Image Warping

Logistics

- We will have lecture next Tuesday (changed to later flight)

Date	Topic	References	Misc
1/16	Introduction		[First class]
1/18	Filters		Special Recitation on HW0; NSH 3305 at 5pm
1/23	Edges		HW0 - Python Intro (not graded)
1/25	Texture		
1/30	Linear algebra review		
2/1	Warping		HW1 - HOG Image Classification
2/6	Alignment	Deva in town!	
2/8	Frequency	Guest lecture	
2/13	Correspondence		
2/15	Descriptors		HW2 - LK Tracking
2/20	Image Formation		
2/22	Cameras		
2/27	Motion		
2/29	Flow		HW3 - Homographies
3/5	NO CLASS [Spring Break]		
3/7	NO CLASS [Spring Break]		
3/12	Two-view		
3/14	Stereo		
3/19	SFM		
3/21	Recognition		HW4 - Reconstruction
3/26	Classifiers	Guest Lecture	
3/28	MLPs	Guest Lecture	
4/2	Training Recipes		
4/4	CNNs		
4/9	Interpreting Networks		HW5 - Neural Networks
4/11	NO CLASS [Spring Carnival]		
4/16	Radiometry		
4/18	Color		
4/23	Recent Trends		
4/25	Catchup		HW6 - Photometric Stereo

Printing PDFs; nice YouTube posted to Piazza on multiple pathways to generate PDF
Suggestion; give yourself at least 30 min to iron this out for tonight's deadline

Linear algebra digression

https://www.cs.cmu.edu/~zkolter/course/linalg/linalg_notes.pdf

My own writeup (in lec gdrive folder): updated for today

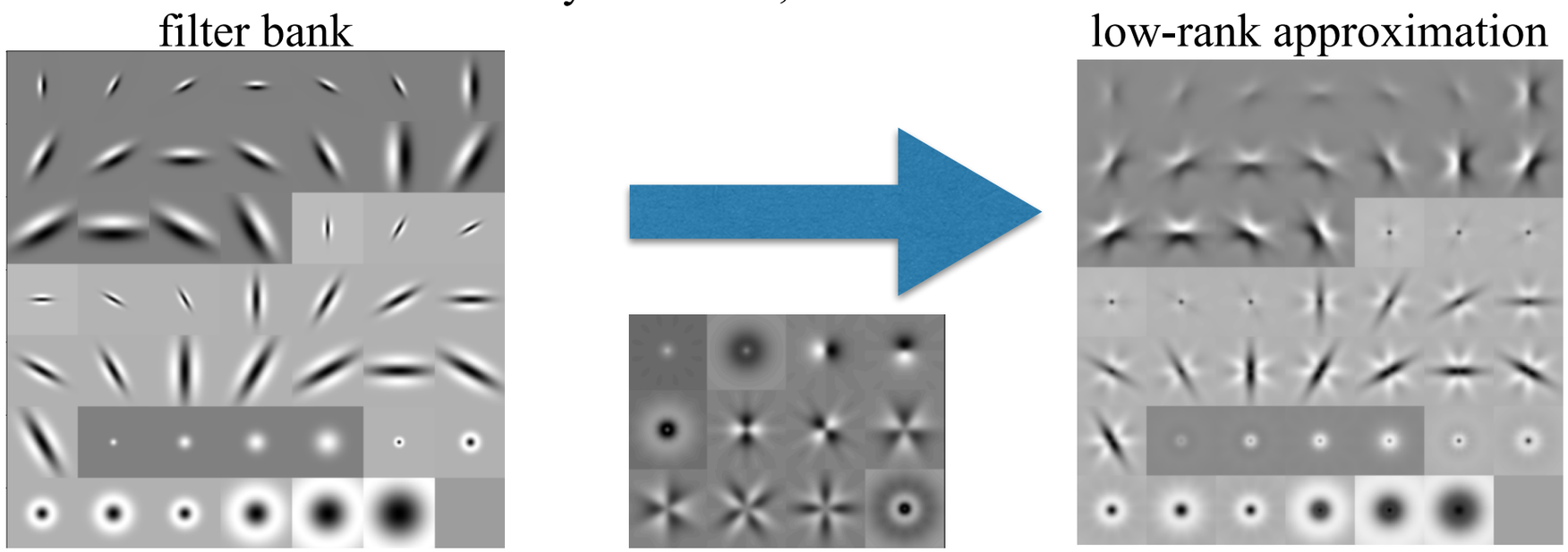
https://drive.google.com/file/d/1c-kjMqIQEKJCHyfMgLzdxRJNReOu6ssk/view?usp=drive_link

Least-squares method of steerability

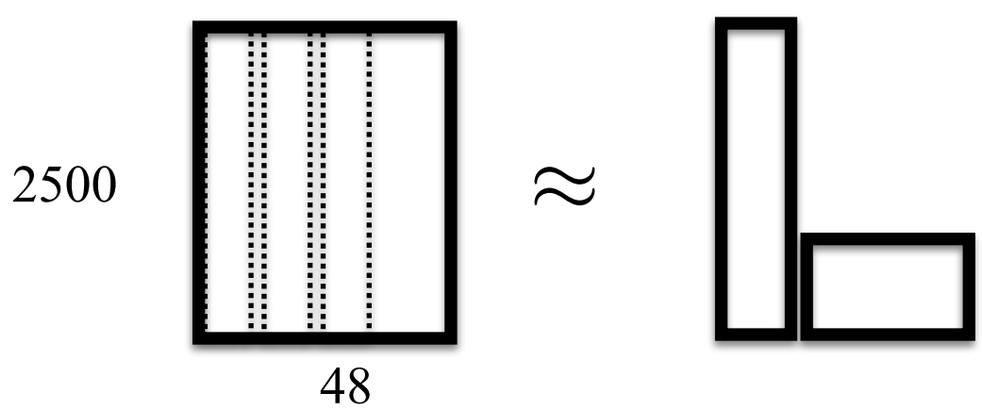
demo_faces.ipynb

demo_filter_bank.ipynb

Shy & Perona, CVPR94



- Column-vectorize each of 48 50x50 filters and stack'em into a 2500x48 matrix
- Apply SVD to generate low-rank approximation of 48 filters as linear combination of 12 *basis* filters

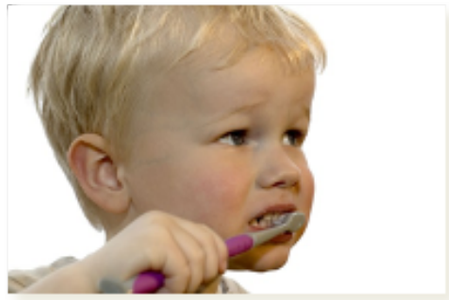


What kind of image transformations can we perform?

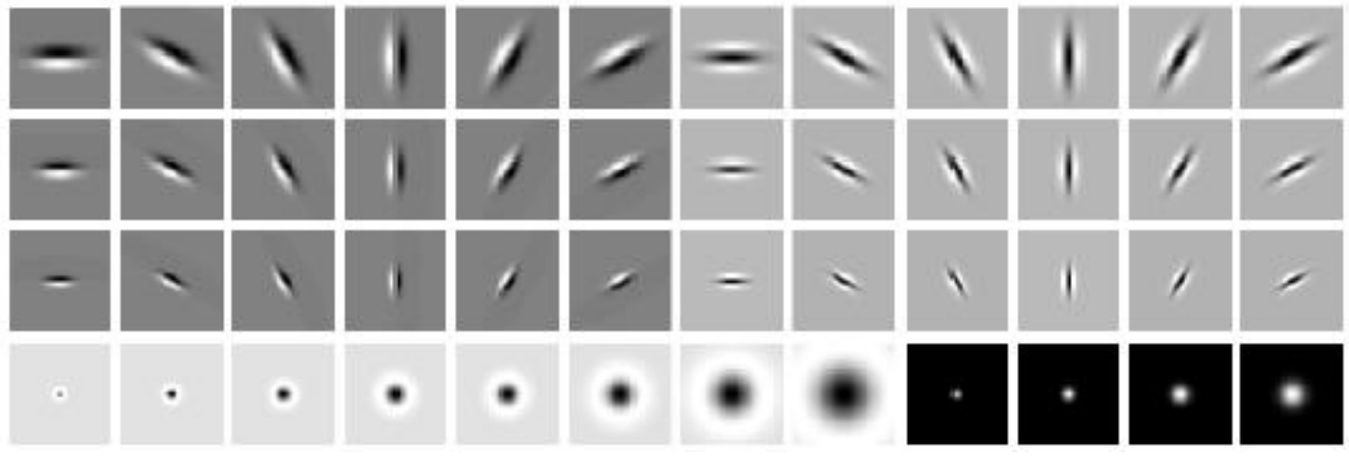
Filtering

Changes pixel values I[x,y]

Warping



Changes pixel coordinates (x,y)



2D transformations

translation

rotation

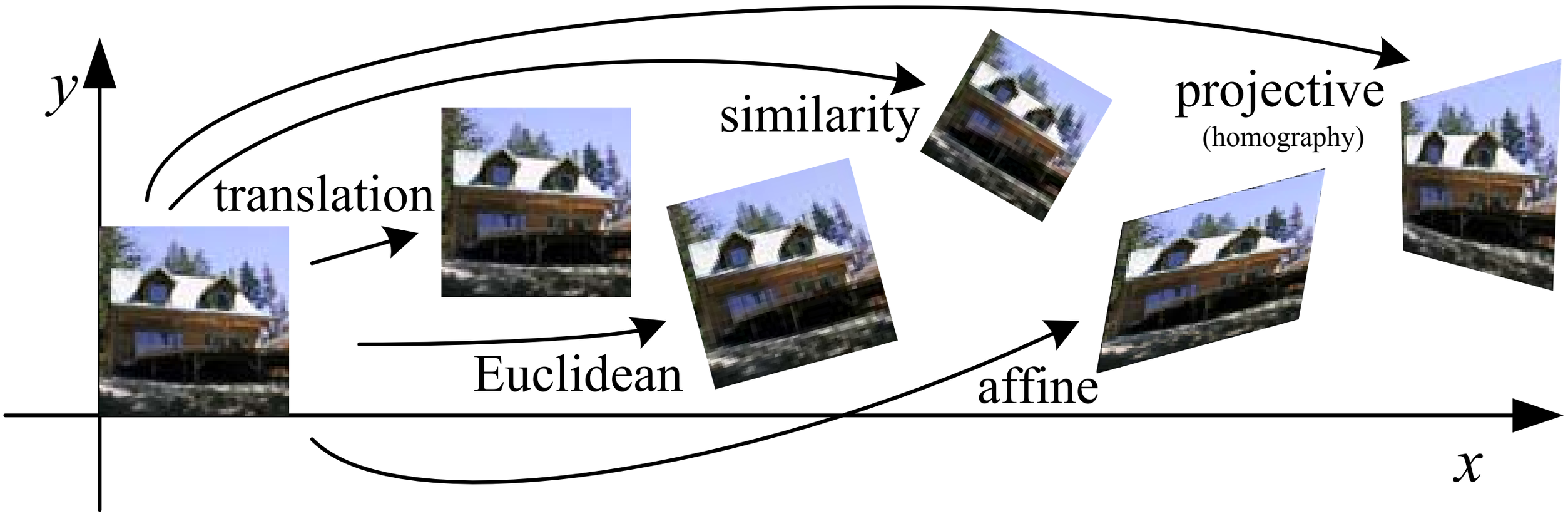
aspect

affine

perspective

cylindrical

Family of “nice” 2D transformations (can be represented with a matrix)

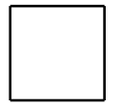


Question: which of these transformations are linear-shift-invariant?

translation is; rotation breaks it

(rotating + then shifting image does not equal shifting then rotating)

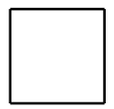
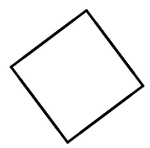
Family of 2D warps

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

$$x' = x + t_x$$
$$y' = y + t_y$$

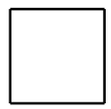
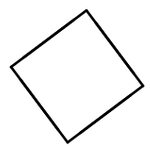
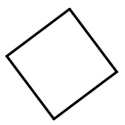
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Family of 2D warps

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	3	lengths	

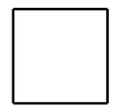
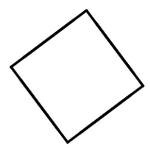
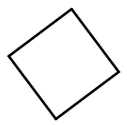
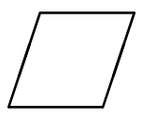
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Family of 2D warps

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	3	lengths	
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	4	angles	

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

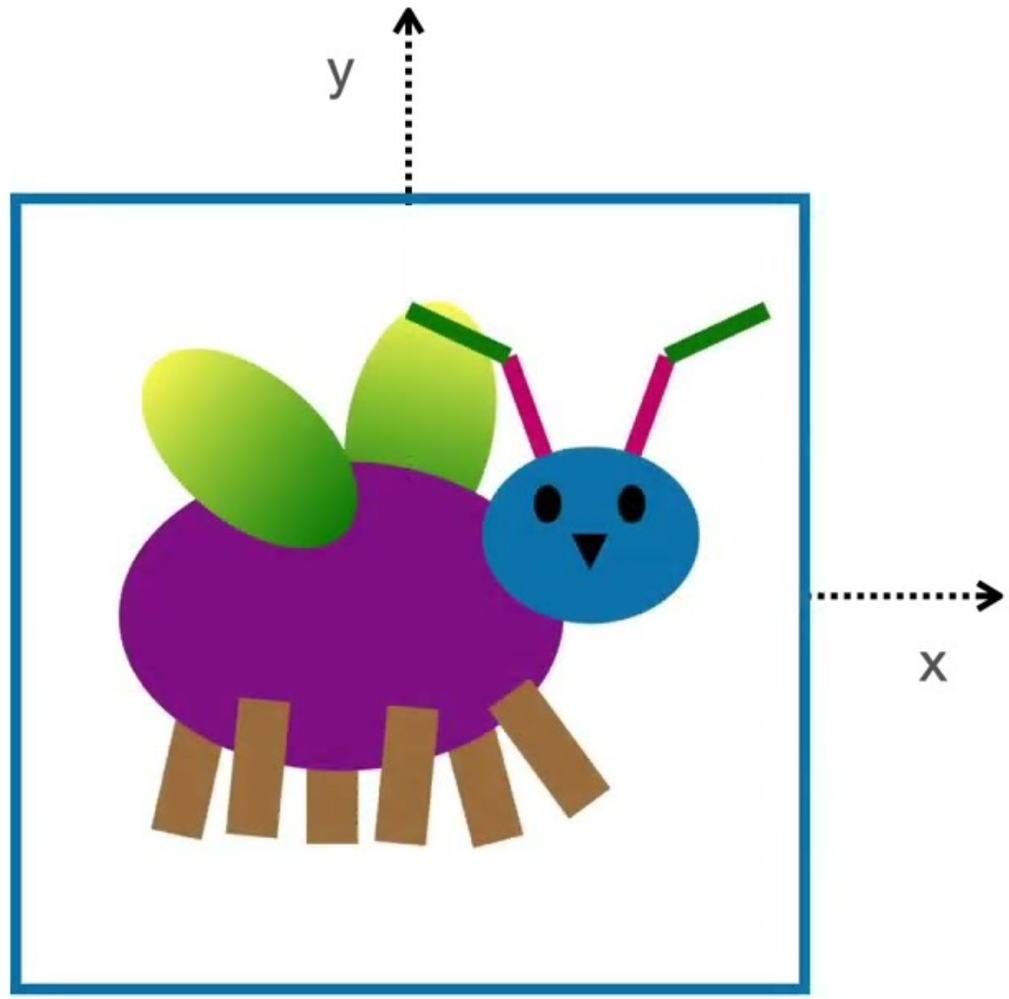
Family of 2D warps

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	3	lengths	
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	4	angles	
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{2\times 3}$	6	parallelism	

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

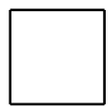
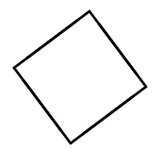
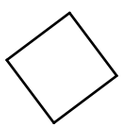
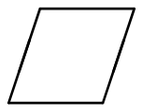
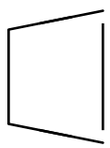
Think of as change of basis and 2D translation

Cool video (from Steve Seitz at UW)

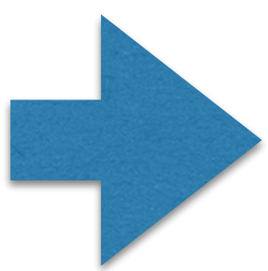


https://www.youtube.com/watch?v=AheaTd_l5Is&list=PLWfDJ5nla8UpwShx-lzLJqcp575fKpsSO&index=1

Family of 2D warps

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

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



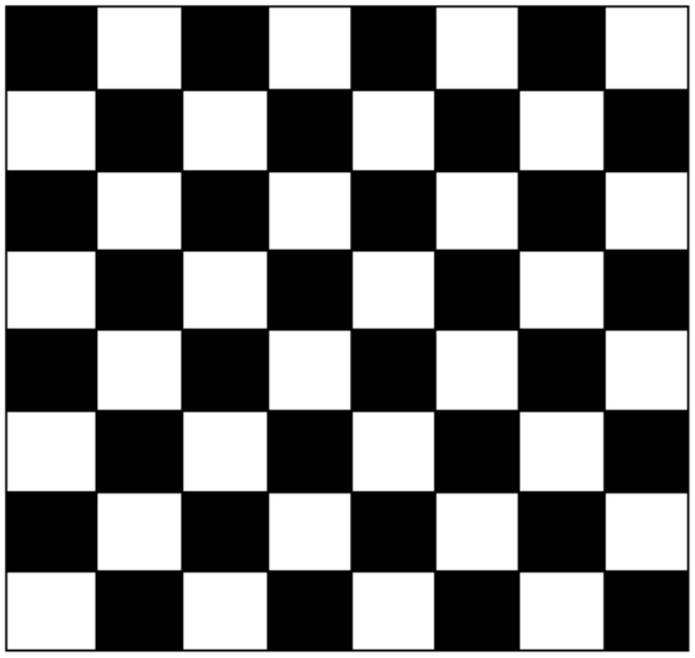
$$x' = \frac{ax + by + c}{gx + hy + i}$$

$$y' = \frac{dx + ey + f}{gx + hy + i}$$

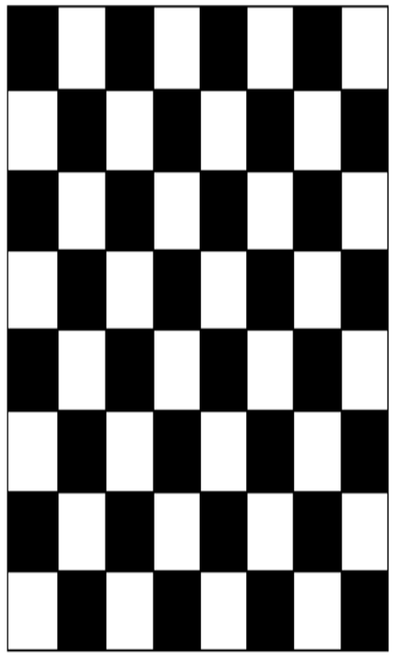
"homogenous 2D coordinates"

(for now, think of as shorthand for…)

Important 3Dish property captured by 2D affine warps: *foreshortening*



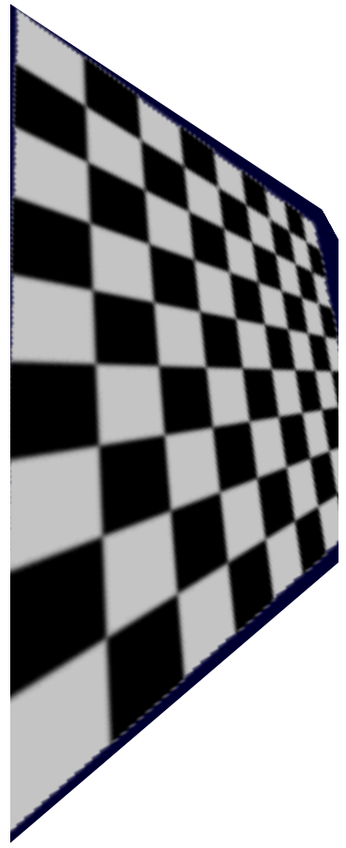
Fronto-parallel view



Foreshortened view

Rotation of far-away plane

Affine warp



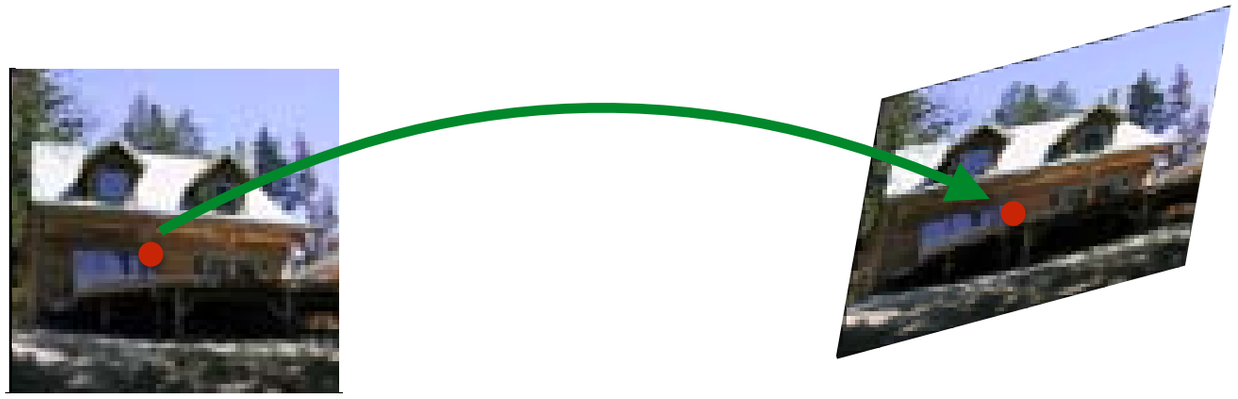
Perspective view

Rotation of close-by plane

Projective warp

(part of plane that gets further away becomes smaller)

Forward vs inverse warps



forward warp

“splat” color onto nearby grid points
(might still result in artifacts/holes)

inverse warp

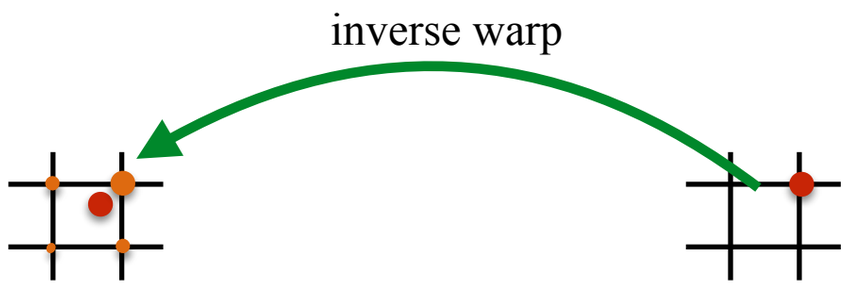
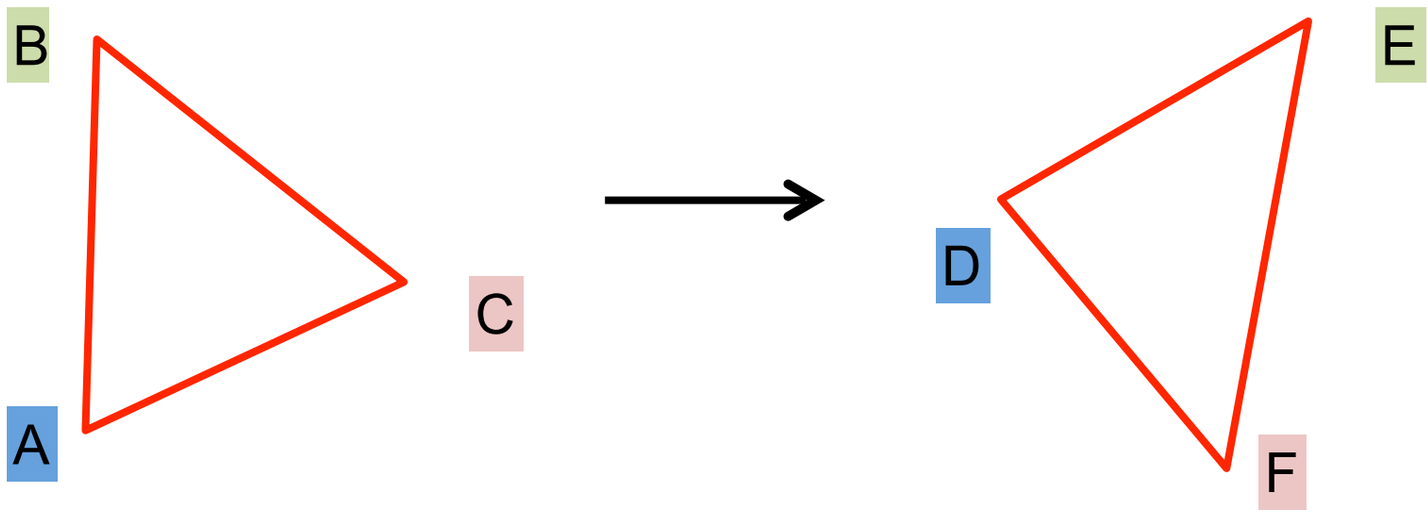


figure out from where each grid point should “steal” color
bilinearly interpolate
https://en.wikipedia.org/wiki/Bilinear_interpolation

[you’ll use this in HW3!]

Estimating transformations from point correspondences

Suppose we have two triangles: ABC and DEF.

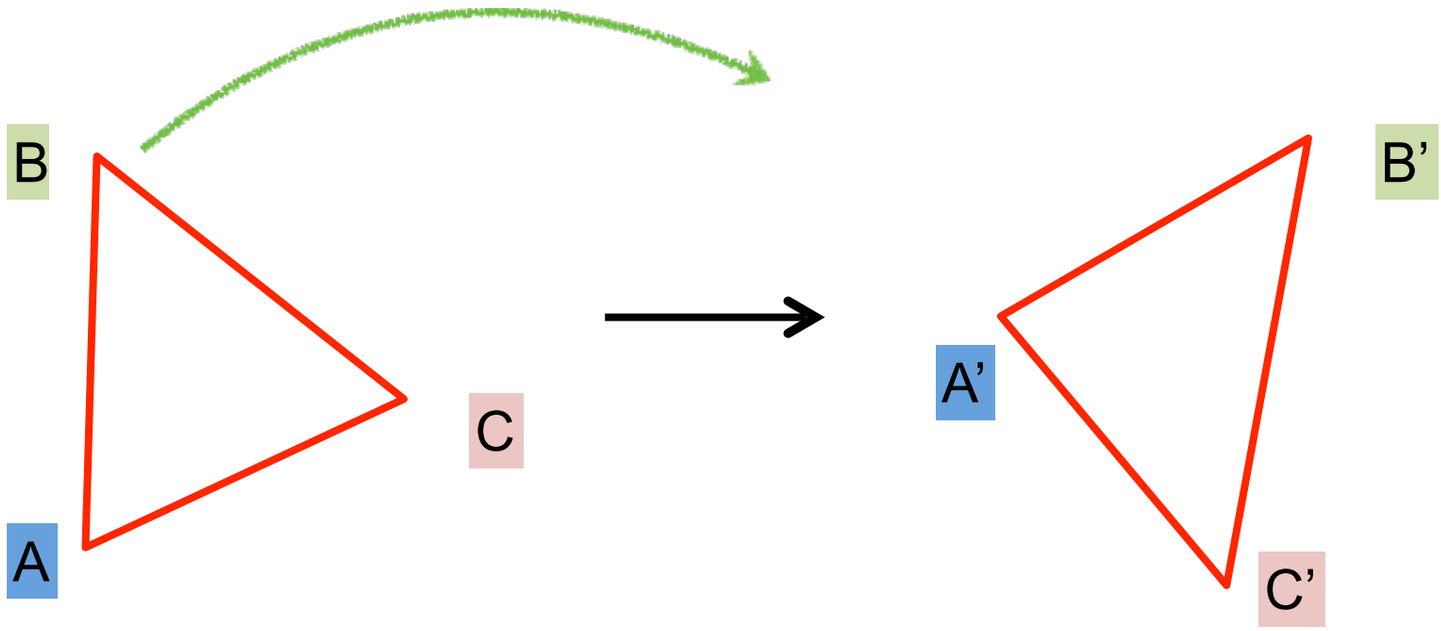


Can this be done with a euclidean / similarity / affine transformation?

Turns out any triangle can be warped to another triangle with an affine transform. Let's prove this useful fact.

Estimating transformations from point correspondences

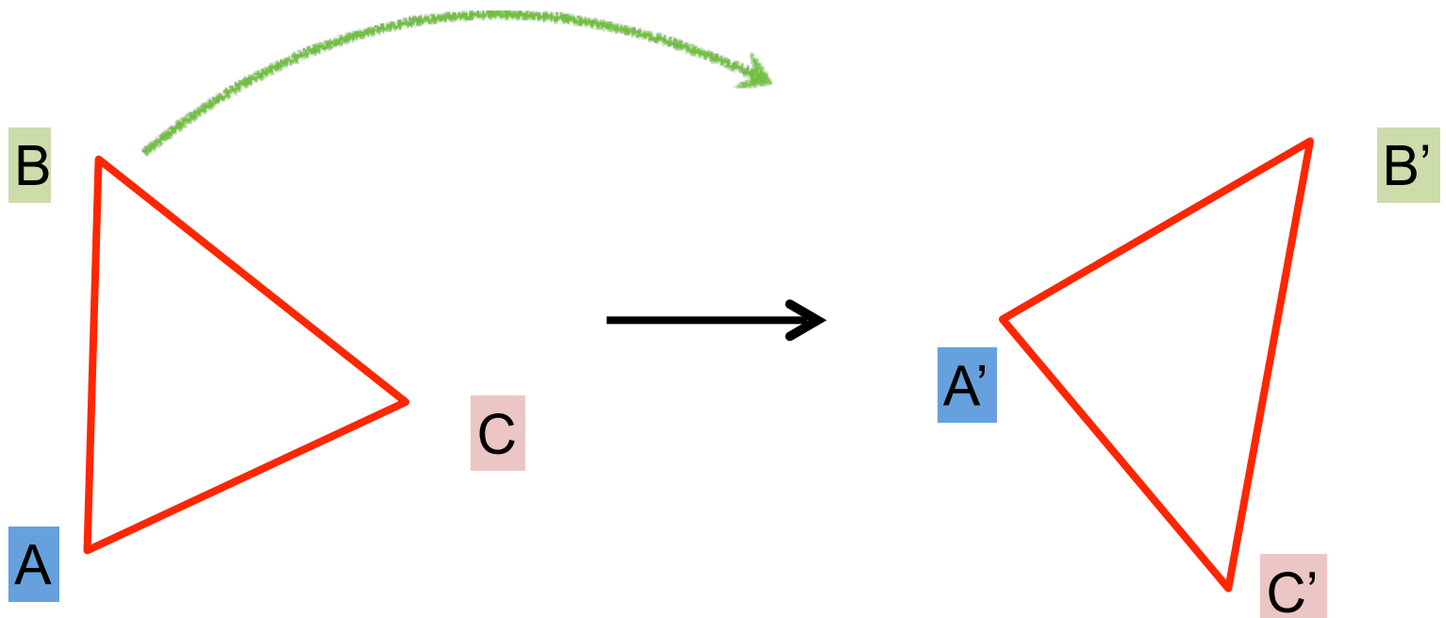
Suppose we have two triangles: ABC and A’B’C’.



$$\begin{bmatrix} A'_x \\ A'_y \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_x \\ A_y \\ 1 \end{bmatrix}$$

Given similar equations for B and C, how to estimate a_1 through a_6?

Least Squares Error



Euclidean
(L2) norm

$$\text{Error} = ||\text{Warp}(A,M) - A'||^2 + ||\text{Warp}(B,M) - B'||^2 + ||\text{Warp}(C,M) - C'||^2$$

predicted
location

measured
location

General form of linear least squares

(**Warning:** change of notation. x is a vector of parameters!)

$$E_{\mathrm{LLS}} = \sum_i \left|\boldsymbol{a}_i \boldsymbol{x} - \boldsymbol{b}_i\right|^2$$

$$= \left\|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\right\|^2 \qquad \text{(matrix form)}$$

Determining unknown transformations

Affine transformation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Why can we drop the last line?

Vectorize transformation parameters:

Stack equations from point correspondences:

$$\begin{bmatrix} x' \\ y' \\ \vdots \\ x' \\ y' \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \\ \vdots & & & & & \\ x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{bmatrix}$$

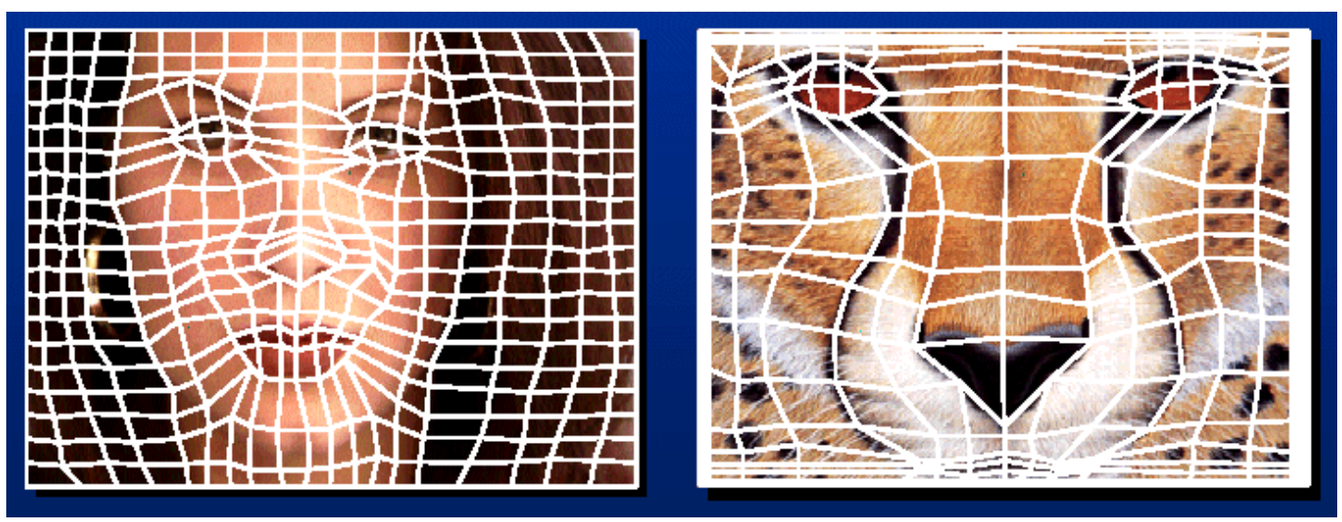
$$\boldsymbol{b} \qquad \mathbf{A} \qquad \boldsymbol{x}$$

Notation in system form:

```
x = numpy.linalg.solve(A, b)
```

What's the minimimum number of points needed to estimate an affine warp?

Image warping via *peicewise affine warps*



Piecewise affine warps (cut each quadrilateral into 2 triangles)

Example application: shape modeling

D'Arcy Thompson
"On Growth and Form" 1915

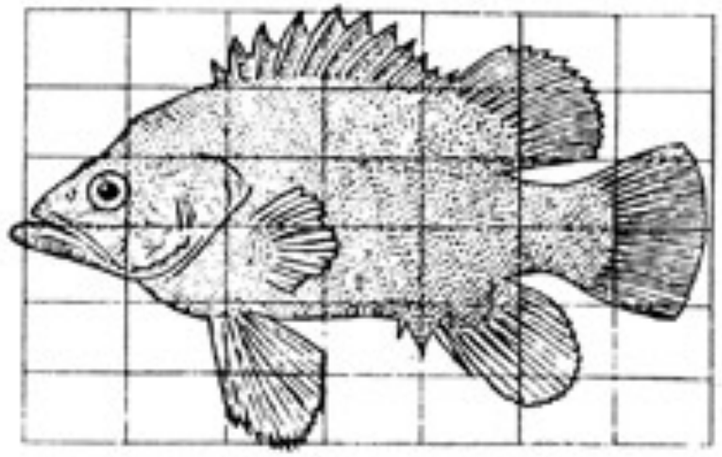


Fig. 150. *Polyprion.*

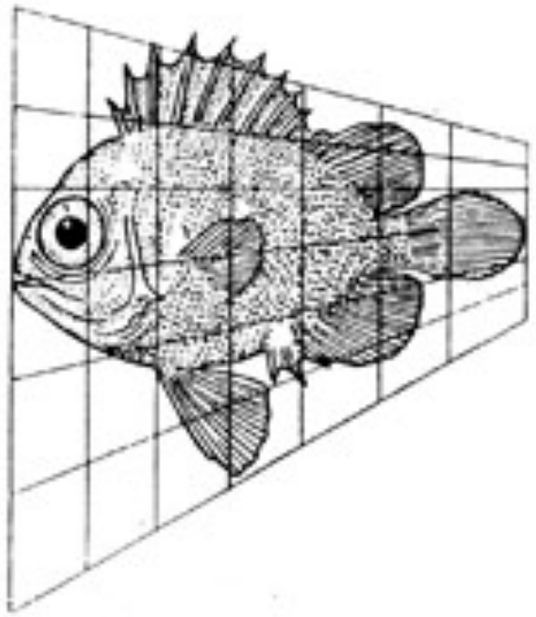


Fig. 151. *Pseudopriacanthus altus.*

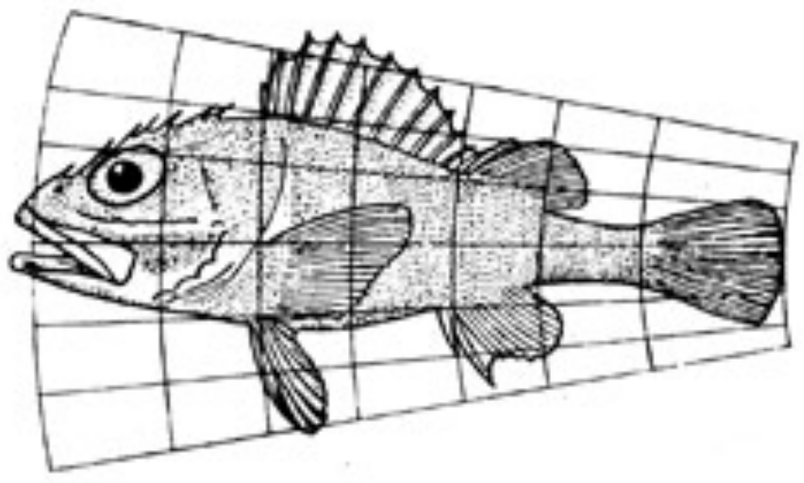


Fig. 152. *Scorpaena* sp.

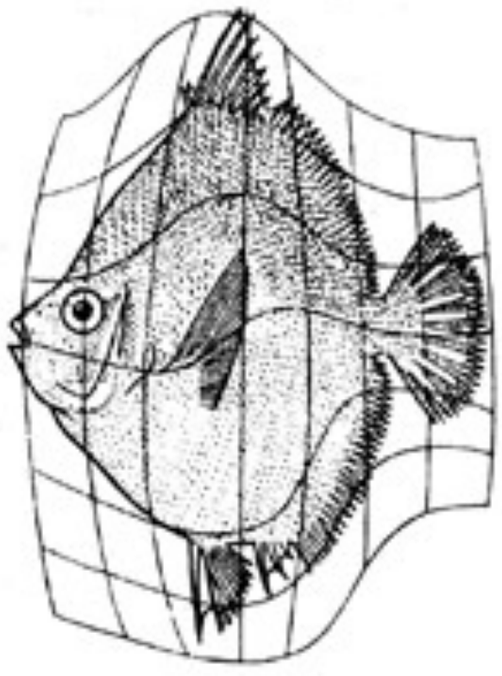
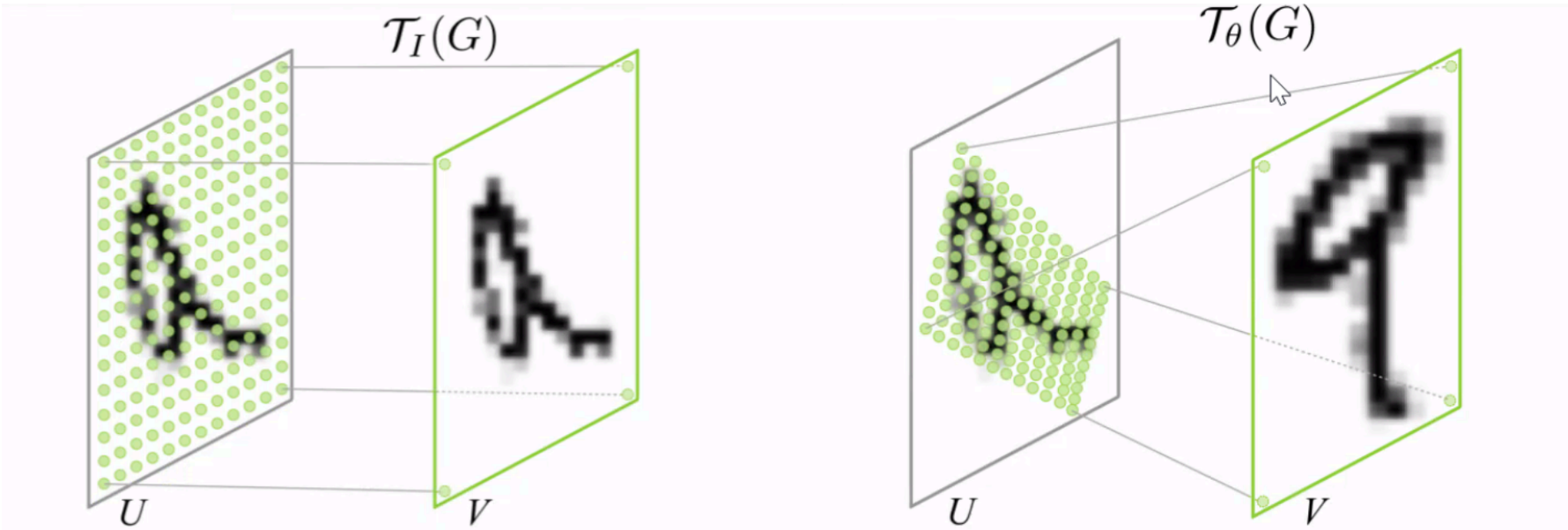
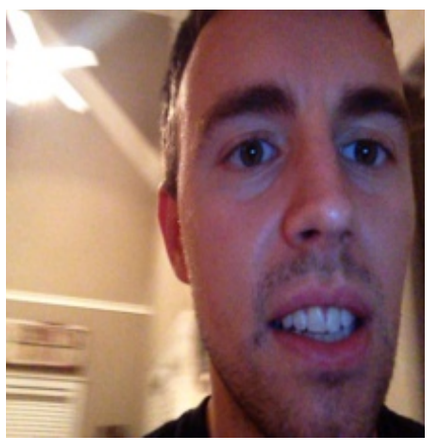


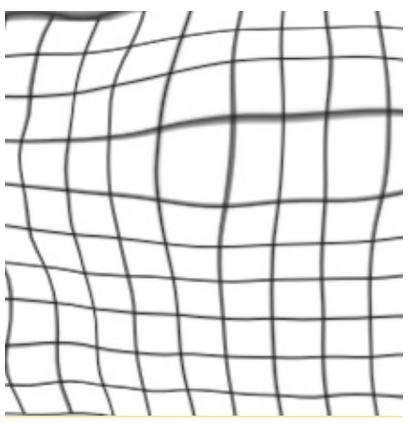
Fig. 153. *Antigonia capros.*

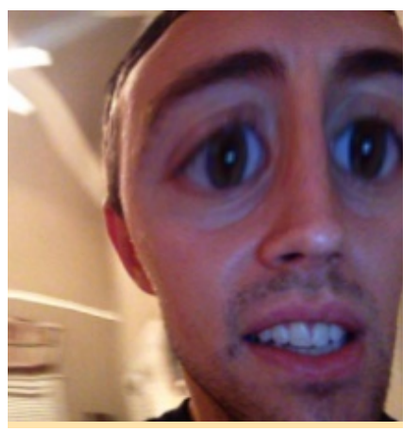
Application: spatial transformer networks



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







classify eye_gaze direction

Spatial Transformer Networks

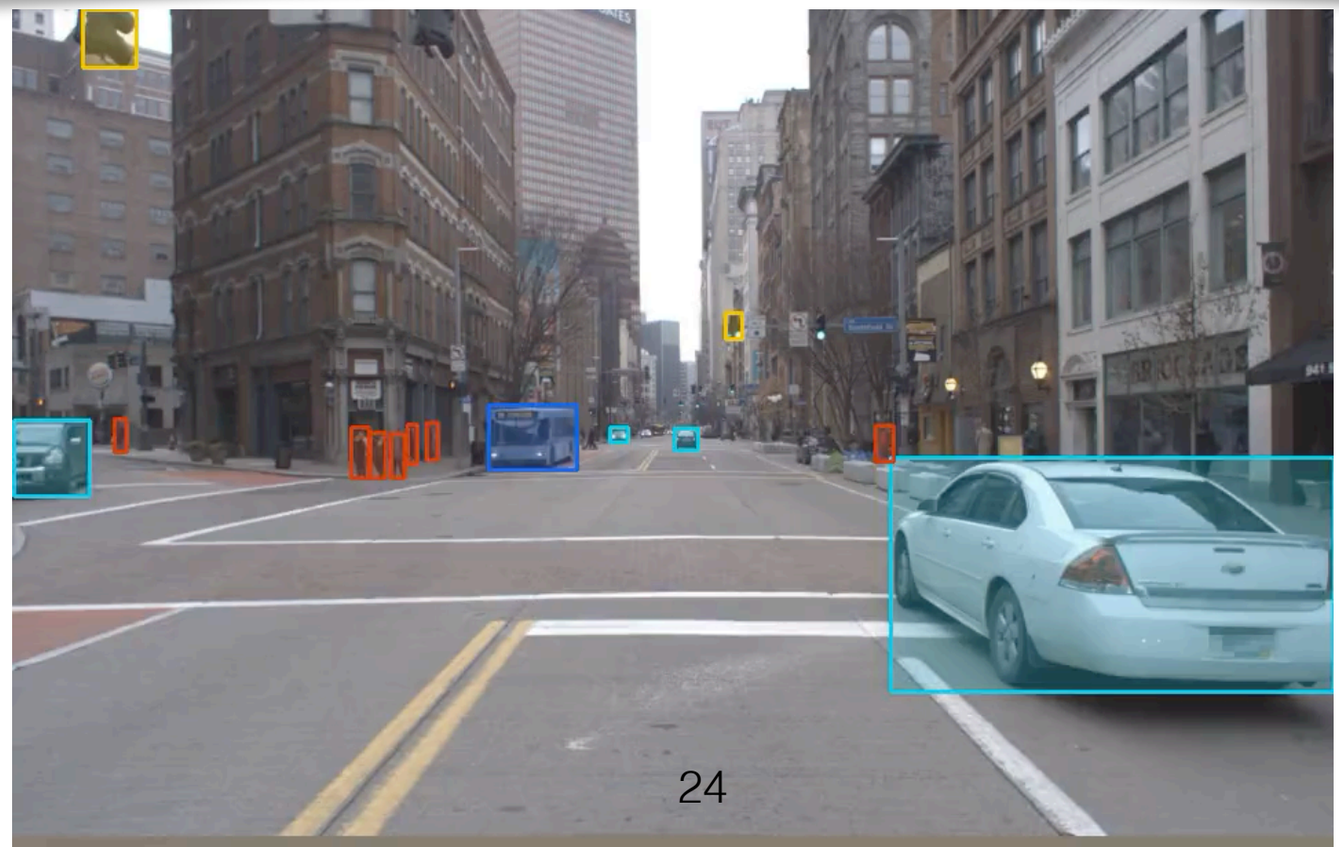
Max Jaderberg Karen Simonyan Andrew Zisserman Koray Kavukcuoglu

Google DeepMind, London, UK
{jaderberg,simonyan,zisserman,korayk}@google.com

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

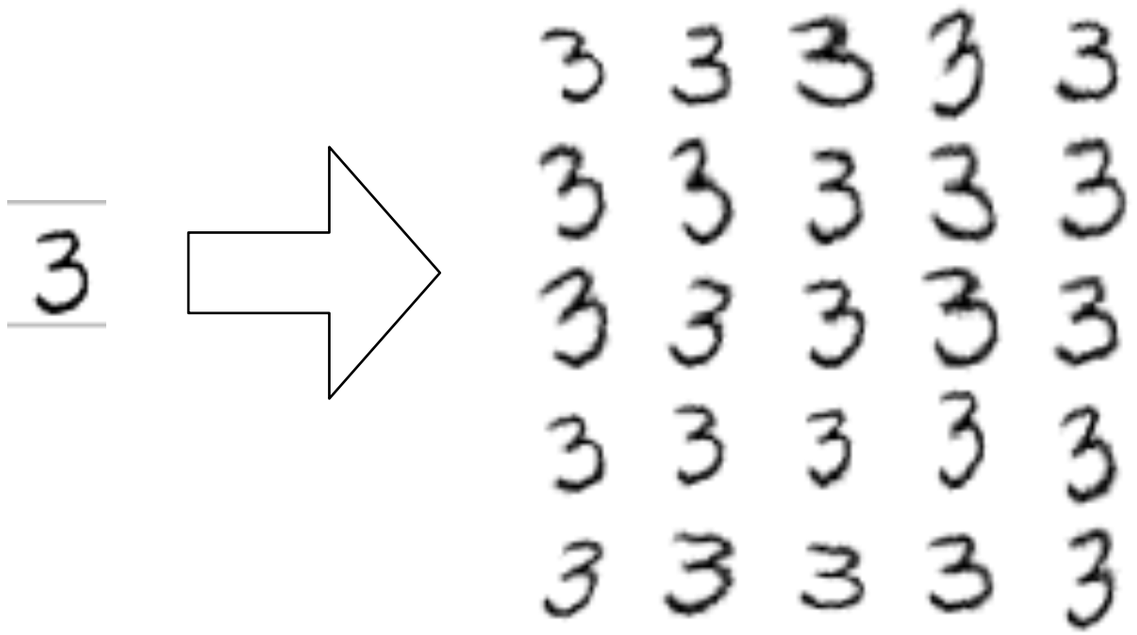
Adrià Recasens*[1], Petr Kellnhofer*[1], Simon Stent[2], Wojciech Matusik[1], and Antonio Torralba[1]

Application: “attentional imaging”



ICCV 21

Application: data augmentation



Hypothesis: we'll see even more of this in the future!

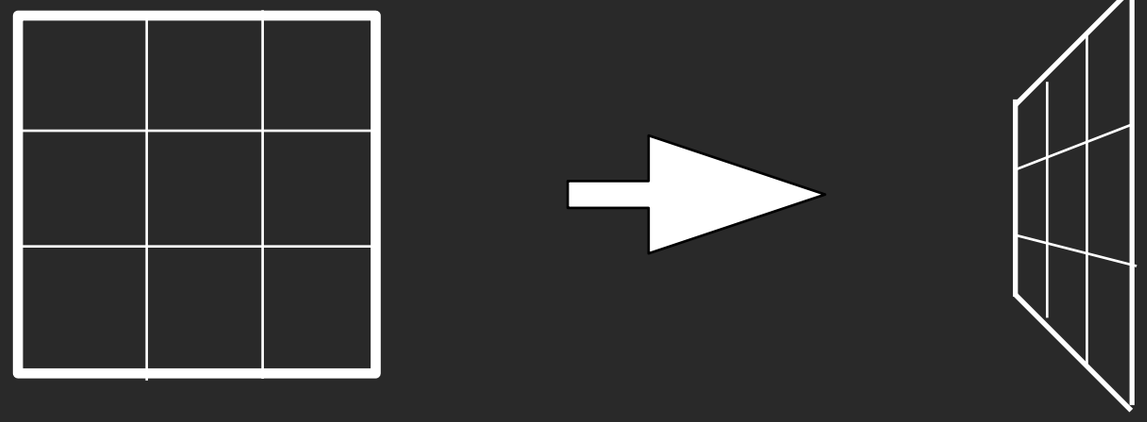
Data augmentation with projective warps

Cabinet

Cargo Container

Dryer

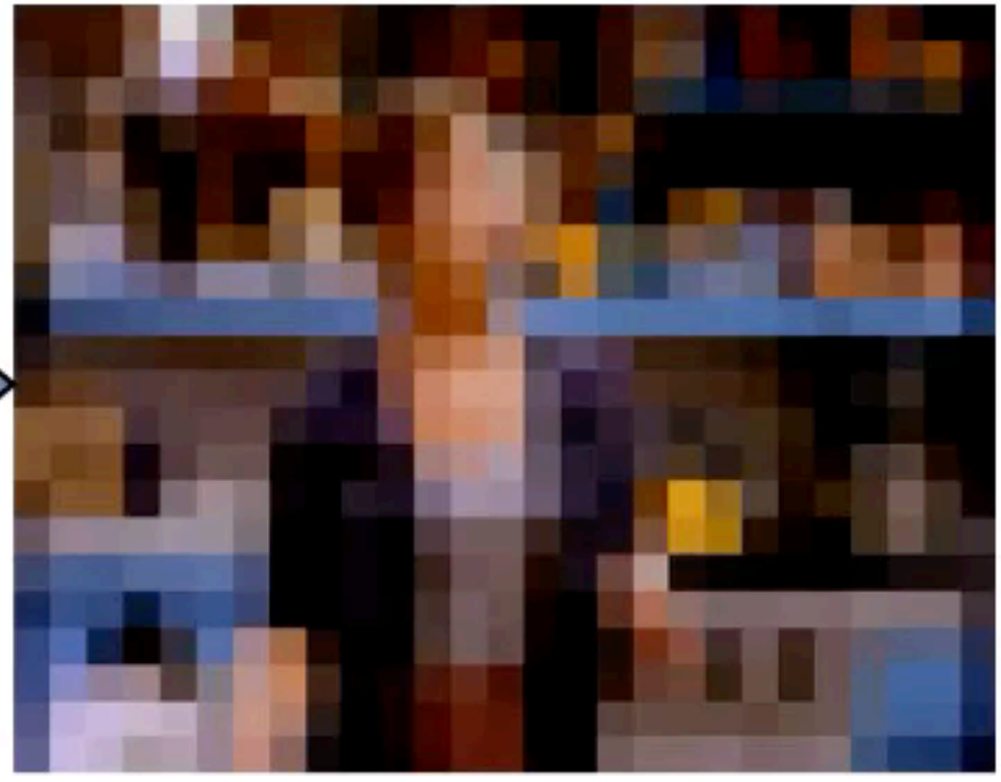
Cardboard Box



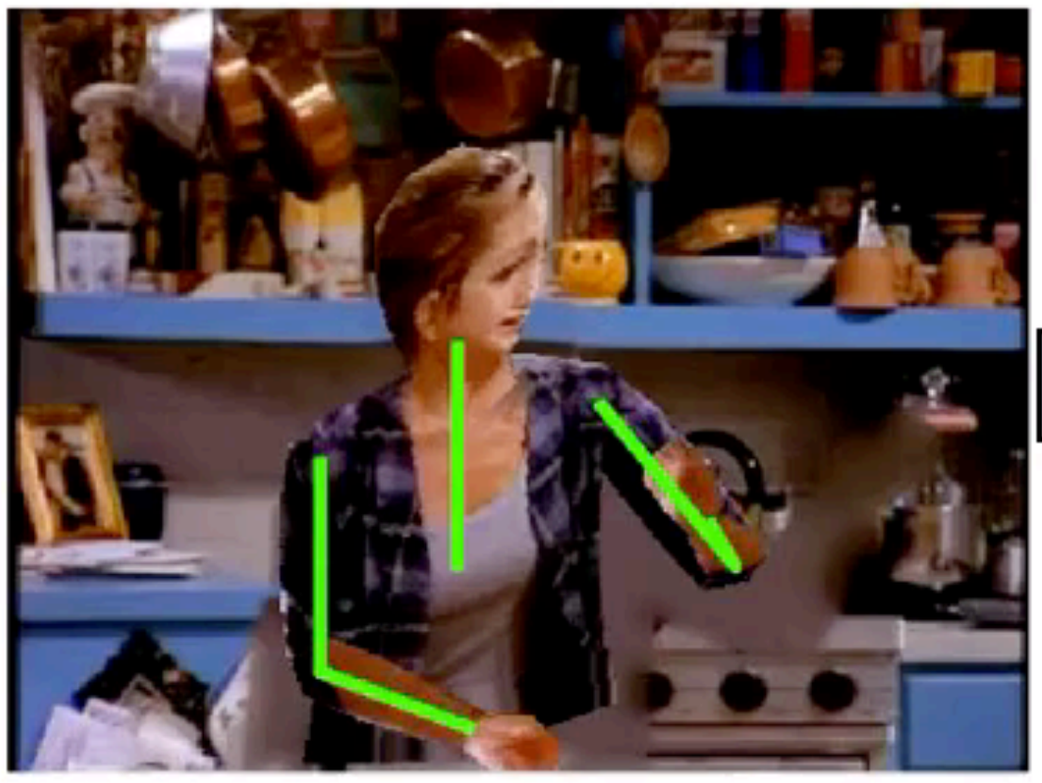
original

low resolution

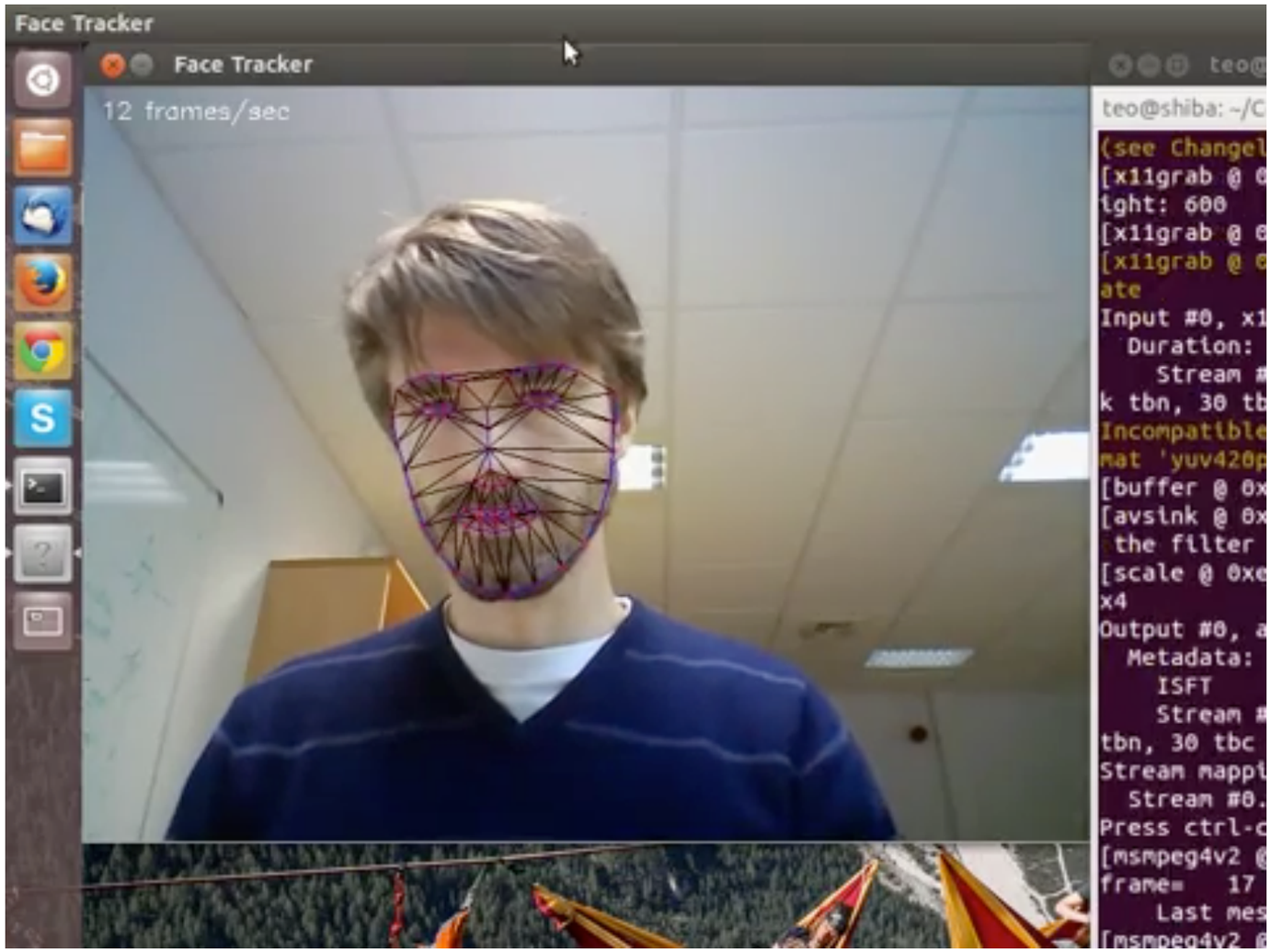
test video



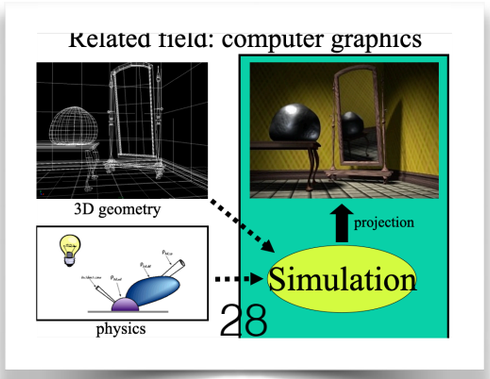
best match



Application: Tracking (HW2)!



Really instance of vision as inverse graphics or “analysis by synthesis”; interpret the world by *optimizing* for best fit of model

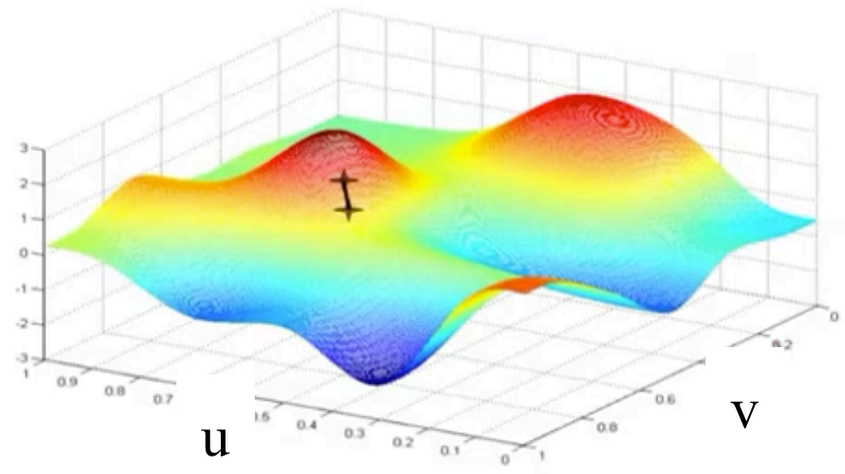


Lucas Kanade Alignment

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

template

E(u,v)



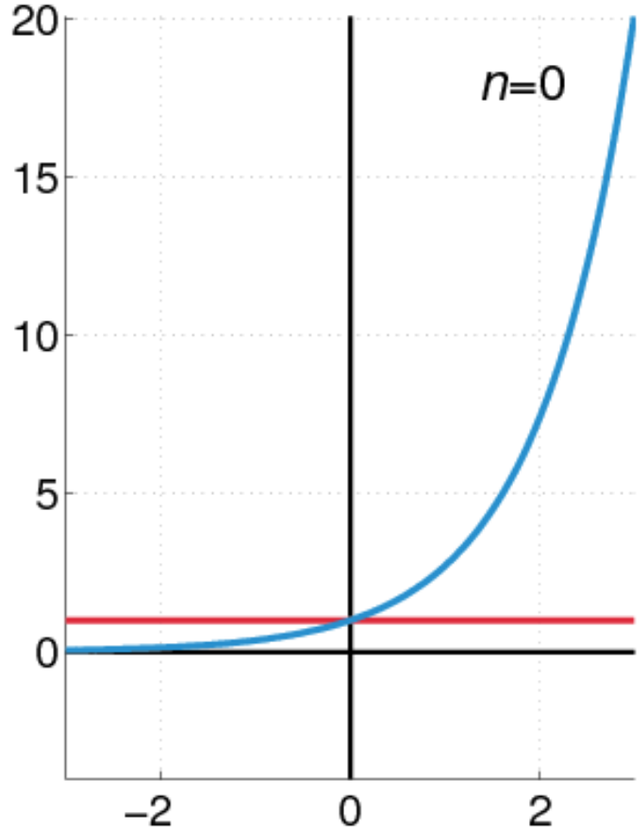
What is the transformation that is being searched over?

What kind of optimization problem is this?

Background: nonlinear optimization

Apply standard linear optimization tricks after using Taylor series approximation

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



How do we generalize the above to a multivariate function f(u,v)?

Generalize first derivative to gradient *vector* and second derivative to Hessian *matrix*

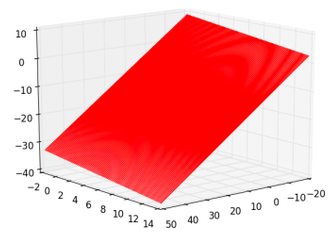
Background: nonlinear optimization

$$\min_{\mathbf{u}} f(\mathbf{u}) \quad \text{where } \mathbf{u} = (u,v)$$

1. First order method (gradient descent)

$$\begin{bmatrix} u \\ v \end{bmatrix} := \begin{bmatrix} u \\ v \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial f}{\partial u} \\ \frac{\partial f}{\partial v} \end{bmatrix}$$

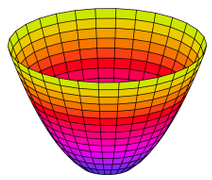
"fit a plane and take a step"



2. Second order method (Newton's method)

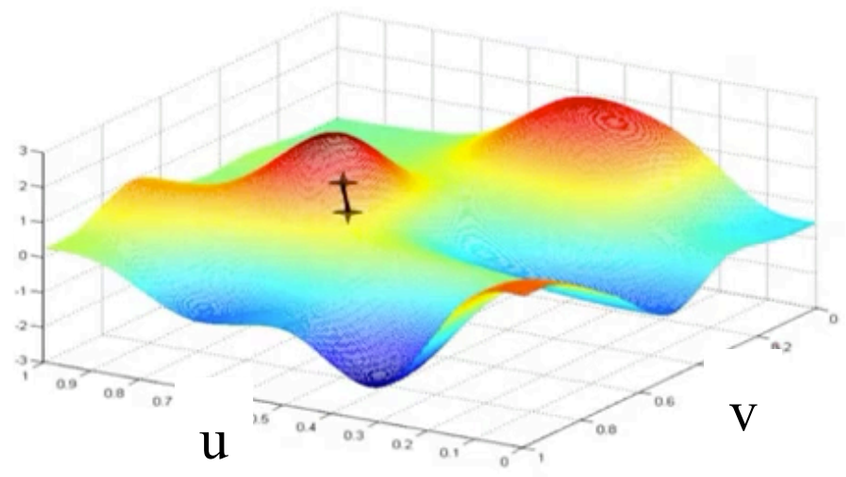
https://en.wikipedia.org/wiki/Newton's_method_in_optimization

"fit a paraboloid and jump to minimum"



$$\begin{bmatrix} u \\ v \end{bmatrix} := \begin{bmatrix} u \\ v \end{bmatrix} - \begin{bmatrix} \frac{\partial^2 f}{\partial u \partial u} & \frac{\partial^2 f}{\partial u \partial v} \\ \frac{\partial^2 f}{\partial v \partial u} & \frac{\partial^2 f}{\partial v \partial v} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial f}{\partial u} \\ \frac{\partial f}{\partial v} \end{bmatrix}$$

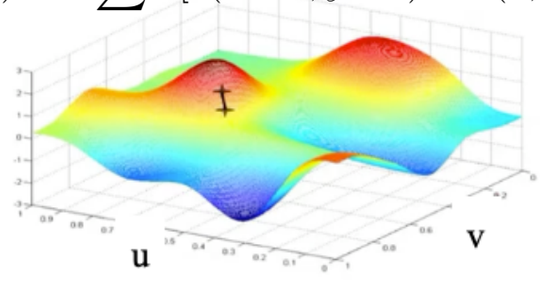
E(u,v)



“Background”: nonlinear least squares

$$E(u,v) = \frac{1}{2}\left\| \begin{matrix} f(u,v) \\ g(u,v) \end{matrix} \right\|^2 = \frac{1}{2}||\mathbf{f}(\mathbf{u})||^2$$

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



1. First order method (gradient descent)

$$\begin{bmatrix} u \\ v \end{bmatrix} := \begin{bmatrix} u \\ v \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} \\ \frac{\partial g}{\partial u} & \frac{\partial g}{\partial v} \end{bmatrix}^T \begin{bmatrix} f(u,v) \\ g(u,v) \end{bmatrix}$$

1.5 Gauss-Newton optimization

https://en.wikipedia.org/wiki/Gauss–Newton_algorithm

$$\begin{bmatrix} f(u+\Delta u, v+\Delta v) \\ g(u+\Delta u, v+\Delta v) \end{bmatrix} \approx \begin{bmatrix} f(u,v) \\ g(u,v) \end{bmatrix} + \begin{bmatrix} \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} \\ \frac{\partial g}{\partial u} & \frac{\partial g}{\partial v} \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}$$

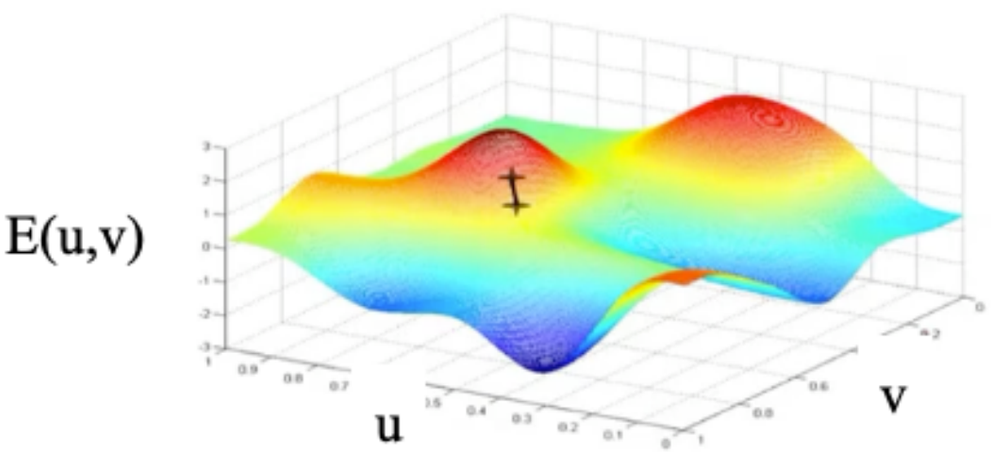
$$\mathbf{f}(\mathbf{u} + \mathbf{\Delta u}) \approx \mathbf{f}(\mathbf{u}) + \mathbf{J}(\mathbf{u})\mathbf{\Delta u}$$

$$\mathbf{u} := \mathbf{u} + \arg\min_{\mathbf{\Delta u}} \frac{1}{2}||\mathbf{f}(\mathbf{u}) + \mathbf{J}(\mathbf{u})\mathbf{\Delta u}||^2$$

Take derivative of error wrt Δu, Δv and set equal to 0;
get the performance of second-order optimizers with first-order compute!

Where we are headed…

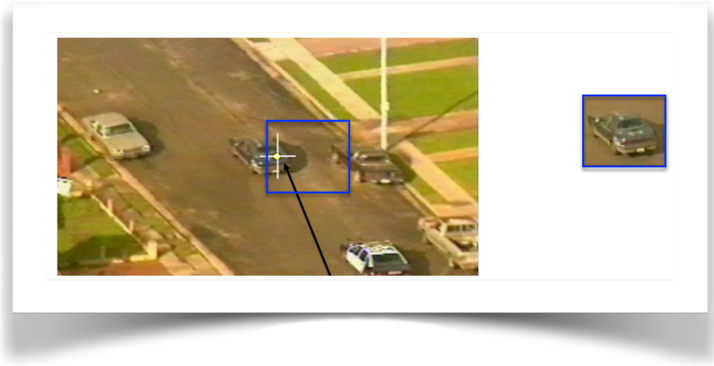
$$\min_{u,v} \sum_{(x,y)\in W} [I(x+u, y+v) - T(x,y)]^2$$



initialization

template

Lucas Kanade alignment



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

Start with initial guess of (u,v). Linearize image about this guess

$$I(x+u+\Delta u, y+v+\Delta v) \approx I(x+u, y+v) + \frac{\partial I(x+u, y+v)}{\partial x}\Delta u + \frac{\partial I(x+u, y+v)}{\partial y}\Delta v$$

$$I(\mathbf{x}+\mathbf{u}+\mathbf{\Delta u}) \approx I(\mathbf{x}+\mathbf{u}) + \nabla I(\mathbf{x}+\mathbf{u})^T \mathbf{\Delta u}$$

Plug back into error term and solve for $\mathbf{\Delta u}$'s that minimizes the squared error

$$\min_{\mathbf{\Delta u}} \sum_{\mathbf{x}} [I(\mathbf{x}+\mathbf{u}) + \nabla I(\mathbf{x}+\mathbf{u})^T \mathbf{\Delta u} - T(\mathbf{x})]^2$$

Lucas Kanade alignment

$$\min_{\mathbf{\Delta u}} \sum_{\mathbf{x}} [I(\mathbf{x}+\mathbf{u}) + \nabla I(\mathbf{x}+\mathbf{u})^T \mathbf{\Delta u} - T(\mathbf{x})]^2$$

Treat **u** as fixed wrt to optimization. Define $\tilde{I}(\mathbf{x})$ = I(**x**+**u**).

$$\min_{\Delta u, \Delta v} \sum_{x,y} [\tilde{I}(x,y) + \tilde{I}_x(x,y)\Delta u + \tilde{I}_y(x,y)\Delta v - T(x,y)]^2$$

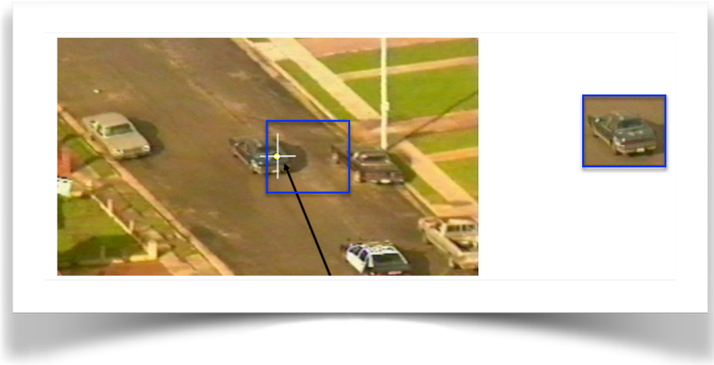
Define D(x,y) = T(x,y) - $\tilde{I}$(x,y).

$$\min_{\Delta u, \Delta v} \sum_{x,y} [\tilde{I}_x(x,y)\Delta u + \tilde{I}_y(x,y)\Delta v - D(x,y)]^2$$

Take derivatives and set them equal to 0

$$\frac{\partial}{\partial \Delta u} = 2 \sum_{x,y} [\tilde{I}_x(x,y)\Delta u + \tilde{I}_y(x,y)\Delta v - D(x,y)]\tilde{I}_x(x,y) = 0$$

Lucas Kanade alignment



Form matrix equation

$$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$$

Aside: how is the A matrix different than a Hessian matrix?

$$\left[\frac{\partial I(x,y)}{\partial x}\right]\left[\frac{\partial I(x,y)}{\partial x}\right] \neq \frac{\partial^2 I(x,y)}{\partial x \partial x}$$

"Ax=b"

"$x=A^{-1}b$"

Lucas Kanade Alignment

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

initialization

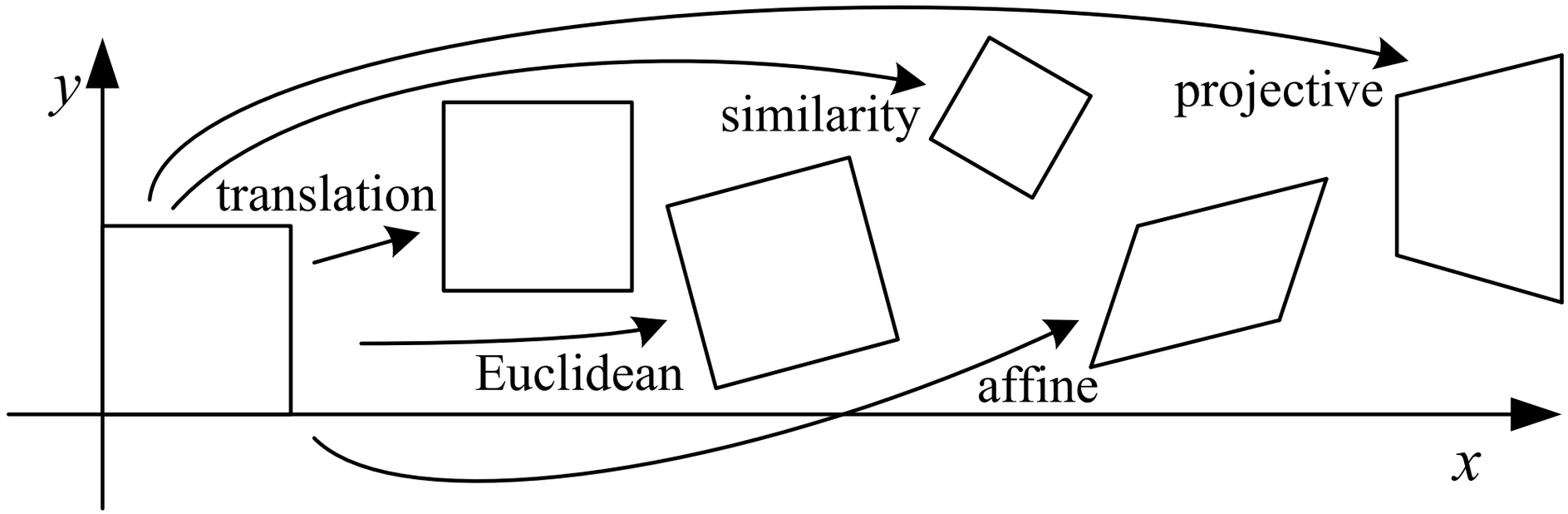
template

HW: Tracking by iterative template alignment

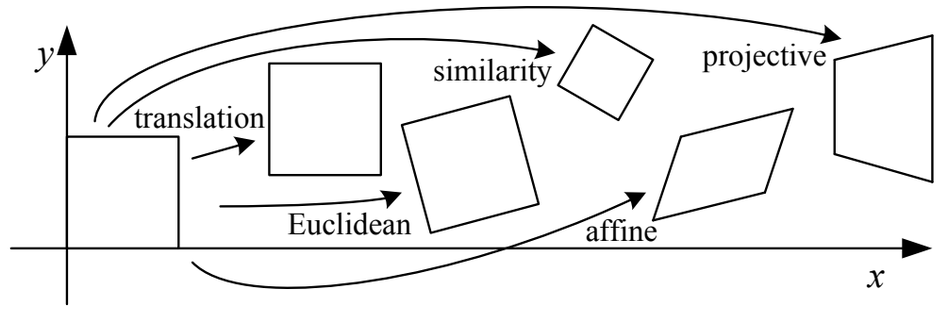
Start with template from first frame and repeat:

1. Align template to new frame with Lucas Kanade
2. Update template with new frame (or not?)

What about other warps?



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



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

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: $W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

Seminal reference

Lucas-Kanade 20 Years On: A Unifying Framework

SIMON BAKER AND IAIN MATTHEWS
The Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
simonb@cs.cmu.edu
iainm@cs.cmu.edu

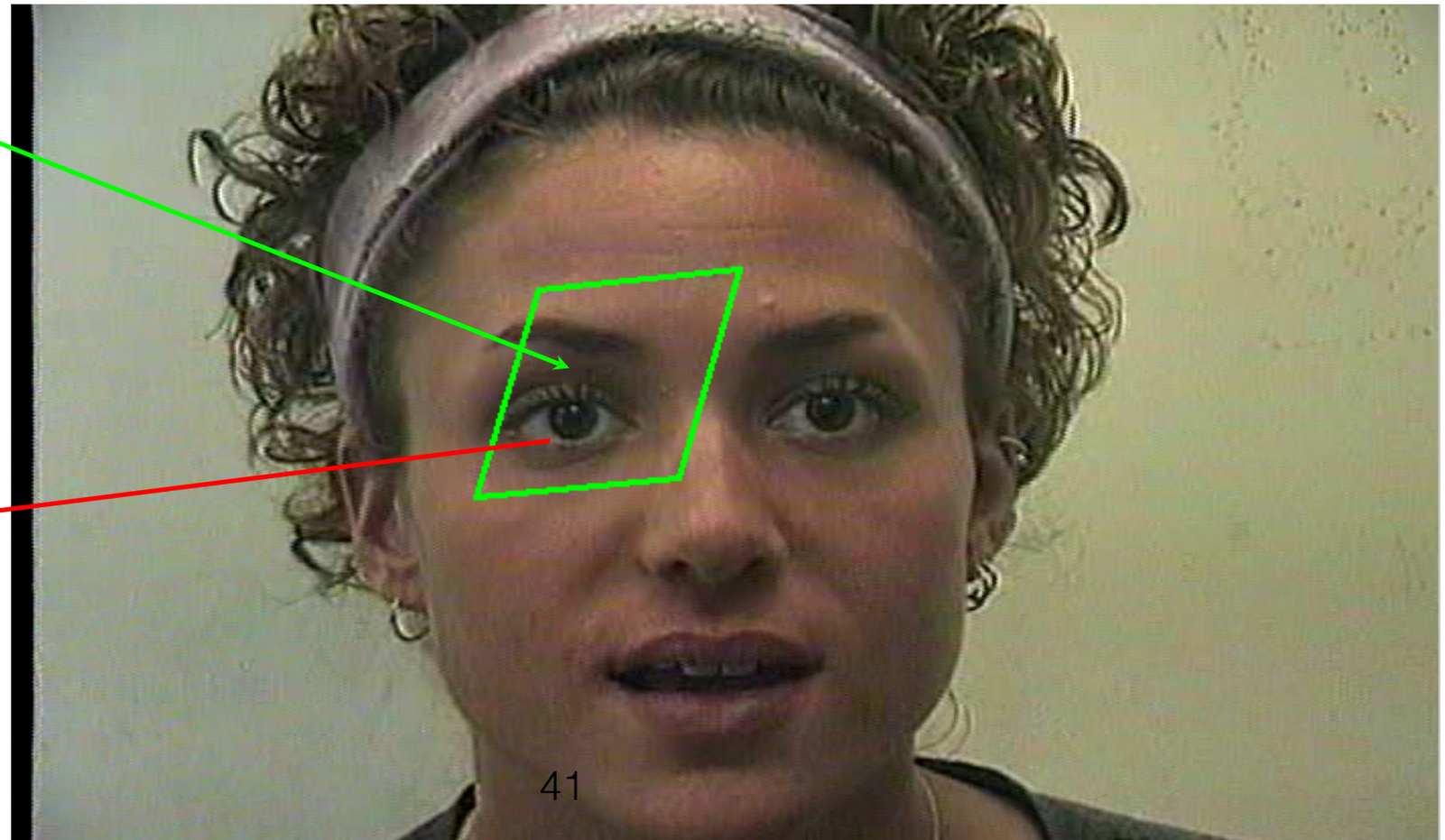
Received July 10, 2002; Revised February 6, 2003; Accepted February 7, 2003

(will follow notation)

Warp image to get $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$; then compute $[I(W(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2$

Template, $T(\mathbf{x})$

Warped, $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$



Nonlinear least squares

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

Apply taylor-series expansion to vector-valued function $\mathbf{W}$

(for simplicity, assume 2D rotation)

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

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

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

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

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

Taylor expansion of warped image

1-parameter warp (e.g., rotation)

$$I(W(\mathbf{x};p+\Delta p)) \approx I(W(\mathbf{x};p)) + \left[\frac{\partial I(W(\mathbf{x};p))}{\partial x} \quad \frac{\partial I(W(\mathbf{x};p))}{\partial y}\right] \left[\begin{array}{c}\frac{\partial W_x(\mathbf{x};p)}{\partial p} \\ \frac{\partial W_y(\mathbf{x};p)}{\partial p}\end{array}\right] \Delta p$$

current warped image

current warped image gradient

jacobian of warp

parameter update

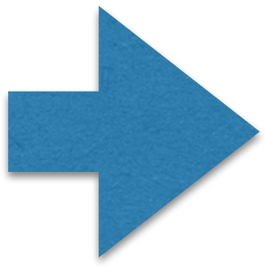
multi-parameter warp (e.g., affine)

$$I(W(\mathbf{x};\mathbf{p}+\mathbf{\Delta p})) \approx I(W(\mathbf{x};\mathbf{p})) + \left[\frac{\partial I(W(\mathbf{x};\mathbf{p}))}{\partial x} \quad \frac{\partial I(W(\mathbf{x};\mathbf{p}))}{\partial y}\right] \left[\begin{array}{cc}\frac{\partial W_x(\mathbf{x};\mathbf{p})}{\partial p_1} & \frac{\partial W_x(\mathbf{x};\mathbf{p})}{\partial p_2} \\ \frac{\partial W_y(\mathbf{x};\mathbf{p})}{\partial p_1} & \frac{\partial W_y(\mathbf{x};\mathbf{p})}{\partial p_2}\end{array}\right] \left[\begin{array}{c}\Delta p_1 \\ \Delta p_2\end{array}\right]$$

$\nabla I(W(\mathbf{x};\mathbf{p}))$ $\quad$ $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ $\quad$ $\mathbf{\Delta p}$

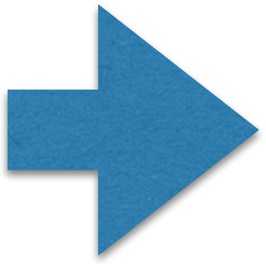
Example: jacobian of affine warp

$$W(\mathbf{x};\mathbf{p}) = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



$$W_x = (1+p_1)x + p_3 y + p_5$$
$$W_y = p_2 x + (1+p_4)y + p_6$$

$$\frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots \end{bmatrix}$$



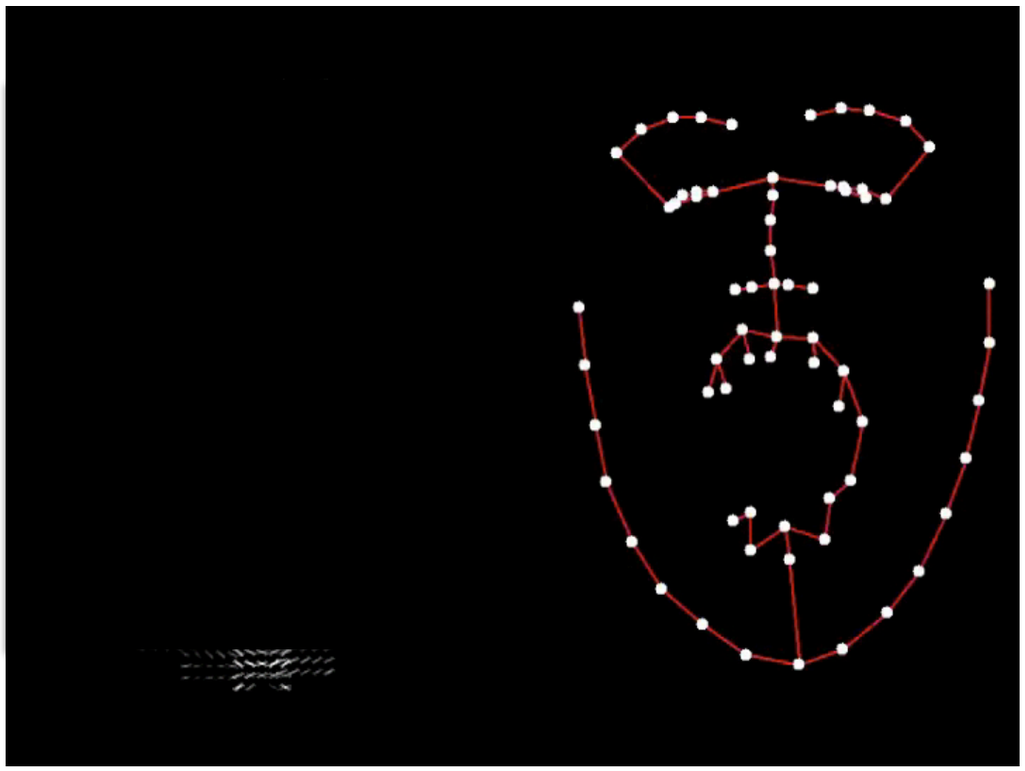
$$\frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$$

Above affine parameterization is better conditioned for optimization because all-zero parameters default to the identity transformation

Aside: low-dimensional parameterizations of warps

$$W(\mathbf{x}; p)$$

Figure 1: Example face image annotated with landmarks



Active Shape Models—Their Training and Application

T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham*
Department of Medical Biophysics, University of Manchester, Oxford Road, Manchester M13 9PT, England

Received July 29, 1992; accepted April 12, 1994

Apply SVD (or PCA) to collection of (x,y) facial landmarks to learn *eigenshapes*

$$\mathbf{s} = \begin{bmatrix} x_1 & y_2 & x_2 & y_2 \ldots \end{bmatrix}$$

$$\mathbf{s} = \mathbf{s}_0 + p_1 \mathbf{s}_1 + p_2 \mathbf{s}_2 + \ldots$$

$$Pr(\mathbf{s}) = N(\mathbf{s}; \mu, \Sigma)$$

Back to the big-picture

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})) - T(\mathbf{x})\,]^2$$

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

Gradient Descent Solution

Least squares problem: Minimize to solve for $\Delta\mathbf{p}$

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

Solution,

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

Error Image

Gradient

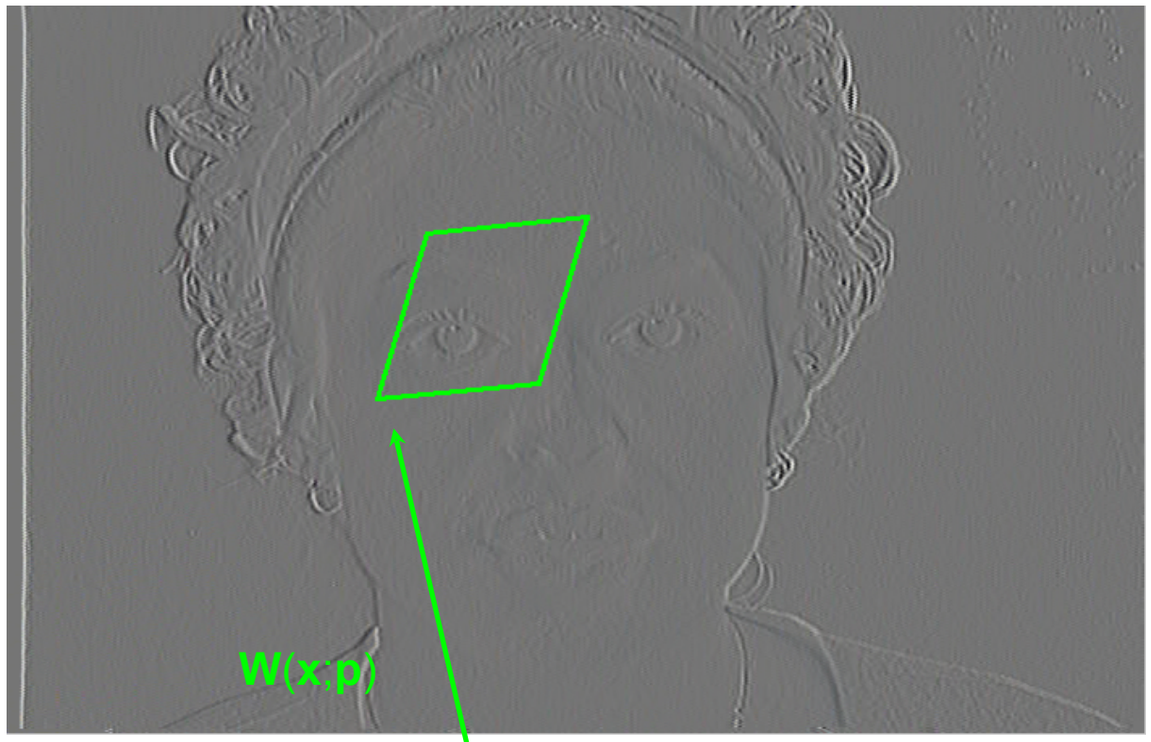
$$H = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

Jacobian

Approx hessian

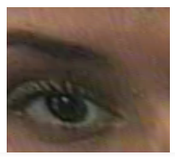
Gradient Images

- Compute image gradient ∇I



∇I_x

$I(\mathbf{W}(\mathbf{x};\mathbf{p}))$



∇I_y

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

Jacobian

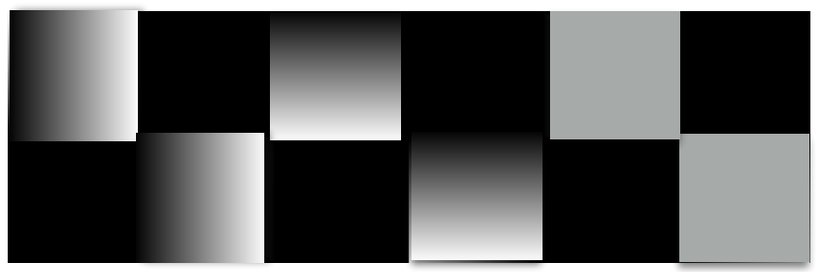
- Compute Jacobian

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{pmatrix}$$

(e.g., for affine warps)

$$\mathbf{W}(\mathbf{x};\mathbf{p}) = \begin{pmatrix} (1+p_1)\cdot x & + & p_3 \cdot y & + & p_5 \\ p_2 \cdot x & + & (1+p_4)\cdot y & + & p_6 \end{pmatrix} = \begin{pmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix} =$$

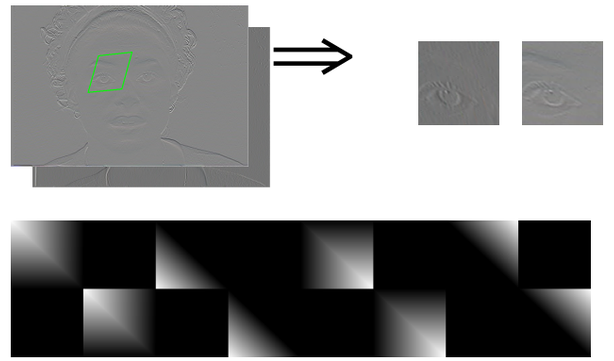


Lucas-Kanade Algorithm

1. Warp *I* with **W**(**x**;**p**) ⇒ *I*(**W**(**x**;**p**))

1. Compute error image *T*(**x**) - *I*(**W**(**x**;**p**))

1. Warp image ∇*I* with **W**(**x**;**p**)



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

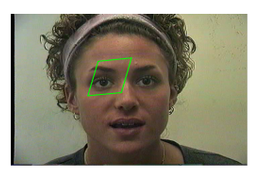
1. **Compute Approx Hessian**

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

1. Compute **Δp**

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

1. Update parameters **p** ← **p** + **Δp**



Important extensions

Lucas-Kanade 20 Years On: A Unifying Framework

SIMON BAKER AND IAIN MATTHEWS
The Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
simonb@cs.cmu.edu
iainm@cs.cmu.edu

image

template

Key insight: exploit flexibility of warping the image *or* warping the template

Overview

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

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = \begin{pmatrix} (1+p_1)\cdot x & + & p_3 \cdot y & + & p_5 \\ p_2 \cdot x & + & (1+p_4)\cdot y & + & p_6 \end{pmatrix}$$

replace "x" with "$(1 + \Delta p_1)x + \Delta p_3 y + \Delta p5$" and "y" with …

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta\mathbf{p}) =$$

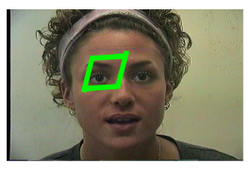
$$\begin{pmatrix} (1+p_1)\cdot((1+\Delta p_1)\cdot x + \Delta p_3 \cdot y + \Delta p_5) + p_3 \cdot (\Delta p_2 \cdot x + (1+\Delta p_4)\cdot y + \Delta p_6) + p_5 \\ p_2 \cdot ((1+\Delta p_1)\cdot x + \Delta p_3 \cdot y + \Delta p_5) + (1+p_4)\cdot (\Delta p_2 \cdot x + (1+\Delta p_4)\cdot y + \Delta p_6) + p_6 \end{pmatrix}$$

Overview

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

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

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

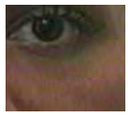


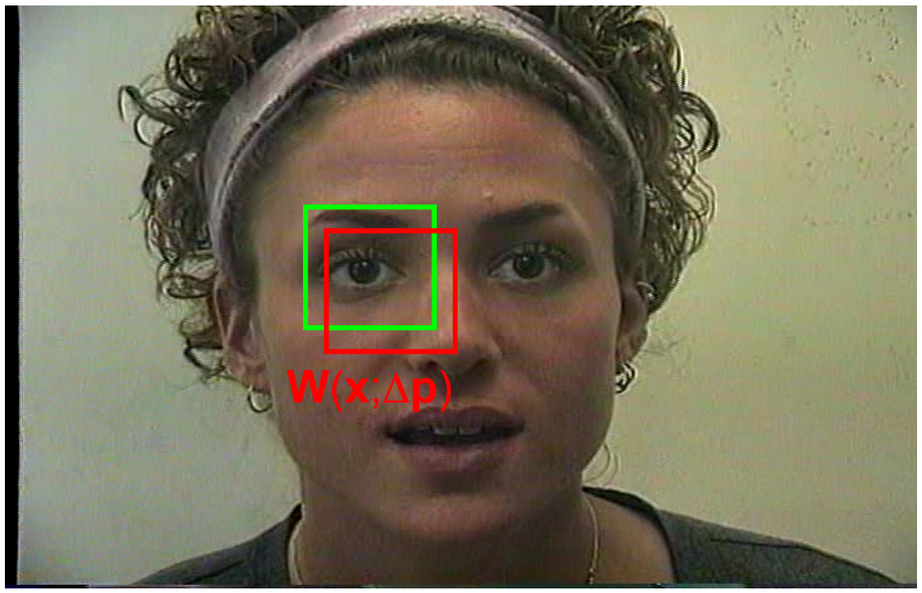
Key idea: compute update on template (not image). But apply *inverse* update to image. Because the template is never warped, it's Jacobian *and* approx Hessian (and its inverse!) can be precomputed.

Forward and Inverse Compositional

- Forwards compositional

$T(\mathbf{x})$

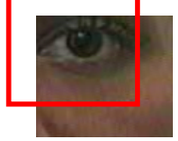




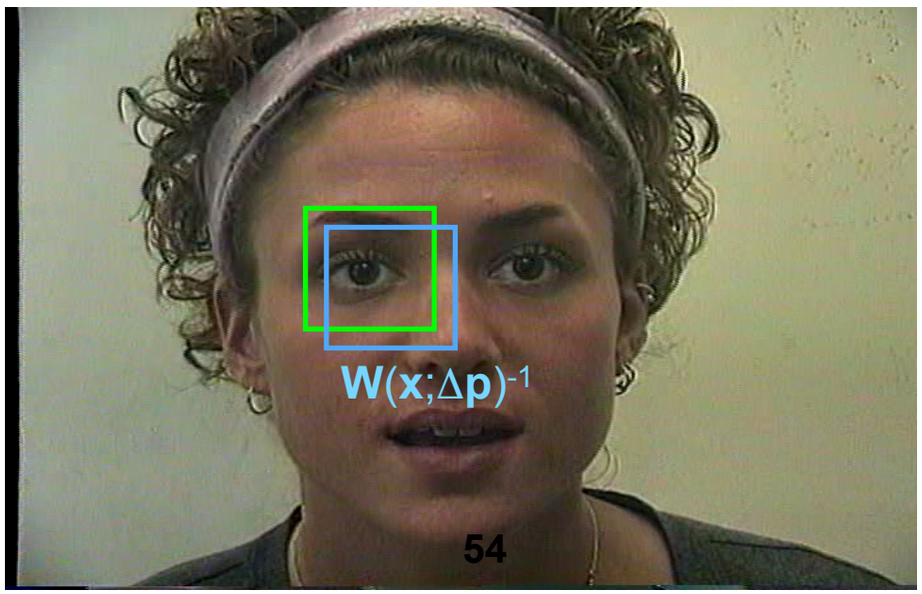
$I(\mathbf{x})$

- Inverse compositional

$T(\mathbf{x})$



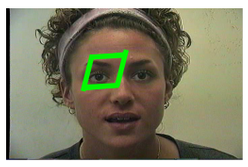
$\mathbf{W(x;\Delta p)}$



$I(\mathbf{x})$

Inverse Compositional

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



Taylor approx of template: $T(\mathbf{W}(\mathbf{x};\mathbf{\Delta p})) \approx T(\mathbf{W}(\mathbf{x};\mathbf{0})) + \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \mathbf{\Delta p}$

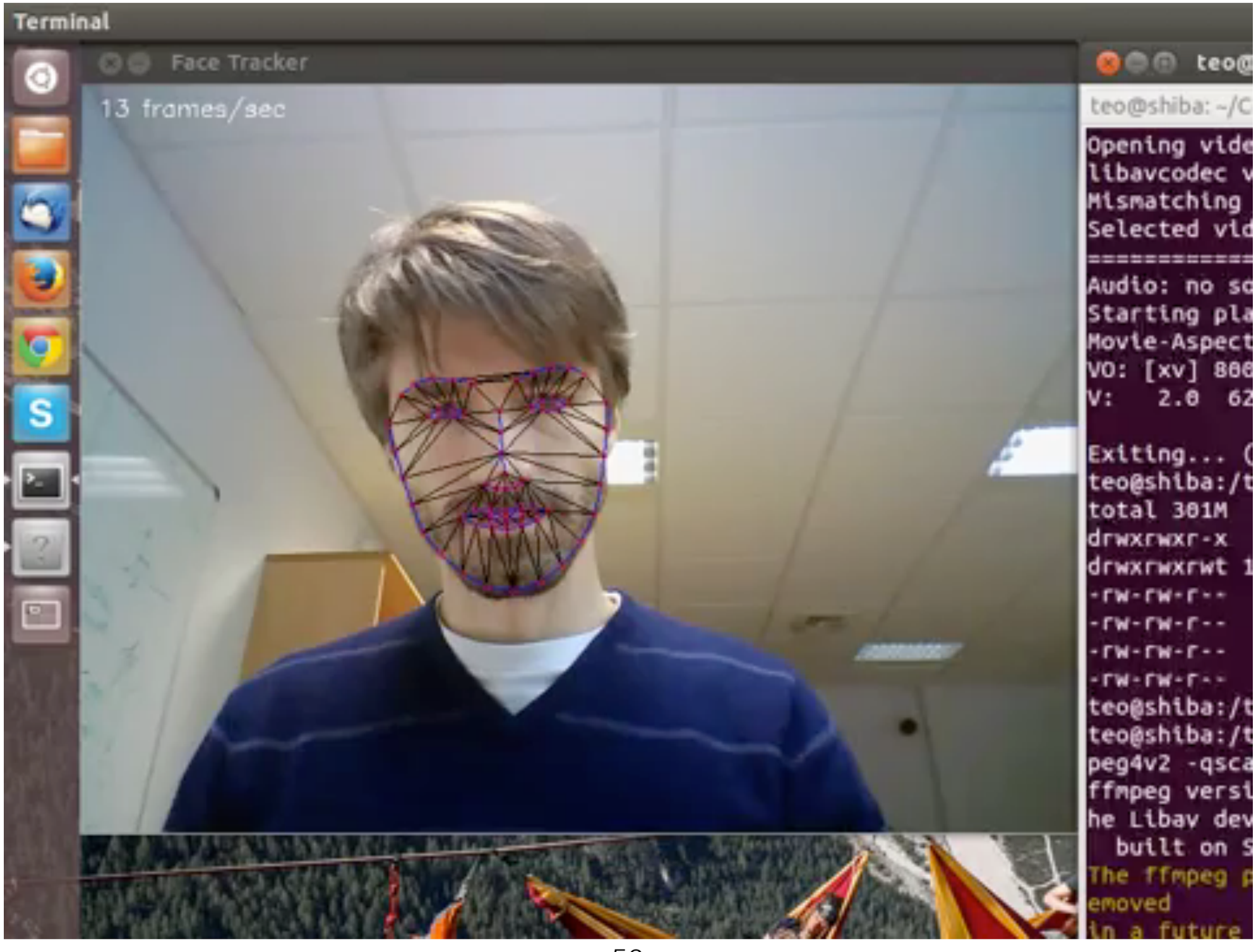
Soln: $\Delta\mathbf{p} = -\sum_{\mathbf{x}} H^{-1} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [\, T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\,]$

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

Precompute

Update: $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$

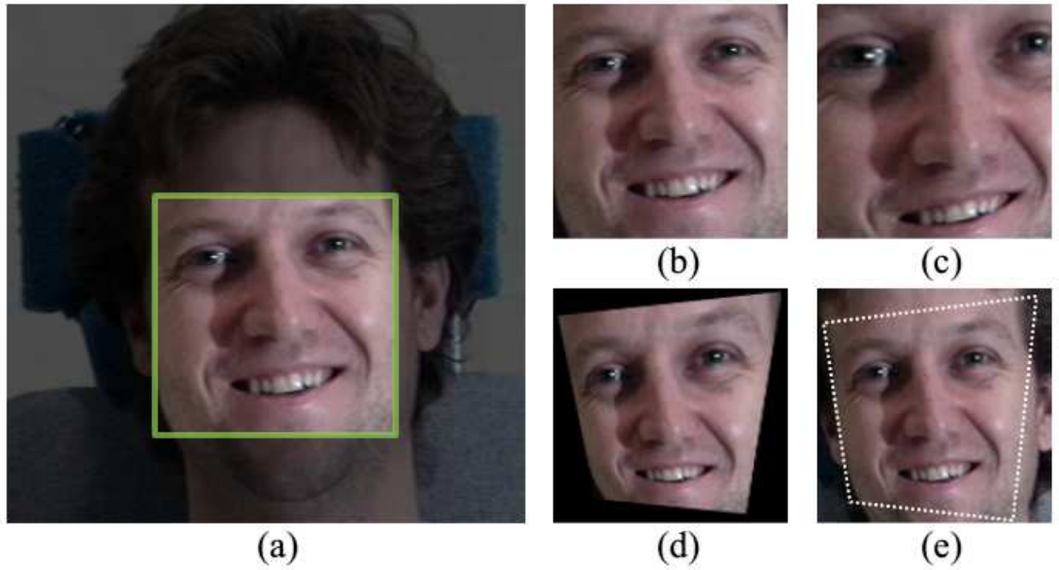
Peicewise affine-tracking

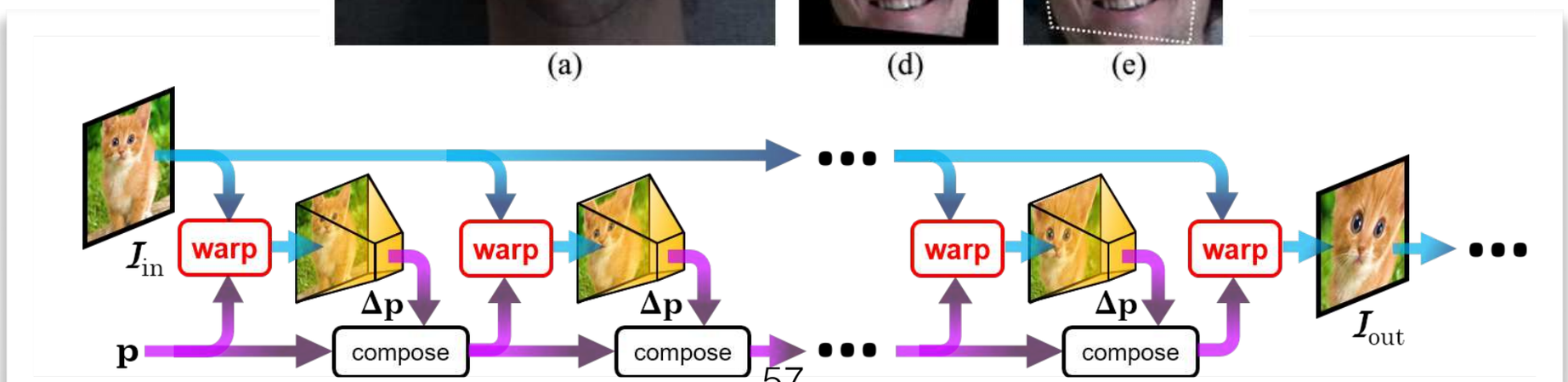


Contemporary extensions

Inverse Compositional Spatial Transformer Networks

Chen-Hsuan Lin Simon Lucey
The Robotics Institute
Carnegie Mellon University
chenhsul@andrew.cmu.edu slucey@cs.cmu.edu





A look back

- Geometric warps
  - Foreshortening-vs-projective
  - Peicewise affine
  - Fwd-vs-inverse warps
- Lucas Kanade
  - Gauss Newton optimiztion of nonlinear least squares
  - Inverse composition