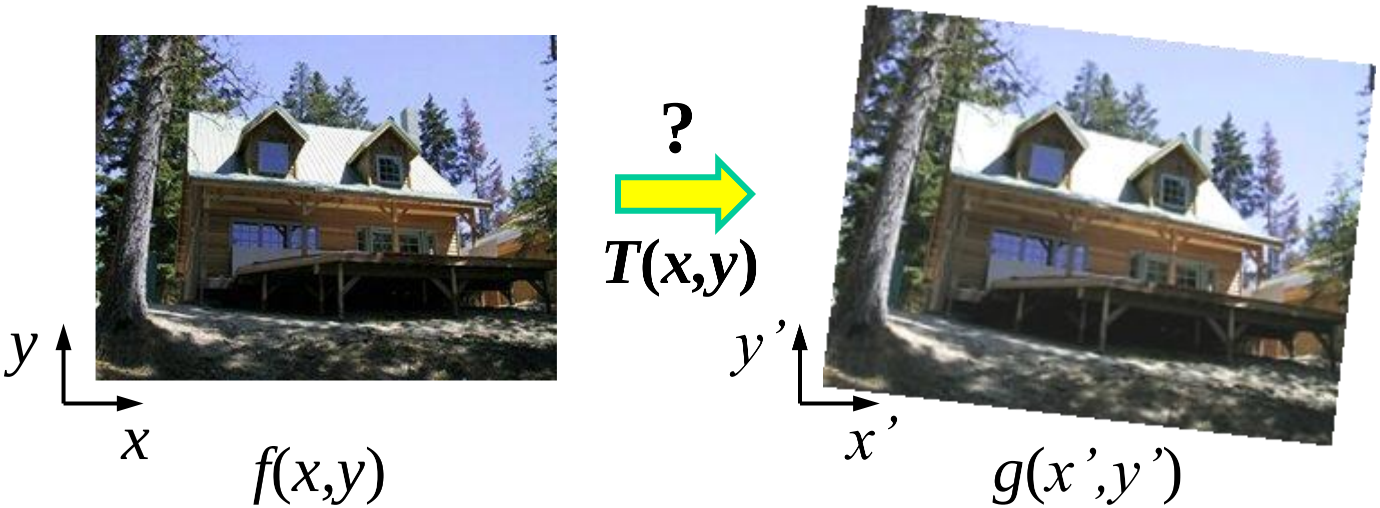


Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times 3}$			
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$			
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$			
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{2\times 3}$			
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{3\times 3}$			

These transformations are a nested set of groups

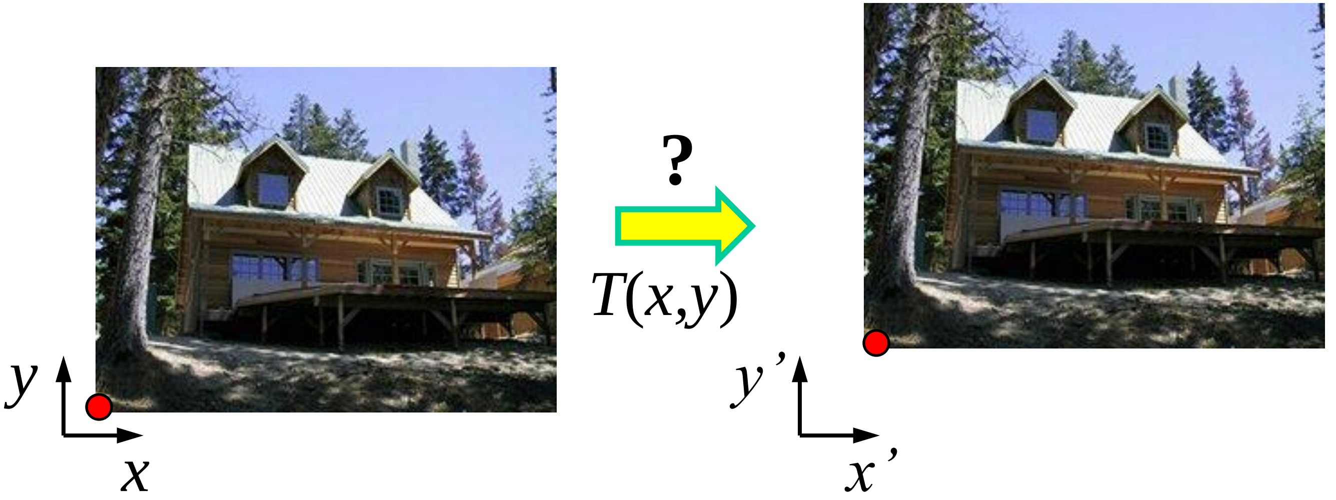
- Closed under composition and inverse is a member

Recovering Transformations



- What if we know *f* and *g* and want to recover the transform T?
  - – Using correspondences
    - How many do we need?

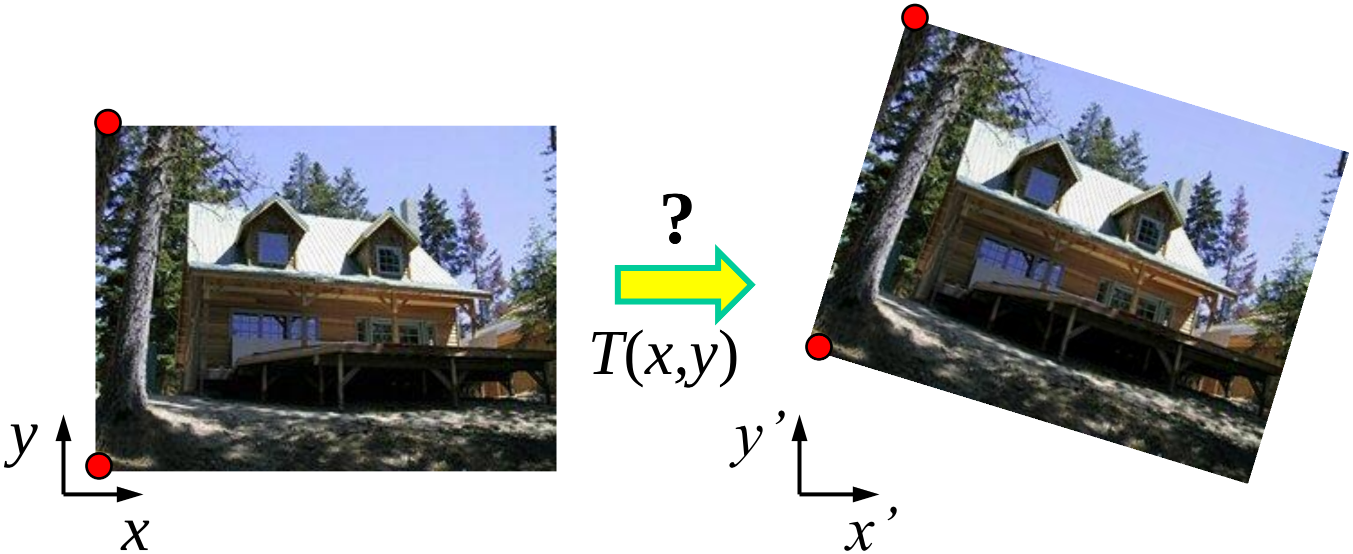
Translation: # correspondences?



- How many correspondences needed for translation?
- How many Degrees of Freedom?
- What is the transformation matrix?

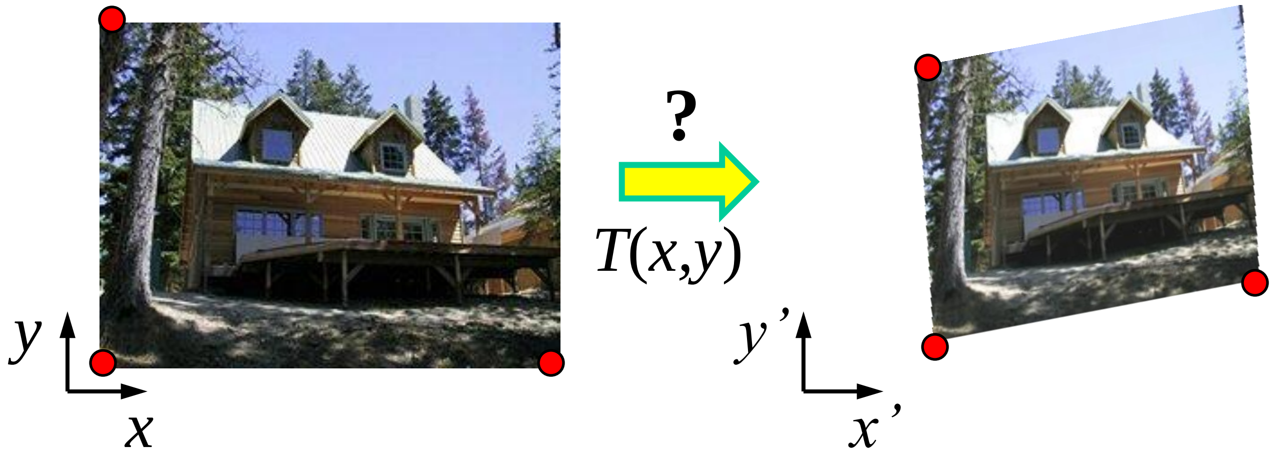
$$\mathbf{M} = \begin{bmatrix} 1 & 0 & p'_x - p_x \\ 0 & 1 & p'_y - p_y \\ 0 & 0 & 1 \end{bmatrix}$$

Euclidian: # correspondences?



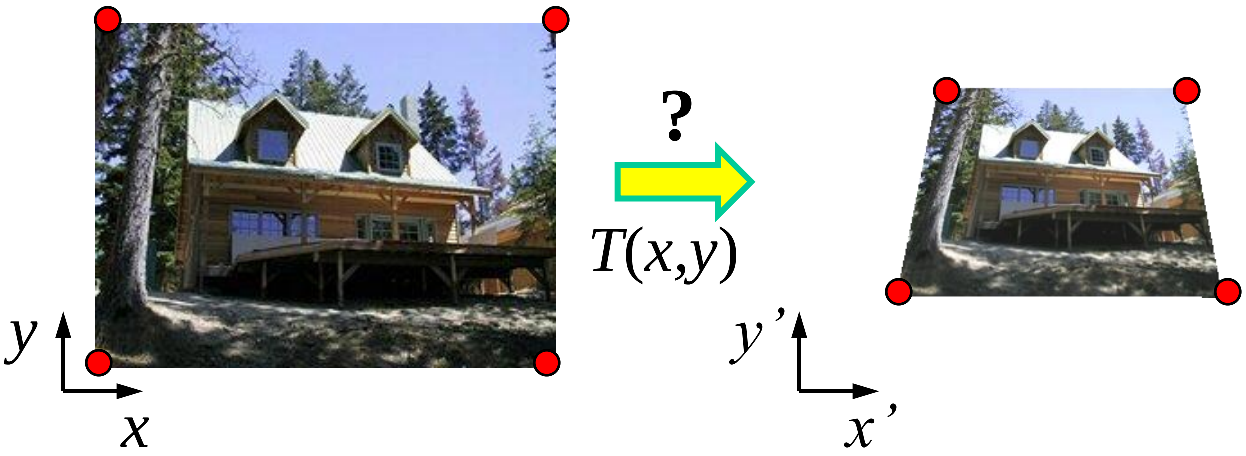
- How many correspondences needed for translation+rotation?
- How many DOF?

Affine: # correspondences?



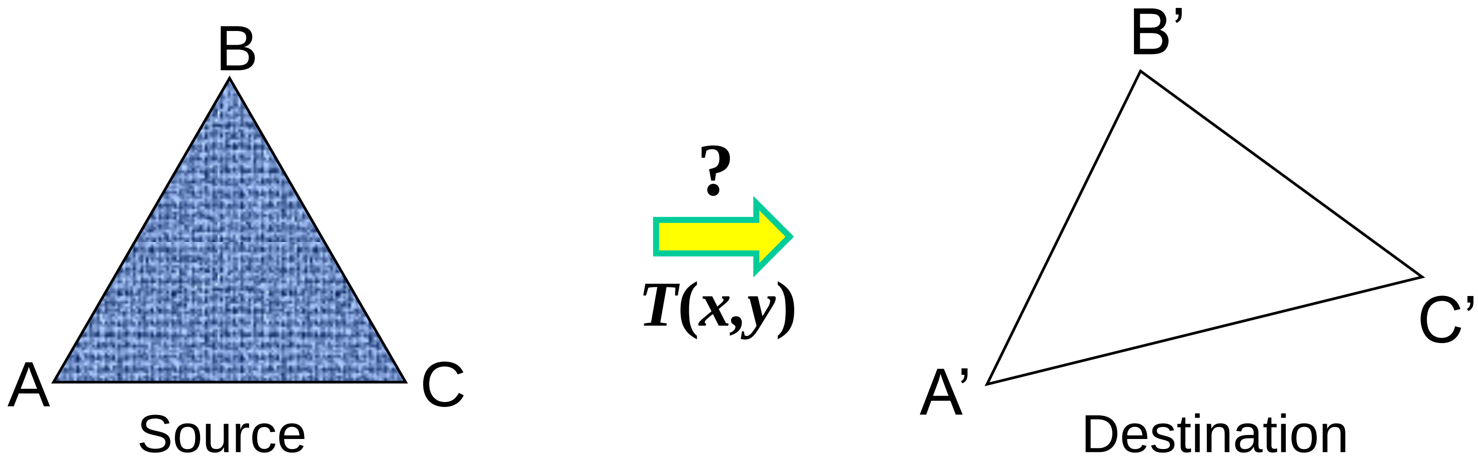
- How many correspondences needed for affine?
- How many DOF?

Projective: # correspondences?



- How many correspondences needed for projective?
- How many DOF?

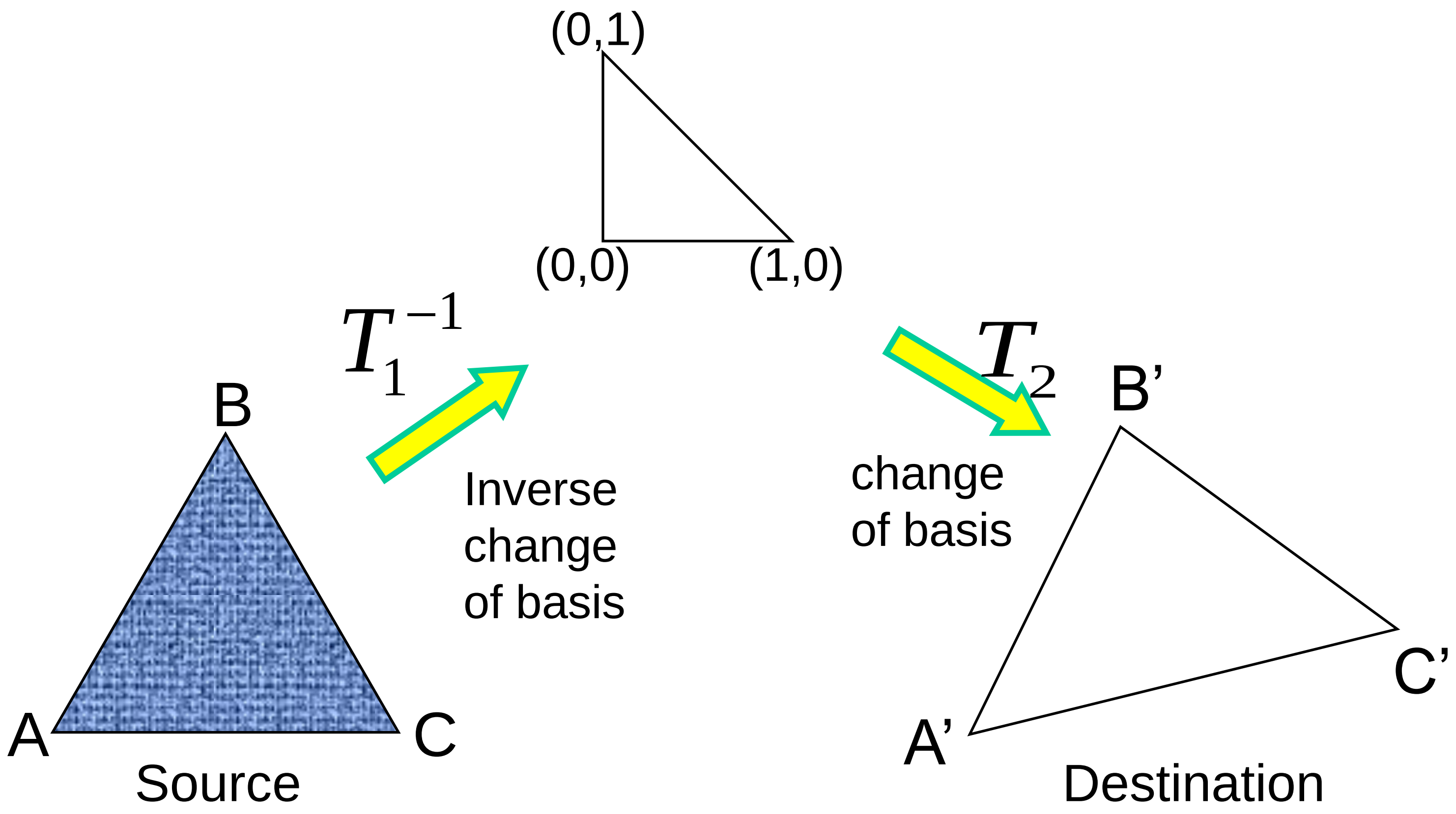
Example: warping triangles



- Given two triangles: ABC and A'B'C' in 2D (12 numbers)
- Need to find transform T to transfer all pixels from one to the other.
- What kind of transformation is T?
- How can we compute the transformation matrix:

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

warping triangles (Barycentric Coordinaes)



Don't forget to move the origin too!

- Very useful in Graphics…

Image morphing

- The goal is to synthesize a fluid transformation from one image to another.
- Cross dissolving is a common transition between cuts, but it is not good for morphing because of the ghosting effects.

image #1

dissolving

image #2

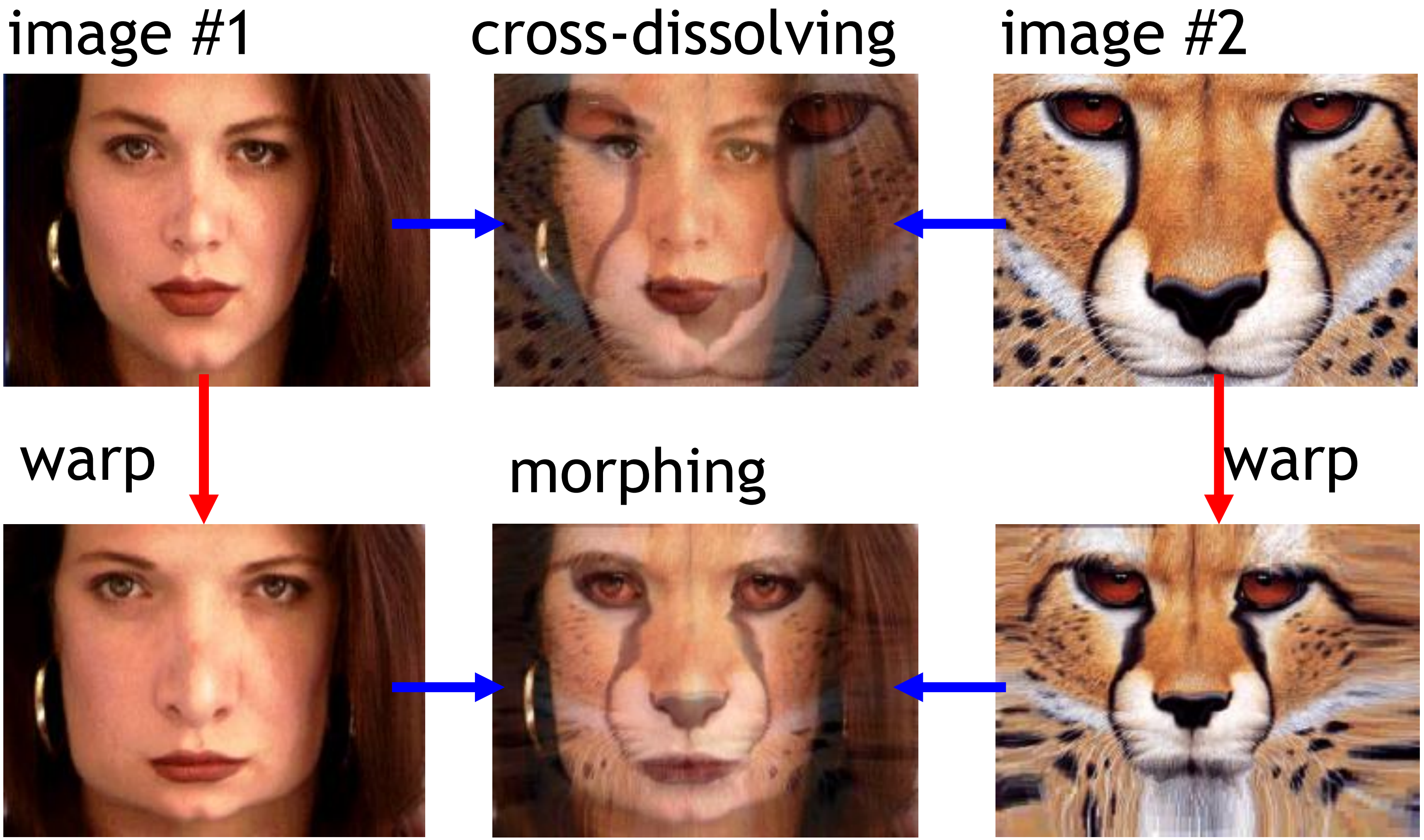
Image morphing

- Why ghosting?
- Morphing = warping + cross-dissolving

shape (geometric) → warping

color (photometric) → cross-dissolving

Image morphing



Morphing sequence

Warp by triangulation

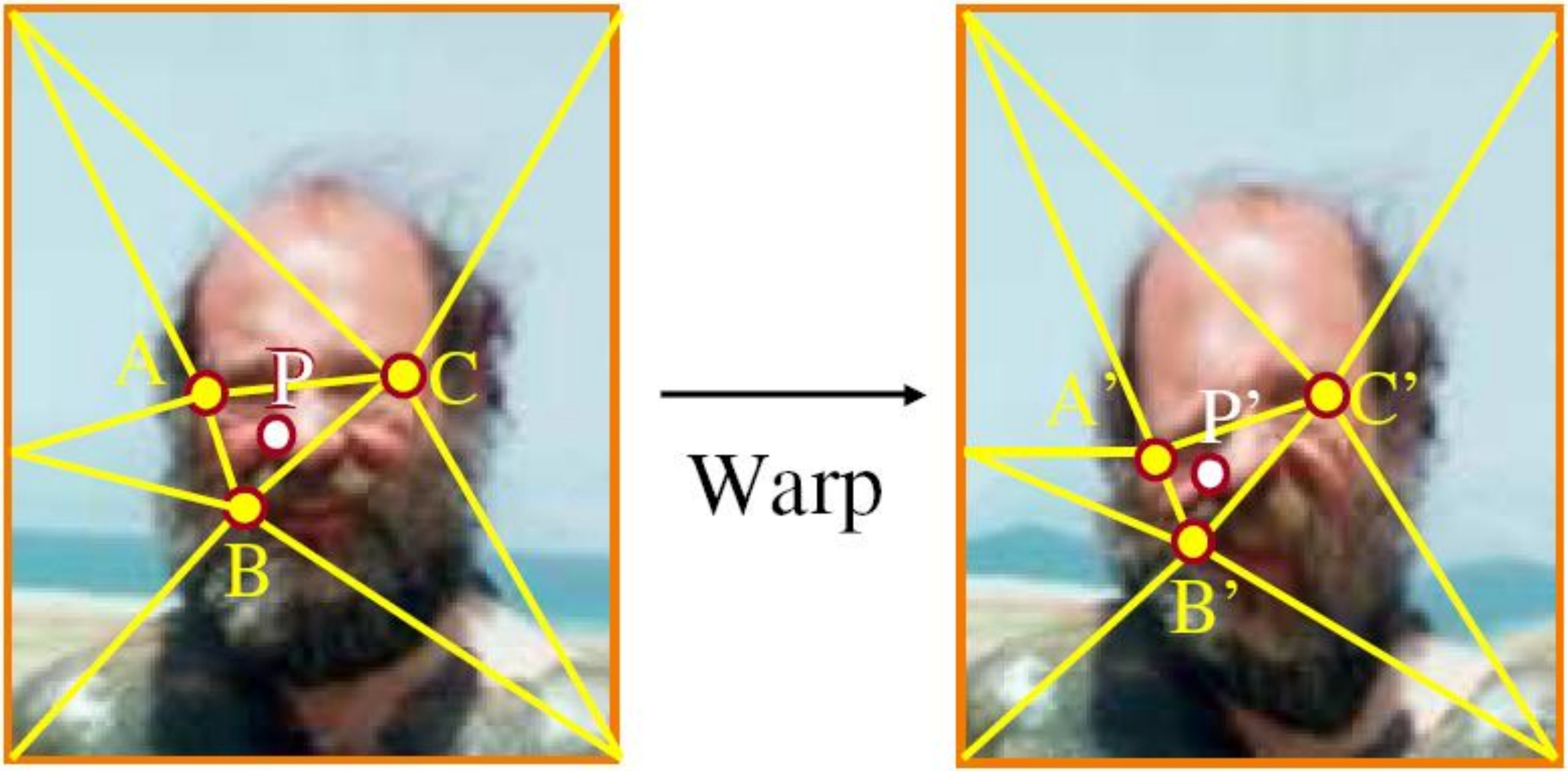
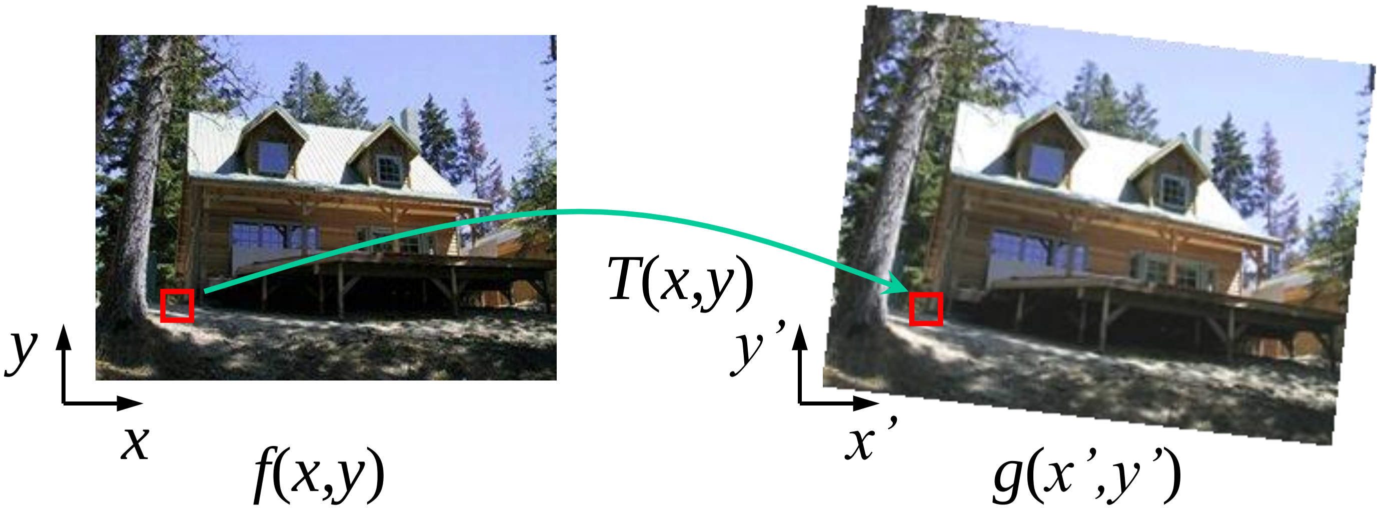
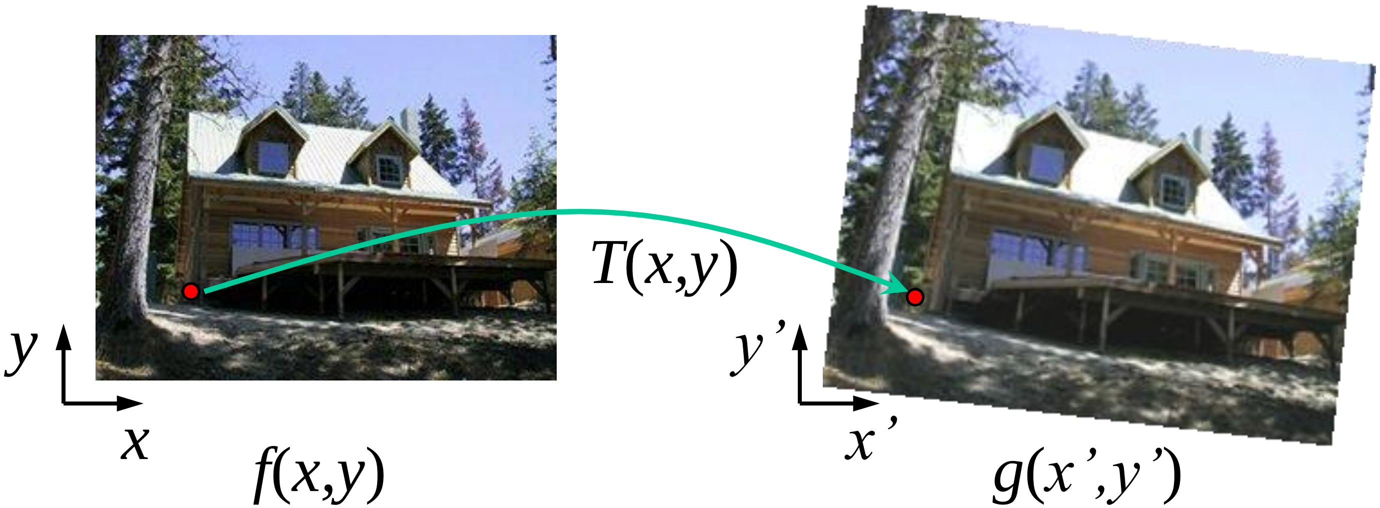


Image warping



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

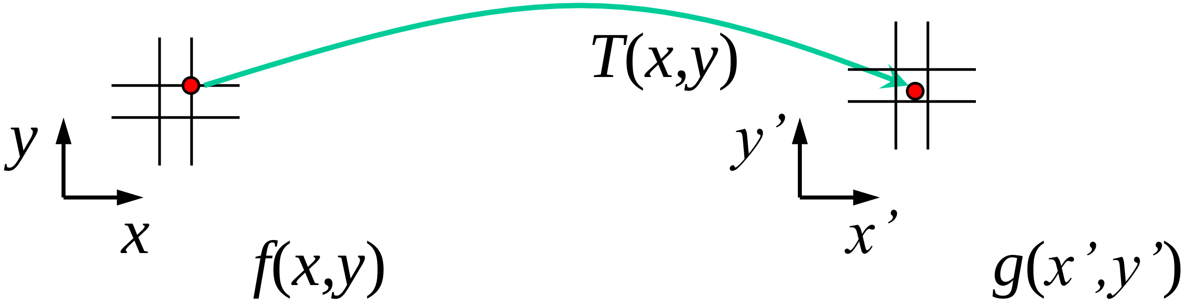
Forward warping



- 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?

Forward warping



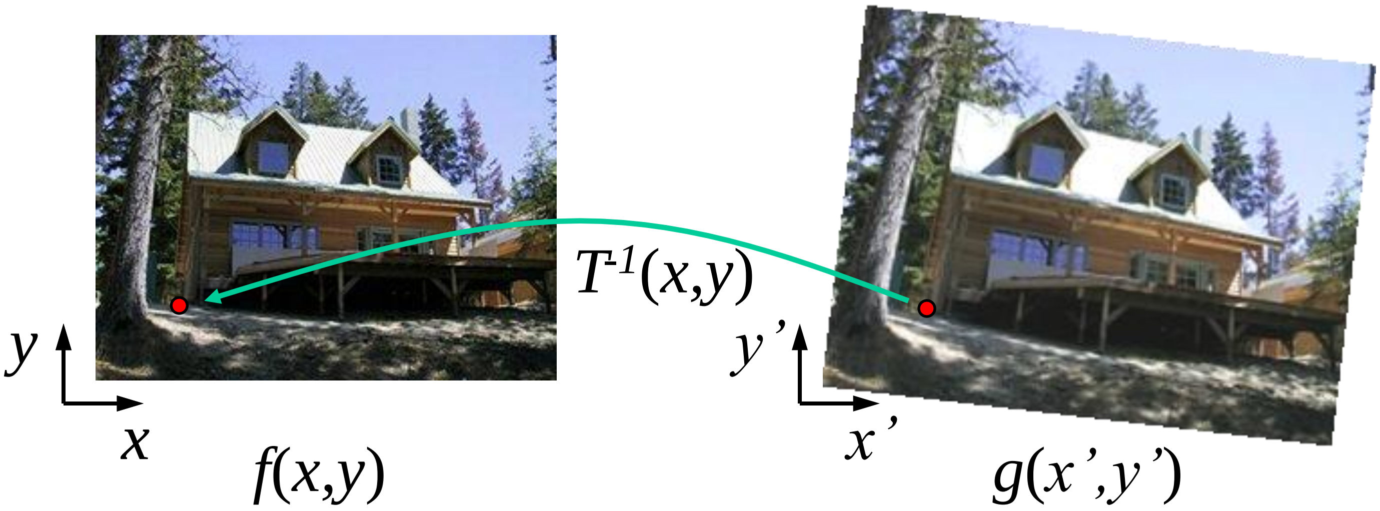
- Send each pixel *f*(*x,y*) to its corresponding location
- (*x',y'*) = *T*(*x,y*) in the second image

Q: what if pixel lands "between" two pixels?

A: distribute color among neighboring pixels (x',y')

– Known as "splatting"

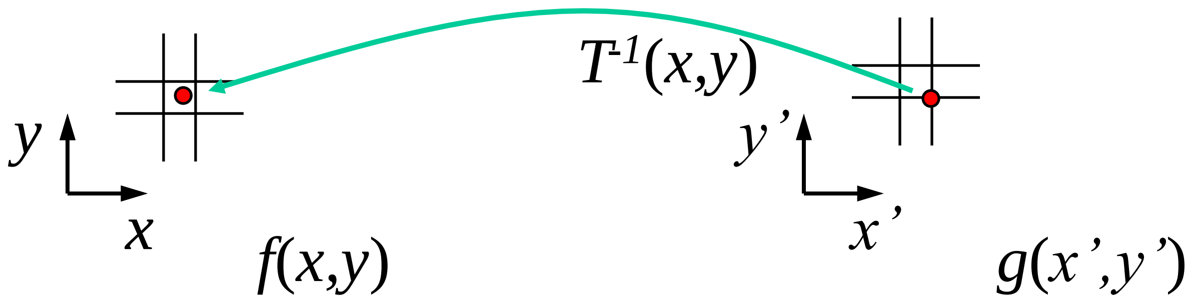
Inverse warping



- 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

– nearest neighbor, bilinear, Gaussian, bicubic