

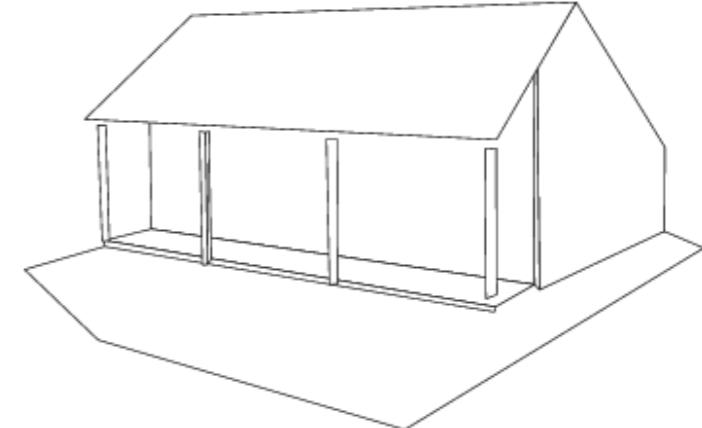
# Transformations and alignment



Slide credits: Deva Ramanan, Jitendra Malik

# Outline

- Transformations (2D/3D)
- Direct methods
- Lucas Kanade



Given a 2D image

Can we infer the 3D world that created this image?

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

**2D image points**      **Camera matrix M**      **3D points X**

Problem: both M and X are unknown!

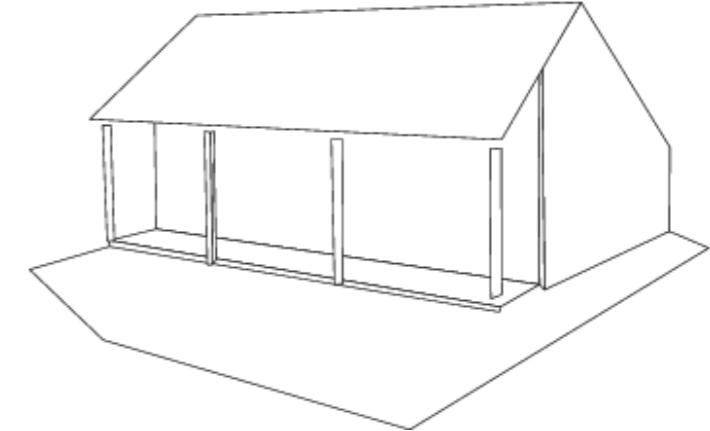
We can generate the same image with a modified camera  $M' = MQ$  and modified 3D geometry  $Q^{-1}X$  for any invertible matrix Q

$$M\mathbf{X} = MQQ^{-1}\mathbf{X}$$

**New Camera matrix**      **New 3D points**

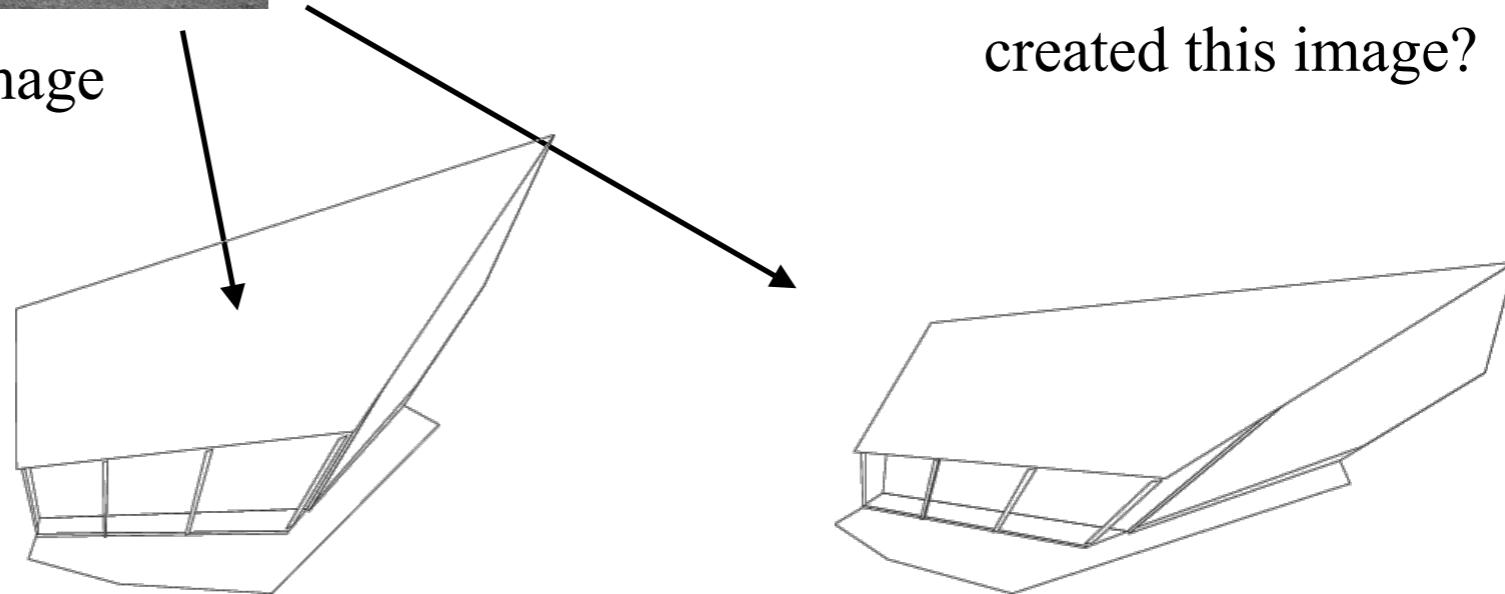


Given a 2D image



Can we infer the 3D world that created this image?

How do we know that this wasn't the 3D world that created this image?



$$M\mathbf{X} = M\mathbf{Q}\mathbf{Q}^{-1}\mathbf{X}$$

$\frac{\text{New}}{\text{New}}$   
**Camera matrix**    **3D points**

The different possible 3D reconstructions are all related by  $\mathbf{Q}^{-1}\mathbf{X}$  (a 3D homography!)

Thus, what do we know about the 3D world?

straight lines, intersection

Not: angles, lengths, parallel lines

# Family of 2D warps

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation	

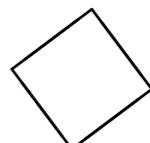
$$x' = x + t_x$$

$$y' = y + t_y$$

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

# Family of 2D warps

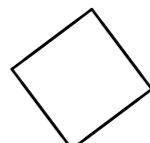
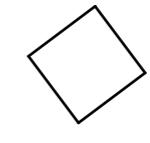
---

Transformation	Matrix	# DoF	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	

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

# Family of 2D warps

---

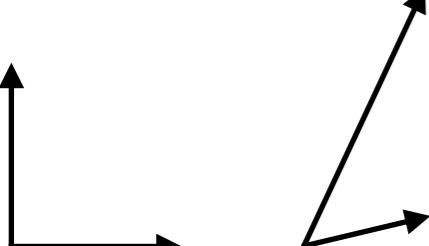
Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	4	angles	

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

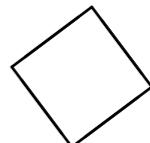
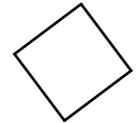
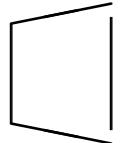
# Family of 2D warps

Transformation	Matrix	# DoF	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	

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


  
**change of basis**  
 (rotate, scale x and  
 y, rotate)      **2D**  
**translation**

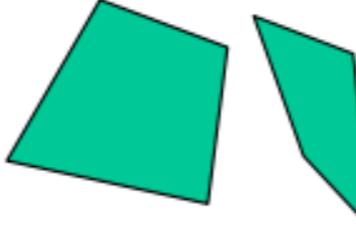
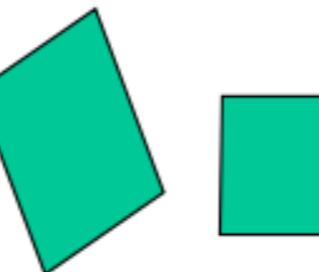
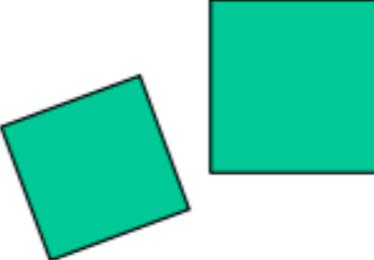
# Family of 2D warps

Transformation	Matrix	# DoF	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	

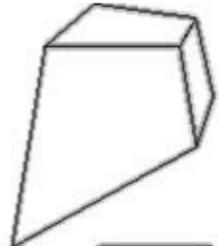
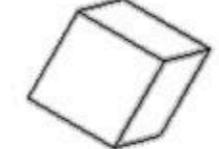
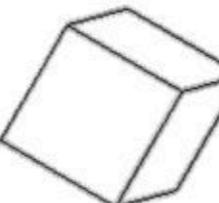
$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Relates the image projections of  
 (1) planar scene under any camera or (2) any scene under rotated cameras

# Hierarchy of 2D Transformations

		transformed squares	invariants
Projective 8dof	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$		Concurrency, collinearity, order of contact (intersection, tangency, inflection, etc.), cross ratio
Affine 6dof	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Parallelism, ratio of areas, ratio of lengths on parallel lines (e.g. midpoints), linear combinations of vectors (centroids), <b>The line at infinity <math>I_\infty</math></b>
Similarity 4dof	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Ratios of lengths, angles, The circular points I,J
Euclidean 3dof	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Absolute lengths, angles, areas

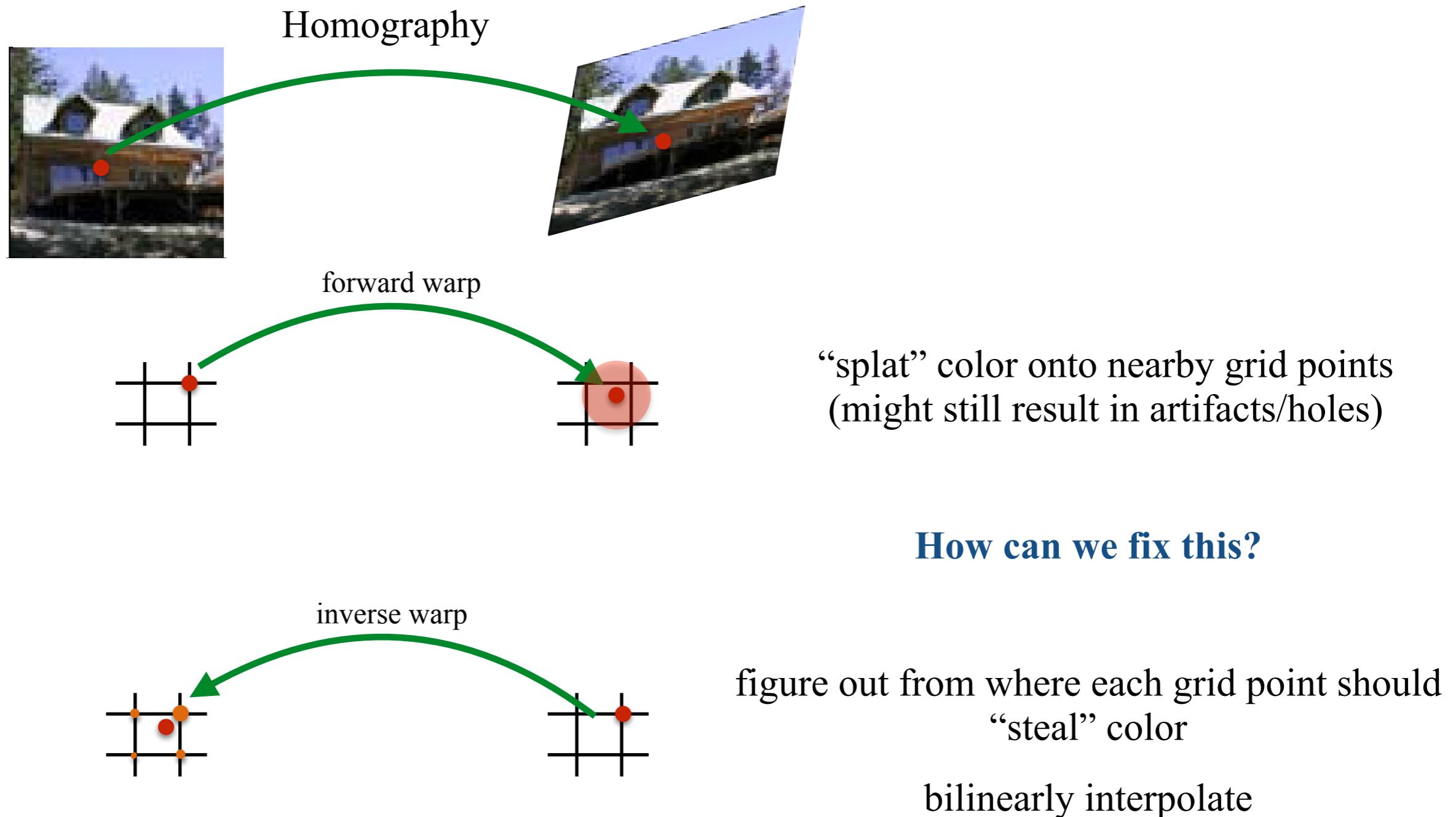
# Transformations in 3D

Projective 15dof	$\begin{bmatrix} A & t \\ v^T & v \end{bmatrix}$		Preserves intersection and tangency	
Affine 12dof	$\begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix}$		Preserves parallelism, volume ratios	
Similarity 7dof	$\begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix}$		Preserves angles, ratios of length	
“rigid body”	Euclidean 6dof	$\begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}$		Preserves angles, lengths

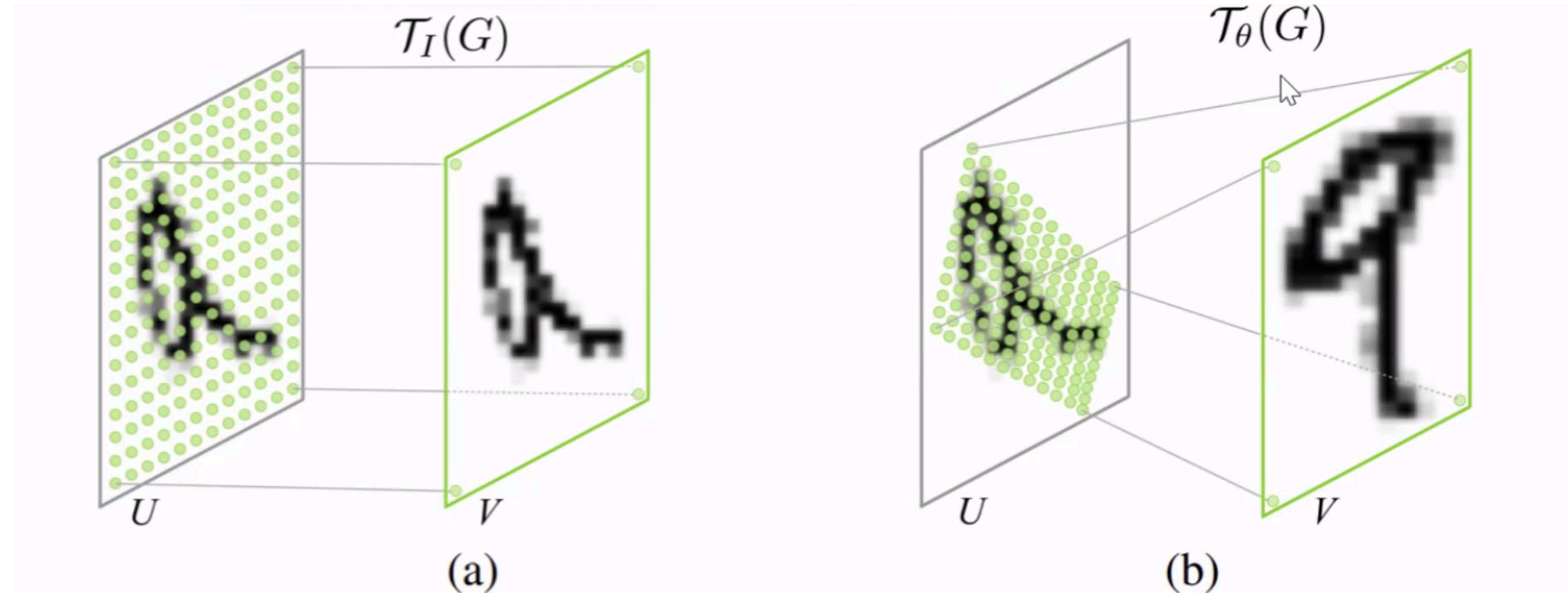
Make use of 3D homogenous coordinates (Normalize by last coordinate to recover 3D points)

$$\lambda \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

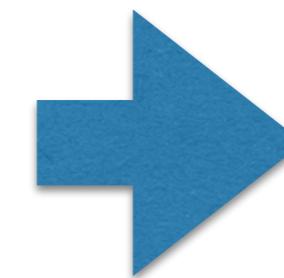
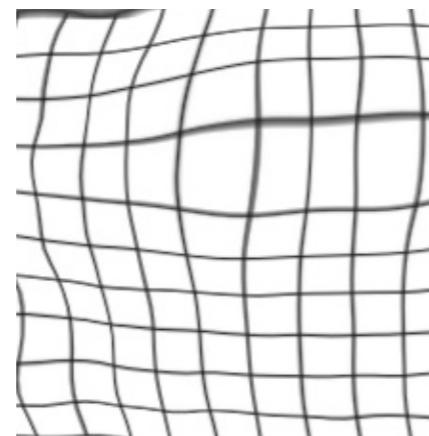
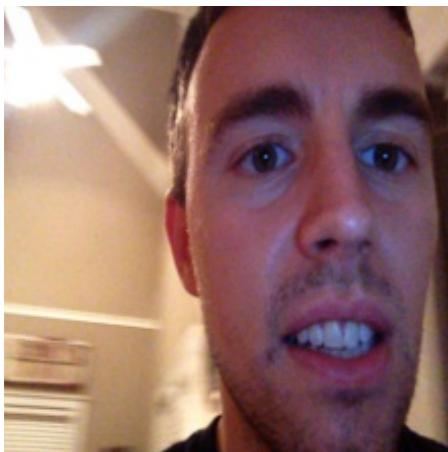
# How to warp an image



# Digression: spatial transformer networks



Neural networks that explicitly warp input images (or even feature maps)



classify eye\_gaze direction

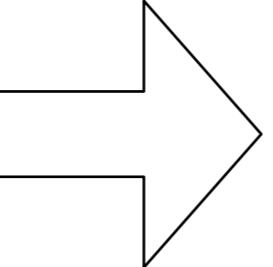
## Spatial Transformer Networks

Max Jaderberg   Karen Simonyan   Andrew Zisserman   Koray Kavukcuoglu  
Google DeepMind, London, UK  
[{jaderberg,simonyan,zisserman,korayk}@google.com](mailto:{jaderberg,simonyan,zisserman,korayk}@google.com)

## Learning to Zoom: a Saliency-Based Sampling Layer for Neural Networks

Adrià Recasens<sup>\*1</sup>, Petr Kellnhofer<sup>\*1</sup>, Simon Stent<sup>2</sup>, Wojciech Matusik<sup>1</sup>, and Antonio Torralba<sup>1</sup>

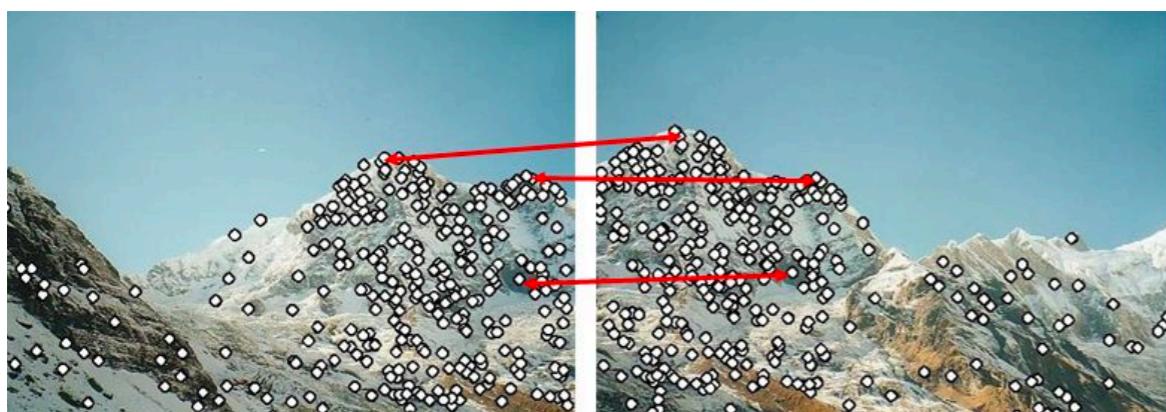
# Example application: data augmentation

3   
3 3 3 3 3  
3 3 3 3 3  
3 3 3 3 3  
3 3 3 3 3  
3 3 3 3 3

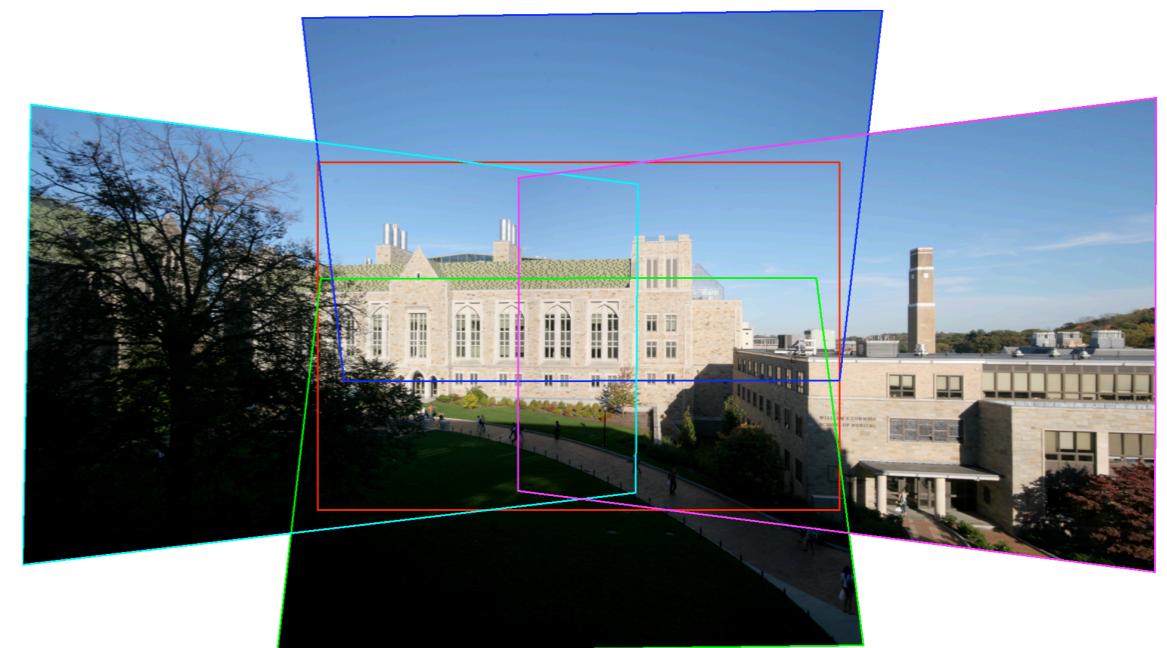
# Outline

- Transformations (2D/3D)
- Direct methods
- Lucas Kanade

# Approaches for image alignment



Alignment from sparse interest points



Dense, direct estimation of warp using **all** pixels

# HW2: Tracking

4 (10,445 milliseconds)



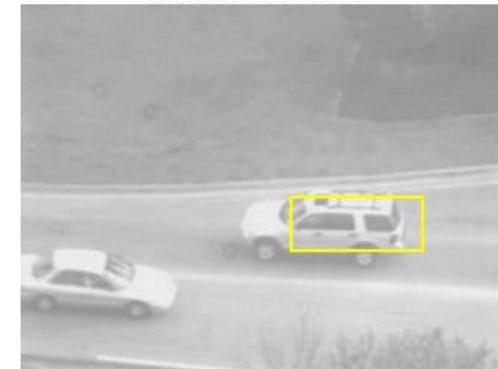
125 (12,648 milliseconds)



200 (7,493 milliseconds)



302 (11,301 milliseconds)



380 (5,634 milliseconds)



# Tracking



current frame



template  
(model)

$u, v$  = hypothesized location of template in current frame

We can track an object by matching a template to an image in every frame

# Tracking



current frame



template  
(model)

$u, v$  = hypothesized location of template in current frame

$$\min_{u,v} E(u, v), \quad \text{where} \quad E(u, v) = \sum_{x,y} [I(x + u, y + v) - T(x, y)]^2$$

**image patch at (u,v)      template**

**What is the transformation that is being searched over?      Translation**

**Think about: how would you modify this if you wanted to search over affine transformations**

# Tracking



current frame



template  
(model)

$u, v$  = hypothesized location of template in current frame

$$\min_{u,v} E(u, v), \quad \text{where} \quad E(u, v) = \sum_{x,y} [I(x + u, y + v) - T(x, y)]^2$$

**image patch at (u,v)**      **template**

This is Non-linear Least squares

Why is it non-linear?

The image function  $I(\dots)$  is nonlinear

# Feature detection: The Math

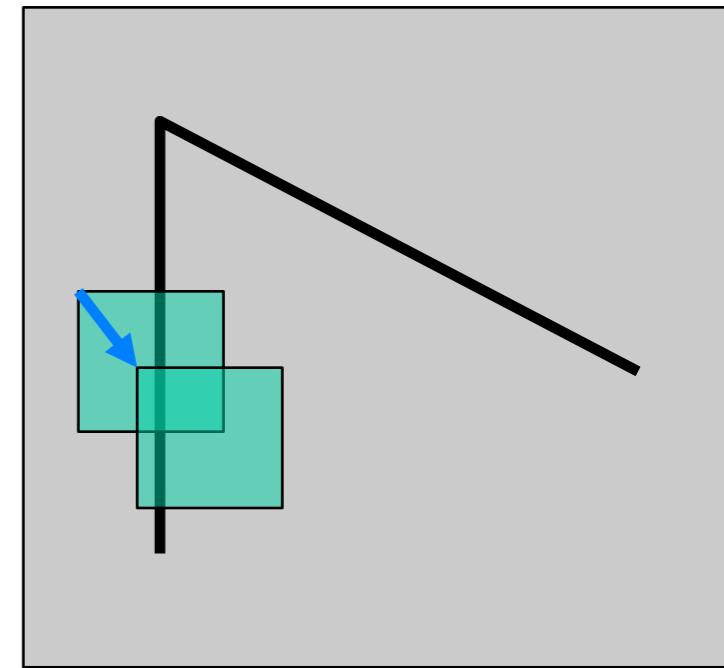
---

Consider shifting an image patch by  $(u, v)$

- How do the pixels in the patch change?
- Compare each pixel before and after by summing up the squared differences
- This defines an “error” of  $E(u, v)$ :

$$\text{Corner}(x_0, y_0) = \min_{u^2 + v^2 = 1} E(u, v)$$

**image patch**



$$\begin{aligned}
 E(u, v) &= \sum_{(x,y) \in W} [I(x + u, y + v) - I(x, y)]^2 \\
 &\quad \text{at } (u, v) \quad \text{original patch} \\
 &\approx \sum_{(x,y) \in W} [\mathbf{I} + \mathbf{I}_x u + \mathbf{I}_y v - \mathbf{I}]^2 \\
 &\quad \text{1st order approx} \\
 &= \sum_{(x,y) \in W} [\mathbf{I}_x^2 u^2 + \mathbf{I}_y^2 v^2 + 2\mathbf{I}_x \mathbf{I}_y uv] \\
 &= [u \quad v] A \begin{bmatrix} u \\ v \end{bmatrix}, \quad A = \sum_{(x,y) \in W} \begin{bmatrix} \mathbf{I}_x^2 & \mathbf{I}_x \mathbf{I}_y \\ \mathbf{I}_y \mathbf{I}_x & \mathbf{I}_y^2 \end{bmatrix}
 \end{aligned}$$

# Feature detection: The Math

Consider shifting an image patch by  $(u, v)$

- How do the pixels in the patch change?
- Compare each pixel before and after by summing up the squared differences
- This defines an “error” of  $E(u, v)$ :

$$\text{Corner}(x_0, y_0) = \min_{\substack{\text{(no constraint)} \\ \text{image patch}}} E(u, v)$$

**at  $(u, v)$**       **template**

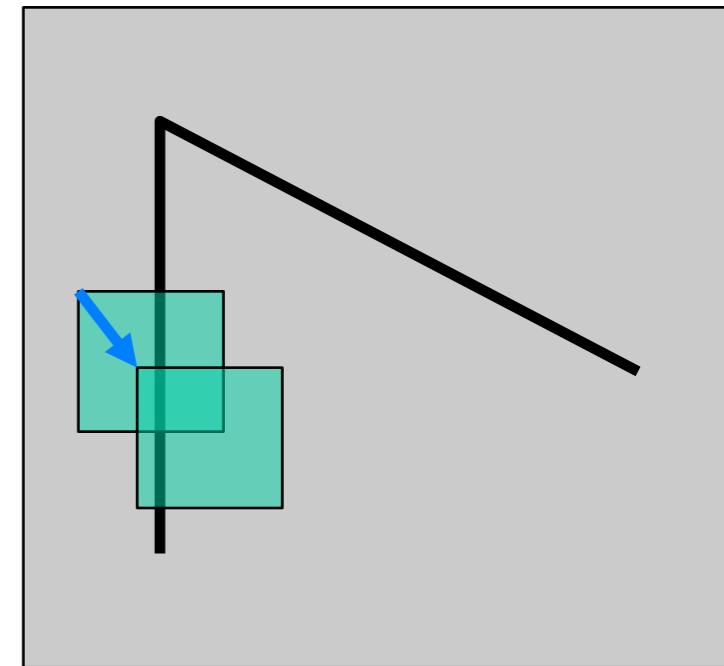
$$E(u, v) = \sum_{(x, y) \in W} [I(x + u, y + v) - T(x, y)]^2$$

**1st order approx**

$$\approx \sum_{(x, y) \in W} [\mathbf{I} + \mathbf{I}_x u + \mathbf{I}_y v - \mathbf{I}]^2$$

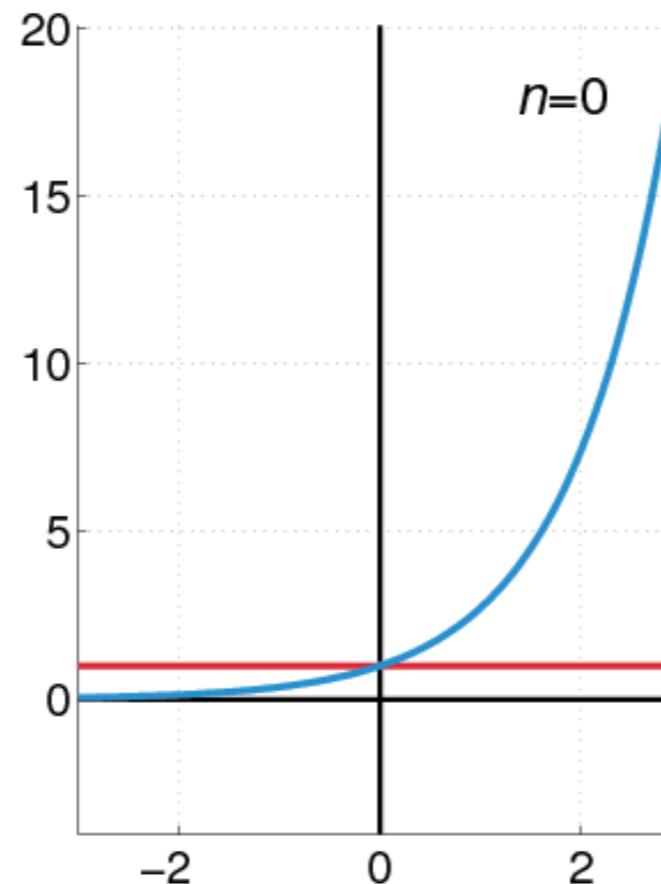
$$= \sum_{(x, y) \in W} [\mathbf{I}_x^2 u^2 + \mathbf{I}_y^2 v^2 + 2\mathbf{I}_x \mathbf{I}_y uv]$$

$$= \begin{bmatrix} u & v \end{bmatrix} A \begin{bmatrix} u \\ v \end{bmatrix}, \quad A = \sum_{(x, y) \in W} \begin{bmatrix} \mathbf{I}_x^2 & \mathbf{I}_x \mathbf{I}_y \\ \mathbf{I}_y \mathbf{I}_x & \mathbf{I}_y^2 \end{bmatrix}$$



# Recall: Taylor Series

$$f(x + u) = f(x) + \frac{\partial f(x)}{\partial x}u + \frac{1}{2}\frac{\partial^2 f(x)}{\partial x^2}u^2 + \text{Higher Order Terms}$$



For multivariate functions, we make use of first-order gradient vector and second-order Hessian matrix

# Nonlinear least squares

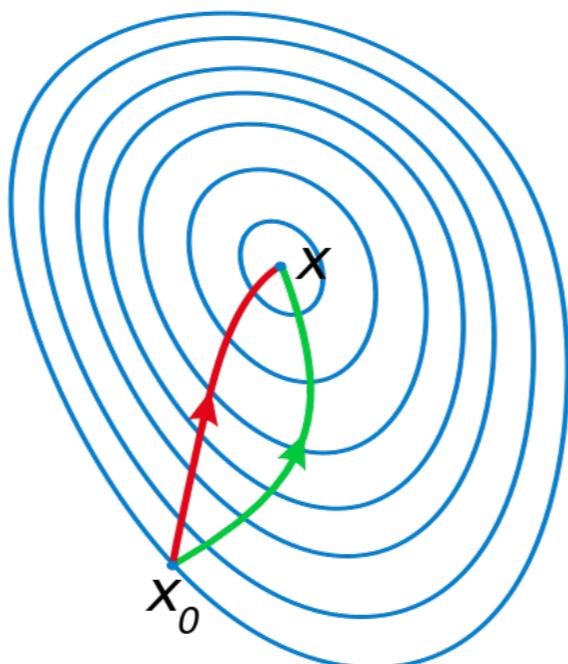
1. First order method (gradient descent)

$$u := u - \alpha g$$

2. Second order method (Newton's method)

[https://en.wikipedia.org/wiki/Newton%27s\\_method\\_in\\_optimization](https://en.wikipedia.org/wiki/Newton%27s_method_in_optimization)

$$u := u - H^{-1}g$$



We will do something a little different... (Gauss-Newton)

$$E(\mathbf{u}) = \frac{1}{2} \|\mathbf{f}(\mathbf{u})\|^2$$

**function I want to  
minimize: outputs a vector  
(e.g. R, G, B errors)**

$\mathbf{u}$  is a vector of my transformation parameters (e.g. translation, affine, etc)

How does the output of  $f$  change for small changes in  $u$ ?

**If  $f(u)$  outputs  
a single value**

$$\nabla_u f(u) = \begin{bmatrix} \frac{\partial f(u)}{\partial u_1} \\ \frac{\partial f(u)}{\partial u_2} \end{bmatrix}$$

**gradient vector**

**If  $f(u)$  outputs a vector:  
 $[f_1(u) \ f_2(u)]$**

$$J(u) = \begin{bmatrix} \frac{\partial f_1(u)}{\partial u_1} & \frac{\partial f_1(u)}{\partial u_2} \\ \frac{\partial f_2(u)}{\partial u_1} & \frac{\partial f_2(u)}{\partial u_2} \end{bmatrix}$$

**Jacobian matrix**

# Nonlinear least squares

$$E(\mathbf{u}) = \frac{1}{2} \|\mathbf{f}(\mathbf{u})\|^2$$

**function I want to minimize: outputs a vector (e.g. R, G, B errors)**

$u$  is a vector of my transformation parameters (e.g. translation, affine, etc)

If  $f(u)$  were a 1-D function:

$$E(u) = \frac{1}{2} f(u)^2 \quad \frac{dE(u)}{du} = f(u) \frac{df(u)}{du} \quad u \leftarrow u - \alpha \frac{dE(u)}{du}$$

If  $f(u)$  were a n-D function (but a 1D output):

$$E(u) = \frac{1}{2} f(u)^2 \quad \nabla_u E(u) = f(u) \nabla_u f(u)$$

$$u \leftarrow u - \alpha \nabla_u E(u)$$

$$\nabla_u f(u) = \begin{bmatrix} \frac{\partial f(u)}{\partial u_1} \\ \frac{\partial f(u)}{\partial u_2} \end{bmatrix}$$

**gradient vector**

If  $f(u)$  were a n-D function (with an m-dimensional output):

$$E(u) = \frac{1}{2} \|f(u)\|^2 \quad \nabla_u E(u) = J(u)^T f(u)$$

$$u \leftarrow u - \alpha \nabla_u E(u)$$

$$J(u) = \begin{bmatrix} \frac{\partial f_1(u)}{\partial u_1} & \frac{\partial f_1(u)}{\partial u_2} \\ \frac{\partial f_2(u)}{\partial u_1} & \frac{\partial f_2(u)}{\partial u_2} \end{bmatrix}$$

**Jacobian matrix**

# Nonlinear least squares

$$E(\mathbf{u}) = \frac{1}{2} \|\mathbf{f}(\mathbf{u})\|^2$$

**function I want to minimize:** outputs a vector (e.g. R, G, B errors)

$\mathbf{u}$  is a vector of my transformation parameters (e.g. translation, affine, etc)

Gauss-Newton method:

[https://en.wikipedia.org/wiki/Gauss–Newton\\_algorithm](https://en.wikipedia.org/wiki/Gauss–Newton_algorithm)

Instead of linearizing  $E(\mathbf{u})$  like on the previous slide,  
we will instead linearize  $\mathbf{f}(\mathbf{u})$ :

$$E(\mathbf{u} + \Delta\mathbf{u}) \approx \frac{1}{2} \|\mathbf{f}(\mathbf{u}) + \mathbf{J}(\mathbf{u})\Delta\mathbf{u}\|^2$$

**1st order Taylor series approximation of  $\mathbf{f}(\mathbf{u})$  around some initial guess  $\mathbf{u}$**

Known:  $f(u), J(u)$

Unknown:  $\Delta u$

This approximation is now quadratic in  $\Delta u$

We can now solve this for  $\Delta u$  with a simple least-squares!

Take gradient with respect to  $\Delta u$  and set equal to 0:

$$\mathbf{J}(\mathbf{u})^\top (\mathbf{f}(\mathbf{u}) + \mathbf{J}(\mathbf{u})\Delta\mathbf{u}) = 0$$

$$\Delta\mathbf{u} = - \left( \mathbf{J}(\mathbf{u})^\top \mathbf{J}(\mathbf{u}) \right)^{-1} \mathbf{J}(\mathbf{u})^\top \mathbf{f}(\mathbf{u})$$

# Feature detection: The Math

Consider shifting an image patch by  $(u, v)$

- How do the pixels in the patch change?
- Compare each pixel before and after by summing up the squared differences
- This defines an “error” of  $E(u, v)$ :

$$\text{Corner}(x_0, y_0) = \min_{u^2 + v^2 = 1} E(u, v)$$

**image patch**

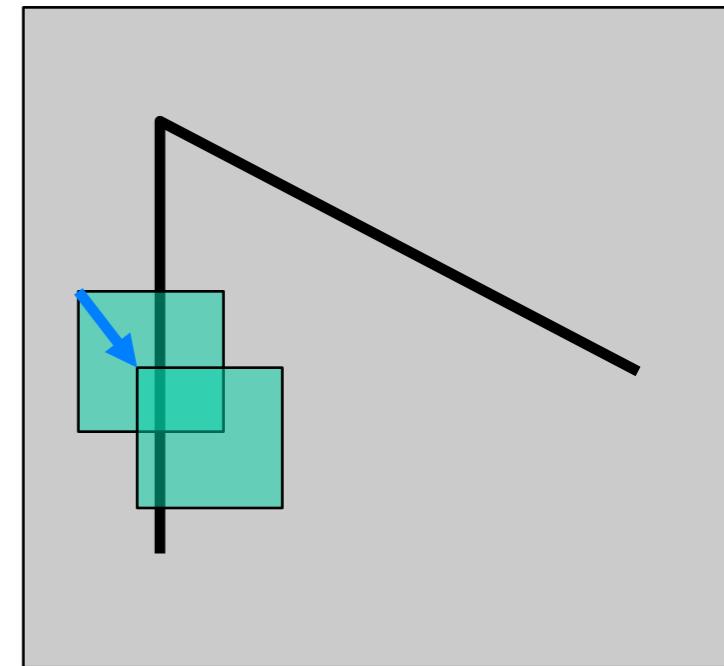
$$E(u, v) = \sum_{(x,y) \in W} [I(x + u, y + v) - I(x, y)]^2$$

**at  $(u, v)$**       **original patch**  
**1st order approx**

$$\approx \sum_{(x,y) \in W} [\mathbf{I} + \mathbf{I}_x u + \mathbf{I}_y v - \mathbf{I}]^2$$

$$= \sum_{(x,y) \in W} [\mathbf{I}_x^2 u^2 + \mathbf{I}_y^2 v^2 + 2\mathbf{I}_x \mathbf{I}_y uv]$$

$$= \begin{bmatrix} u & v \end{bmatrix} A \begin{bmatrix} u \\ v \end{bmatrix}, \quad A = \sum_{(x,y) \in W} \begin{bmatrix} \mathbf{I}_x^2 & \mathbf{I}_x \mathbf{I}_y \\ \mathbf{I}_y \mathbf{I}_x & \mathbf{I}_y^2 \end{bmatrix}$$



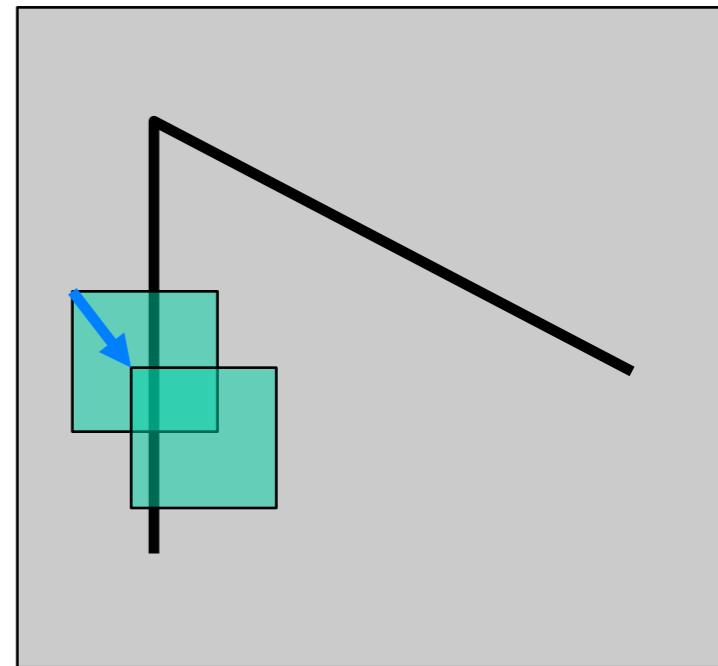
Solve with SVD

# Feature detection: The Math

Consider shifting an image patch by  $(u, v)$

- How do the pixels in the patch change?
- Compare each pixel before and after by summing up the squared differences
- This defines an “error” of  $E(u, v)$ :

$$\text{Corner}(x_0, y_0) = \min_{\frac{u^2 + v^2 = 1}{\text{image patch}}} E(u, v)$$



$$E(u, v) = \sum_{(x,y) \in W} [I(x + u, y + v) - \underset{\text{1st order approx}}{\text{T}(x,y)}]^2$$

$$\approx \sum_{(x,y) \in W} [\mathbf{I} + \mathbf{I}_x u + \mathbf{I}_y v - \mathbf{I}]^2$$

$$= \sum_{(x,y) \in W} [\mathbf{I}_x^2 u^2 + \mathbf{I}_y^2 v^2 + 2\mathbf{I}_x \mathbf{I}_y uv]$$

$$= \begin{bmatrix} u & v \end{bmatrix} A \begin{bmatrix} u \\ v \end{bmatrix}, \quad A = \sum_{(x,y) \in W} \begin{bmatrix} \mathbf{I}_x^2 & \mathbf{I}_x \mathbf{I}_y \\ \mathbf{I}_y \mathbf{I}_x & \mathbf{I}_y^2 \end{bmatrix}$$

~~Solve with SVD~~

# Matching a template to an image



current frame



template  
(model)

$u, v$  = hypothesized location of  
template in current frame

$$\min_{u,v} E(u, v), \quad \text{where} \quad E(u, v) = \sum_{x,y} [I(x + u, y + v) - T(x, y)]^2$$

image patch at (u,v)      template

Sum over pixels  
in template

# Lucas Kanade alignment

**image patch  
at  $(u, v)$**       **template**

$$\begin{aligned}
 E(u, v) &= \sum_{x, y} [I(x + u, y + v) - T(x, y)]^2 \\
 &\approx \sum_{x, y} [I(x, y) + uI_x(x, y) + vI_y(x, y) - T(x, y)]^2 && \text{1st order approximation} \\
 &= \sum_{x, y} [uI_x(x, y) + vI_y(x, y) + D(x, y)]^2 && \text{of } I(x+u, y+v) \\
 &&& = I(x, y) - T(x, y)
 \end{aligned}$$

Now Take partial derivs and set to zero (of the linearized energy function)

$$\frac{\delta E}{du} = \sum_{x, y} [uI_x(x, y) + vI_y(x, y) + D(x, y)] I_x(x, y) = 0$$

$$\frac{\delta E}{dv} = \sum_{x, y} [uI_x(x, y) + vI_y(x, y) + D(x, y)] I_y(x, y) = 0$$

Form matrix equation

$$\sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \sum \begin{bmatrix} I_x D \\ I_y D \end{bmatrix} \quad \rightarrow \quad \text{Solve with: } x = A^{-1}b$$

“Ax=b”

# Lucas Kanade alignment

**image patch**  
**at (u,v)**      **template**

$$E(u, v) = \sum [I(x+u, y+v) - T(x, y)]^2$$

$$\sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \sum \begin{bmatrix} I_x D \\ I_y D \end{bmatrix}$$

“Ax=b”      Solve with:  $x = A^{-1}b$

**But is A always invertible?**

**No, it is not invertible if one of the eigenvalues is 0**

**Thus, a template is “easy to track” if both eigenvalues are large**

**Does this sound familiar?**

When we talked about corniness, we were looking for patches that were “easy to match”

We found a “good” solution (i.e. a corner) when both eigenvalues were large!

# Lucas Kanade Alignment

Initialize the template to be aligned to some location in the image

Example: we can initialize based on the estimated position of the template in the previous frame



# Lucas Kanade Alignment

Gauss-Newton step:



template



Solve for parameters to shift the template from the initialization  $[u, v]$ :

$$\sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \sum \begin{bmatrix} I_x D \\ I_y D \end{bmatrix}$$

# Lucas Kanade Alignment

This will result in a new guess as to the best location to align the template with the image

We can then take another Gauss-Newton step (same procedure as the previous slide)

We are now linearizing the image about a different point so we will get a different solution!

We can keep iterating



**Is this guaranteed to converge to the right answer?**

No - this is a non-linear optimization and we are just taking small update steps around the initialization

**Is this guaranteed to converge at all?**

Actually no - we are not performing gradient updates:  $u \leftarrow u + \alpha \nabla_u f(u)$

The Gauss-Newton method is not guaranteed to converge

If we are in the *basin of attraction* (e.g. near the optimum), convergence tends to be fast (few iterations)

# HW2: Tracking by iterative template alignment



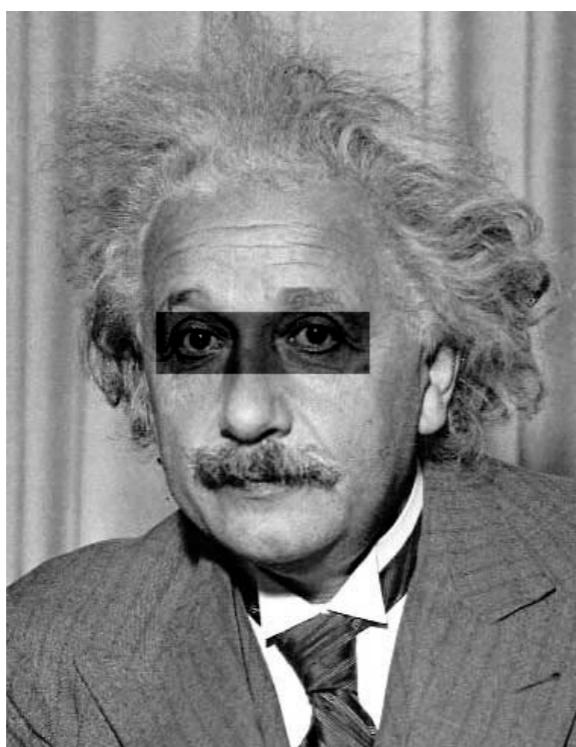
Start with template from first frame and repeat:

1. Align template to new frame with Lucas Kanade
2. (Optional) update template with new frame

**What are some pros and cons of updating the template with the new frame?**

# Recall: Normalized cross correlation (NCC)

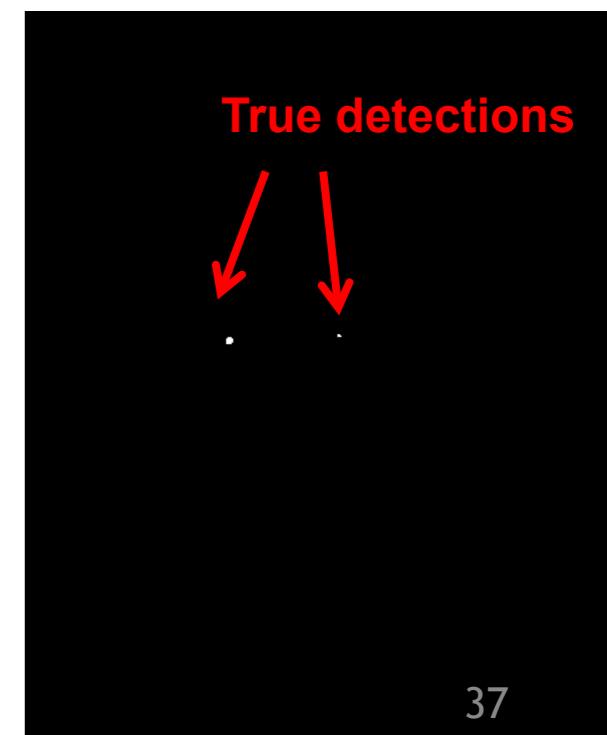
$$NCC[i, j] = \frac{H^T F_{ij}}{\|H\| \|F_{ij}\|} = \cos \theta$$



NCC



NCC thresholded



# Why do we need Gauss-Newton?

Why not just run normalized cross-correlation?



current frame



template  
(model)

$u, v$  = hypothesized location of template in current frame

$$\min_{u,v} E(u, v), \quad \text{where} \quad E(u, v) = \sum_{x,y} [I(x + u, y + v) - T(x, y)]^2$$

image patch at (u,v)      template

Sum over pixels  
in template

What is the transformation that is being searched over?

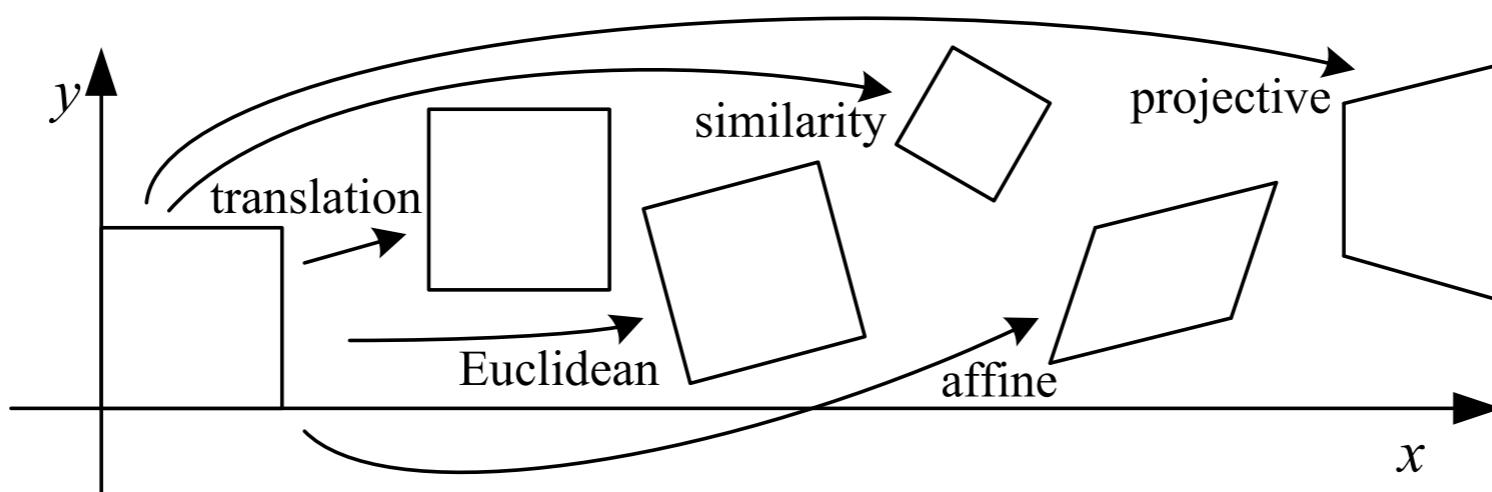
Translation

For translation, NCC is sufficient

We need Gauss-Newton in case we need to search over other types of transformations (e.g. affine) - warping the template to better match the image

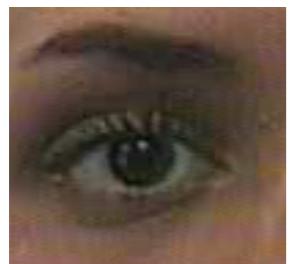
# Family of 2D warps

Transformation	Matrix	# DoF	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	

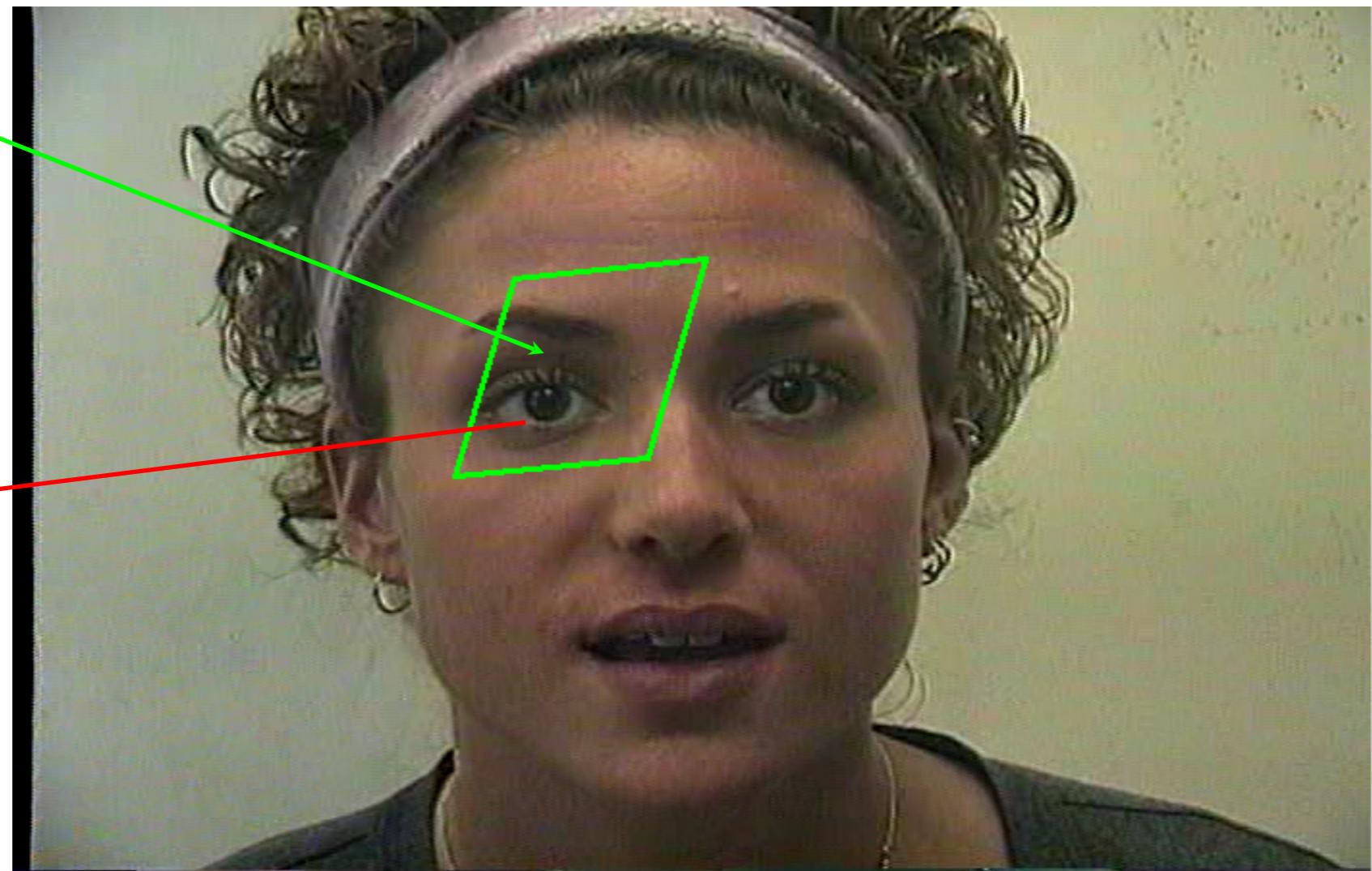
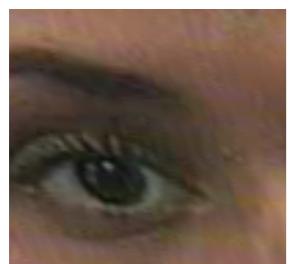


# Example

Image patch

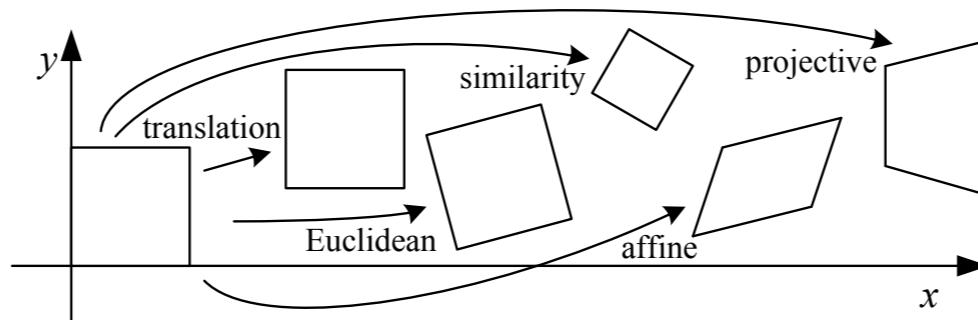


Warped patch



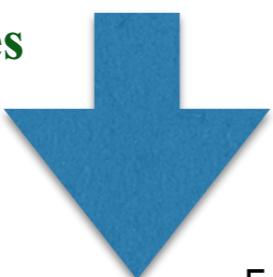
# Replace [u,v] translation with a general warping function $W(x,y; p)$

image warping  
coordinates parameters



$$E(\mathbf{p}) = \sum_{\mathbf{x}} [I(W(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2$$

error function  
(how good of a  
match is this)  
image  
coordinates



e.g., for translation warps:  $W(\mathbf{x}; \mathbf{p}) = \begin{bmatrix} 1 & 0 & p_1 \\ 0 & 1 & p_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

e.g., for affine warps:  $W(\mathbf{x}; \mathbf{p}) = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

# Recall: Translation Warps

$$\begin{aligned}
 E(u, v) &= \sum_{x, y} [I(x+u, y+v) - T(x, y)]^2 \\
 &\approx \sum_{x, y} [I(x, y) + uI_x(x, y) + vI_y(x, y) - T(x, y)]^2 \\
 &= \sum_{x, y} [uI_x(x, y) + vI_y(x, y) + D(x, y)]^2 \\
 &\quad = I(x, y) - T(x, y)
 \end{aligned}$$

image patch  
at (u,v)      template

1st order approximation  
of the translated  
image  $I(x+u, y+v)$

Now Take partial derivs and set to zero (of the approximate energy function)

$$\frac{\delta E}{du} = \sum_{x, y} [uI_x(x, y) + vI_y(x, y) + D(x, y)] I_x(x, y) = 0$$

$$\frac{\delta E}{dv} = \sum_{x, y} [uI_x(x, y) + vI_y(x, y) + D(x, y)] I_y(x, y) = 0$$

Form matrix equation

$$\sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \sum \begin{bmatrix} I_x D \\ I_y D \end{bmatrix} \quad \rightarrow \quad \text{Solve with: } x = A^{-1}b$$

“Ax=b”

# Back to the big-picture

$$E(\mathbf{p}) = \sum_{\mathbf{x}} [I(W(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2$$

error function  
(how good of a  
match is this)

warped image  
image  
coordinates

template

1st order approximation  
of the warped image  
(we will explain this step in  
more detail on the next slide)

$$\approx \sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta \mathbf{p} - T(\mathbf{x}) \right]^2$$

current warped image

current warped image gradient

Jacobian of image coordinates w.r.t. warp parameters  $\mathbf{p}$

template

small parameter updates

Then we compute the gradient of this approximate energy function and set = 0:

$$\nabla_{\Delta \mathbf{p}} E(\Delta \mathbf{p}) = 0$$

Solve for  $\Delta \mathbf{p}$

# 1st order approximation of the warped image

## 1-parameter warp

$$I(W(\mathbf{x}; p + \Delta p)) \approx I(W(\mathbf{x}; p)) + \begin{bmatrix} \frac{\partial I(W(\mathbf{x}; p))}{\partial x} & \frac{\partial I(W(\mathbf{x}; p))}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial W_x(\mathbf{x}; p)}{\partial p} \\ \frac{\partial W_y(\mathbf{x}; p)}{\partial p} \end{bmatrix} \Delta p$$

shifted parameters      current warped image      current warped image gradient      Jacobian of image coordinates w.r.t. warp parameter p      small parameter update

## multi-parameter warp ( $p_1, p_2$ )

$$I(W(\mathbf{x}; \mathbf{p} + \Delta \mathbf{p})) \approx I(W(\mathbf{x}; \mathbf{p})) + \begin{bmatrix} \frac{\partial I(W(\mathbf{x}; \mathbf{p}))}{\partial x} & \frac{\partial I(W(\mathbf{x}; \mathbf{p}))}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial W_x(\mathbf{x}; \mathbf{p})}{\partial p_1} & \frac{\partial W_x(\mathbf{x}; \mathbf{p})}{\partial p_2} \\ \frac{\partial W_y(\mathbf{x}; \mathbf{p})}{\partial p_1} & \frac{\partial W_y(\mathbf{x}; \mathbf{p})}{\partial p_2} \end{bmatrix} \begin{bmatrix} \Delta p_1 \\ \Delta p_2 \end{bmatrix}$$

current warped image       $\nabla I(W(\mathbf{x}; \mathbf{p}))$        $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$       small parameter updates  
current warped image gradient      Jacobian of image coordinates w.r.t. warp parameters p

# 1st order approximation of the warped image

## 1-parameter warp

$$I(W(\mathbf{x}; p + \Delta p)) \approx I(W(\mathbf{x}; p)) + \begin{bmatrix} \frac{\partial I(W(\mathbf{x}; p))}{\partial x} & \frac{\partial I(W(\mathbf{x}; p))}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial W_x(\mathbf{x}; p)}{\partial p} \\ \frac{\partial W_y(\mathbf{x}; p)}{\partial p} \end{bmatrix} \Delta p$$

shifted parameters      current warped image      current warped image gradient      Jacobian of image coordinates w.r.t. warp parameter p      small parameter update

## multi-parameter warp ( $p_1, p_2$ )

$$I(W(\mathbf{x}; \mathbf{p} + \Delta \mathbf{p})) \approx I(W(\mathbf{x}; \mathbf{p})) + \begin{bmatrix} \frac{\partial I(W(\mathbf{x}; \mathbf{p}))}{\partial x} & \frac{\partial I(W(\mathbf{x}; \mathbf{p}))}{\partial y} \\ \nabla I(W(\mathbf{x}; \mathbf{p})) \end{bmatrix} \begin{bmatrix} \frac{\partial W_x(\mathbf{x}; \mathbf{p})}{\partial p_1} & \frac{\partial W_x(\mathbf{x}; \mathbf{p})}{\partial p_2} \\ \frac{\partial W_y(\mathbf{x}; \mathbf{p})}{\partial p_1} & \frac{\partial W_y(\mathbf{x}; \mathbf{p})}{\partial p_2} \end{bmatrix} \begin{bmatrix} \Delta p_1 \\ \Delta p_2 \end{bmatrix}$$

current warped image      current warped image gradient      Jacobian of image coordinates w.r.t. warp parameters p      small parameter updates

Let's look into how to compute this term

Example: Assume just a rotation

$$E(\mathbf{p}) = \sum_{\mathbf{x}} [I(W(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2$$

**error function**       **$\mathbf{x}$**       **warped image**      **template**  
**(how good of a**      **image**  
**match is this)**      **coordinates**

$$\mathbf{W}(\mathbf{x}; p) = \begin{bmatrix} \cos(p) & -\sin(p) \\ \sin(p) & \cos(p) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$W_x = \cos(p)x - \sin(p)y$$

$$W_y = \sin(p)x + \cos(p)y$$

Now we need to re-derive the Gauss-Newton update!

Apply Taylor-series expansion to warped image  $I(W(x; p))$

Make use of chain rule to compute derivate of image wrt  $p=\theta$ :

$$\frac{\partial I(W(\mathbf{x}; p))}{\partial p} = \frac{\partial I(W(\mathbf{x}; p))}{\partial x} \frac{\partial W_x(\mathbf{x}; p)}{\partial p} + \frac{\partial I(W(\mathbf{x}; p))}{\partial y} \frac{\partial W_y(\mathbf{x}; p)}{\partial p}$$

# Change of warped image w.r.t. p

# Change of image w.r.t. x (x-derivative)

**derivative  
of coordinates  
w.r.t. warp  
parameters**

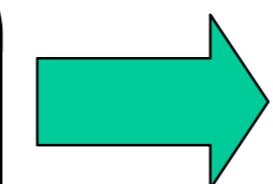
## Change of image w.r.t. y (y-derivative)

## derivative of coordinates w.r.t. warp parameters

# Example: Affine warp

affine warp function (6 parameters)

Jacobian  
of warped image  
w.r.t. warp  
parameters p

$$W([x, y]; P) = \begin{pmatrix} 1 + p_1 & p_3 & p_5 \\ p_2 & 1 + p_4 & p_6 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$\frac{\partial W}{\partial P} = \frac{\partial \begin{bmatrix} x + xP_1 + yP_3 + P_5 \\ xP_2 + y + yP_4 + P_6 \end{bmatrix}}{\partial P}$$

If I set all parameters to 0  
then I get the identity transformation

(this is better for optimization - I can optimize about 0  
if I think that the identity is a good initial guess)

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

# Back to the big-picture

$$E(\mathbf{p}) = \sum_{\mathbf{x}} [I(W(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2$$

error function  
(how good of a  
match is this)

warped image

image coordinates

template

1st order approximation  
of the warped image

$$\approx \sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta \mathbf{p} - T(\mathbf{x}) \right]^2$$

current warped image

current warped image gradient

Jacobian of image coordinates w.r.t. warp parameters  $\mathbf{p}$

template

small parameter updates

Then we compute the gradient of this approximate energy function and set = 0:

$$\nabla_{\Delta \mathbf{p}} E(\Delta \mathbf{p}) = 0$$

Solve for  $\Delta \mathbf{p}$

1st order approximation  
of the warped image

$$\approx \sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta \mathbf{p} - T(\mathbf{x}) \right]^2$$

Then we compute the gradient of this approximate energy function and set = 0:

$$\nabla_{\Delta \mathbf{p}} E(\Delta \mathbf{p}) = 0$$

Image Gradient

$$\Delta \mathbf{p} = \sum_{\mathbf{x}} H^{-1} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \frac{[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]}{\text{template currently warped image}}$$

Error Image

Approx hessian

$$H = \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

What happens when the error image is 0?

No update!

# Lucas-Kanade Algorithm

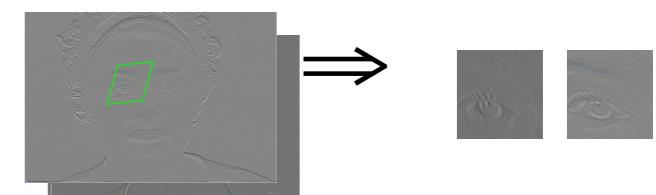
1. Warp  $I$  with  $\mathbf{W}(\mathbf{x}; \mathbf{p}) \Rightarrow I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$   
**current warp**      **warped image**



2. Compute error image  $T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$   
**template**      **warped image**



3. Warp gradient of  $I$  to compute  $\nabla I$



4. Evaluate Jacobian  $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$



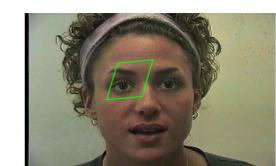
5. Compute approximate Hessian

$$H = \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

6. Compute  $\Delta \mathbf{p}$

$$\Delta \mathbf{p} = \sum_{\mathbf{x}} H^{-1} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$$

7. Update warp parameters  $\mathbf{p} \leftarrow \mathbf{p} + \Delta \mathbf{p}$



# Extra Tricks to Speed up

Additive warp:  $\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x})]^2$   $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$ .

Compositional warps:  $\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}); \mathbf{p})) - T(\mathbf{x})]^2$   $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ ,

Jacobian is always evaluated at  $\mathbf{p} = 0$ , which means it can be precomputed

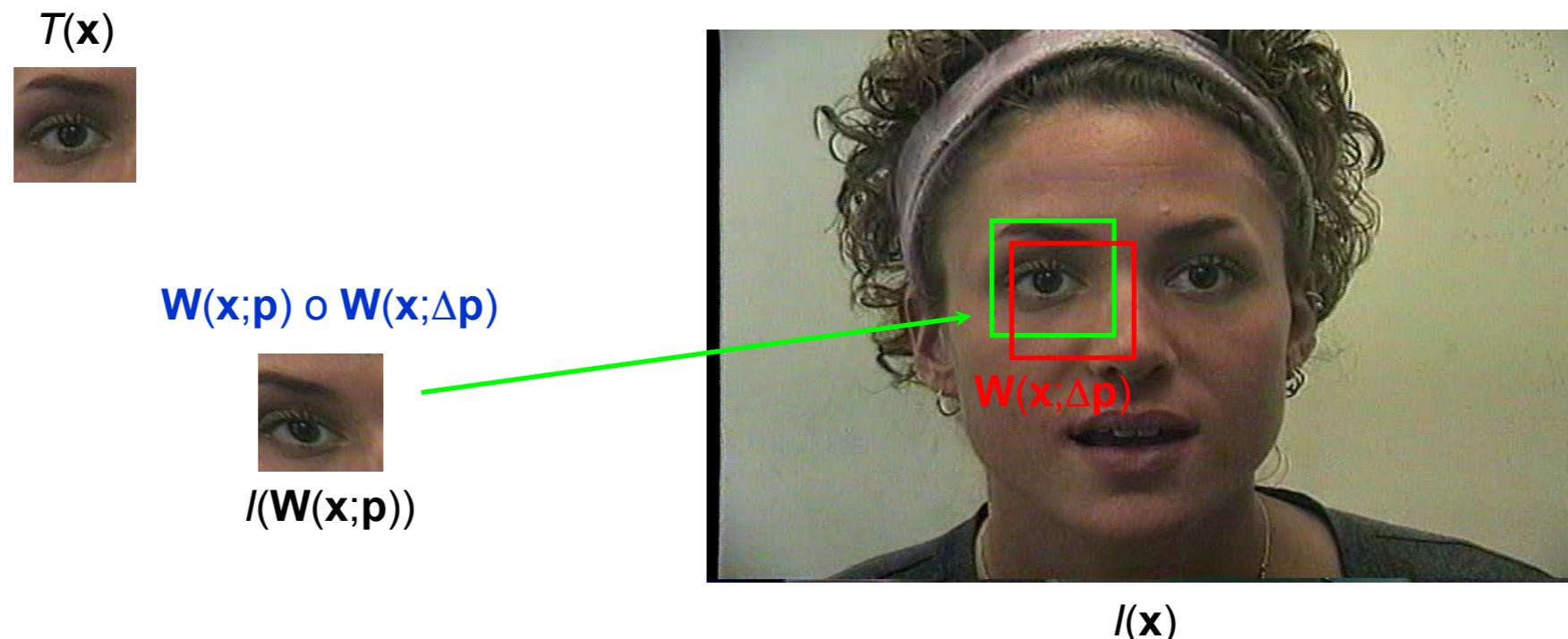
Inverse compositional warp:  $\sum_{\mathbf{x}} [T(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]^2$   $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})^{-1}$ .

Solve for  $\Delta p$  to warp the template, but then instead warp the image with the inverse

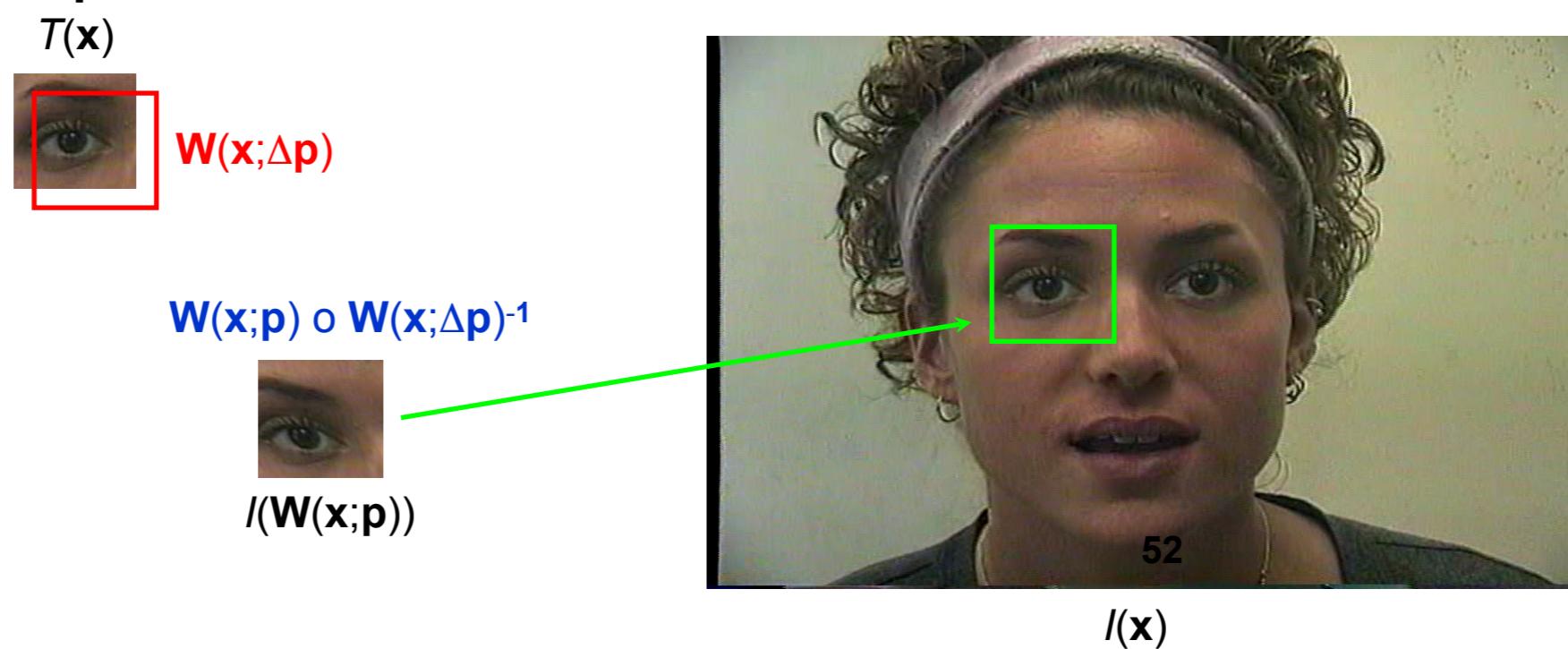
Both Jacobian and Hessian are not a function of current  $\mathbf{p}$  (since the warp changes the image, not the template) and so they can be precomputed

# Forward and Inverse Compositional

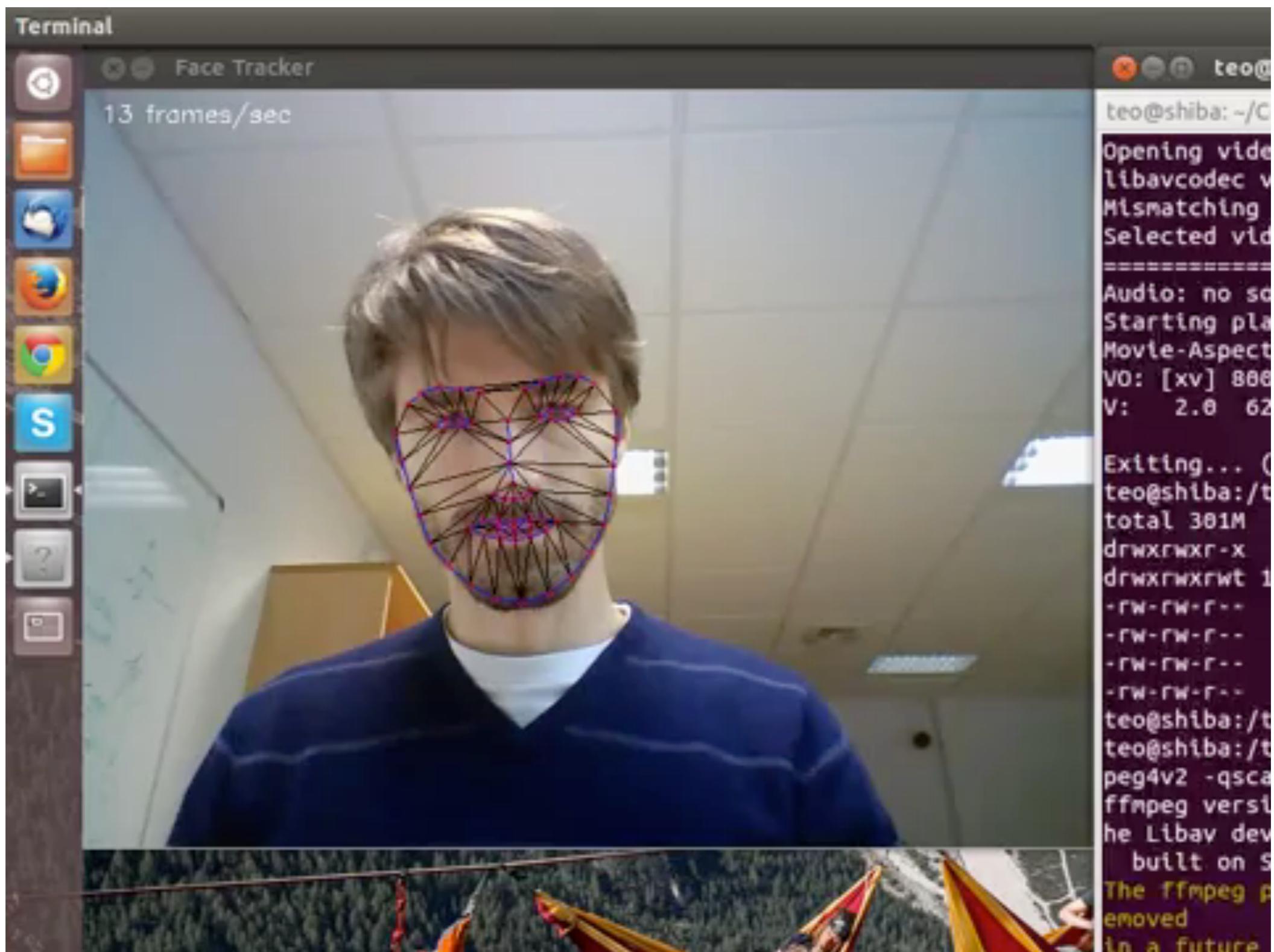
- Forwards compositional



- Inverse compositional



# Real-time Piecewise affine-tracking



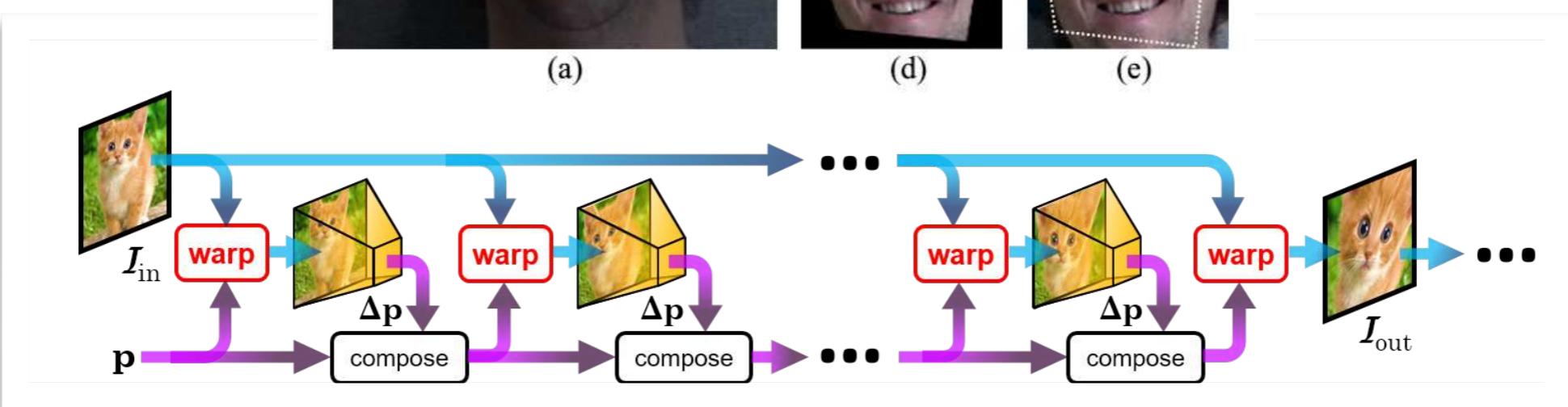
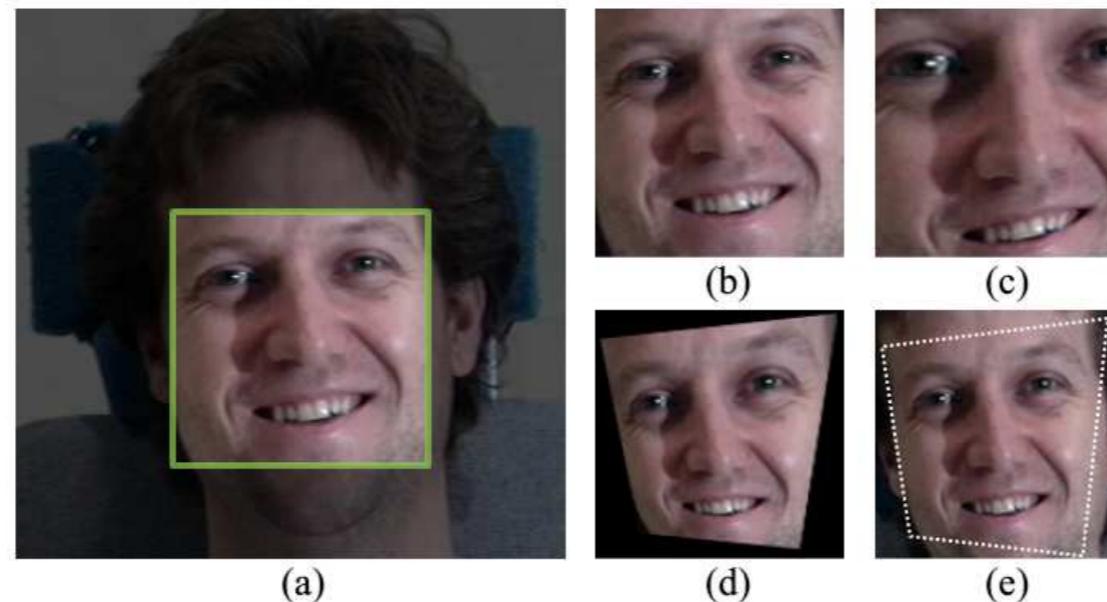
# Contemporary extensions

## Inverse Compositional Spatial Transformer Networks

Chen-Hsuan Lin      Simon Lucey

The Robotics Institute  
Carnegie Mellon University

chenhsul@andrew.cmu.edu      slucey@cs.cmu.edu



# Summary

- Gauss Newton optimization of nonlinear least squares
- Lucas Kanade for tracking a template