Least squares and motion

Nuno Vasconcelos

ECE Department, UCSD

Plan for today

- today we will discuss motion estimation
- this is interesting in two ways
  - motion is very useful as a cue for recognition, segmentation, compression, etc.
  - is a great example of least squares problem
- we will also wrap up discussion on least squares
- introduce two types of motion estimation
  - block matching
  - differential methods
- will talk about motion ambiguities, and local vs global motion

Least squares

- a least squares problem is one where we have
  - two variables (X,Y) related by an unknown function $Y = g(X)$
  - a training set $D = \{(x_1,y_1), \ldots, (x_n,y_n)\}$
  - a model $Y = f(x;\Phi)$ where $\Phi = (\phi_1,\ldots,\phi_k)$ is a vector of parameters
- the goal is:
  - to find the model parameters that lead to the best approximation to the observed data, i.e. to determine

$$\varepsilon^* = \min_{\Phi} \sum_i \left[y_i - f(x_i,\Phi)\right]^2$$

  - the canonical example is the problem of fitting a line to a set of points
  - here $\Phi = (a,b)$, and $f(x;a,b) = ax + b$

Two main cases

- non-linear least squares
  - $f(x,\Phi)$ not linear on Φ, e.g.

$$f(x;\Phi) = \sum_k \sin(\phi_k x)$$

- linear least squares
  - $f(x,\Phi)$ linear on Φ, e.g.

$$f(x;\Phi) = \sum_k \phi_k \sin(x)$$

- note: all that matters is linearity on Φ, both nonlinear on x
- other linear models: polynomials, splines, neural networks, Fourier decompositions, etc.

Non-linear least squares

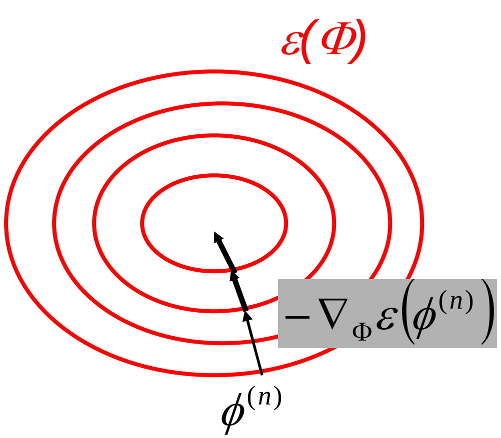
- most difficult case
- optimal solution if and only if:
  - gradient of ε is zero
  - Hessian of ε negative definite

$$\nabla_{\Phi}\varepsilon = \begin{bmatrix} \partial\varepsilon / \partial\phi_0 \\ \vdots \\ \partial\varepsilon / \partial\phi_k \end{bmatrix} = 0$$

$$\nabla^2_{\Phi}\varepsilon = \begin{bmatrix} \partial^2\varepsilon / \partial\phi_0^2 & \ldots & \partial^2\varepsilon / \partial\phi_0\partial\phi_k \\ & \vdots & \\ \partial^2\varepsilon / \partial\phi_k\partial\phi_0 & \ldots & \partial^2\varepsilon / \partial\phi_k^2 \end{bmatrix}$$

$$\forall z, \quad z^T\left(\nabla^2_{\Phi}\varepsilon\right)z < 0$$

- in general this has no closed form
- numerical solution, e.g. gradient descent
  - pick initial estimate $\Phi^{(0)}$
  - iterate $\Phi^{(n+1)} = \Phi^{(n)} - \alpha\nabla_{\Phi}\varepsilon\left(\Phi^{(n)}\right)$



Linear least squares

- closed form solution
  - write

$$\begin{bmatrix} f(x_1,\Phi) \\ \vdots \\ f(x_n,\Phi) \end{bmatrix} = \Gamma(x_1,\cdots,x_n)\Phi, \qquad y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

  - solution is given by normal equations

$$\Phi = \left(\Gamma^T\Gamma\right)^{-1}\Gamma^T y$$

- e.g. for a line $f(x;\phi_0,\phi_1) = \phi_0 + \phi_1 x$

$$\Gamma(x_1,\cdots,x_n) = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \qquad \begin{bmatrix} \phi_0 \\ \phi_1 \end{bmatrix} = \begin{bmatrix} \sum_i 1 & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_i y_i \\ \sum_i y_i x_i \end{bmatrix}$$

Very powerful

- Q: what is the best linear approximation of a N point sequence by M DFT style exponentials?

$$x_n = \sum_{k=1}^{M} \phi_k e^{j\frac{2\pi}{N}kn}$$

- to get least squares solution, we need *Γ(1,…,N)*

$$\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} \sum_{k=0}^{M-1} \phi_k e^{j\frac{2\pi 0}{N}k} \\ \sum_{k=0}^{M-1} \phi_k e^{j\frac{2\pi}{N}k} \\ \vdots \\ \sum_{k=0}^{M-1} \phi_k e^{j\frac{2\pi(N-1)}{N}k} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ e^{j\frac{2\pi}{N}} & e^{j\frac{4\pi}{N}} & \cdots & e^{j\frac{2(M-1)\pi}{N}} \\ & & \vdots & \\ e^{j\frac{2\pi(N-1)}{N}} & e^{j\frac{4\pi(N-1)}{N}} & \cdots & e^{j\frac{2(M-1)\pi(N-1)}{N}} \end{bmatrix} \begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_{M-1} \end{bmatrix}$$

Best Fourier approximation

- this means that

$$\Gamma = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ e^{j\frac{2\pi}{N}} & e^{j\frac{4\pi}{N}} & \cdots & e^{j\frac{2(M-1)\pi}{N}} \\ & & \vdots & \\ e^{j\frac{2\pi(N-1)}{N}} & e^{j\frac{4\pi(N-1)}{N}} & \cdots & e^{j\frac{2(M-1)\pi(N-1)}{N}} \end{bmatrix}$$

- this is orthonormal, i.e. $\Gamma^T\Gamma = I$, and

$$\Phi = \left(\Gamma^T\Gamma\right)^{-1}\Gamma^T x = \Gamma^T x \quad \Leftrightarrow \quad \phi_k = \sum_{n=0}^{N-1} x_n e^{j\frac{2\pi}{N}kn}, \qquad k = 0,\ldots,M\text{-}1$$

i.e. the best approximation are the *M* DFT coefficients associated with the exponentials

Signal approximation

- Q: what is the band-pass filter h(n) whose output y(n) best approximates a signal x(n) in the frequency range Ω?
- we have seen that y(n) must have DFT

$$Y(k) = \begin{cases} X(k), & k \in \Omega \\ 0, & otherwise \end{cases}$$

- hence optimal filter has DFT

$$H(k) = \begin{cases} 1, & k \in \Omega \\ 0, & otherwise \end{cases}$$

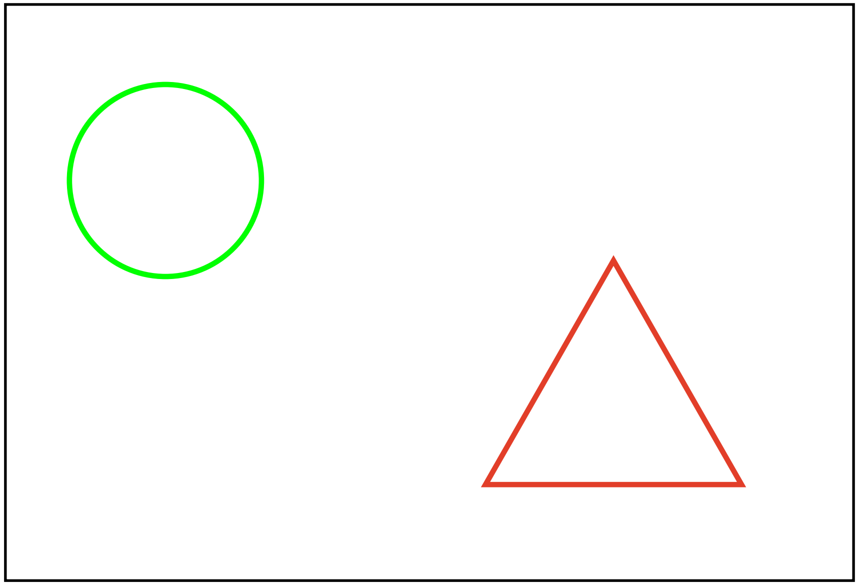
- i.e. it is the ideal band-pass filter of band Ω
- intuitive: ideal = best approximation in LS sense!

Motion estimation

- is an important practical example of LS problems
- many applications:
  - recognition: many events are characterized by the type of motion (e.g. walking vs running)
  - strong clues about scene structure (e.g. when we rotate a 3D object, motion of a pixel determined by how far the 3D point is from camera)
  - segmentation (things that move “together” belong to the same object)
  - alignment (once we know the motion we can align images in a sequence, e.g. the NASA panoramas)
  - compression (estimate motion, align images, transmit only error)
  - etc

Motion estimation

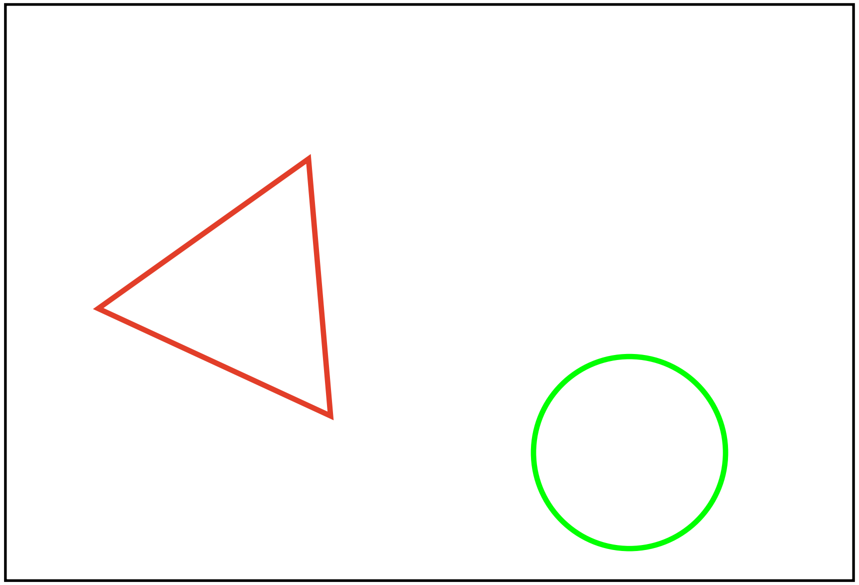
- consider the following two images



time t

Motion estimation

- consider the following two images



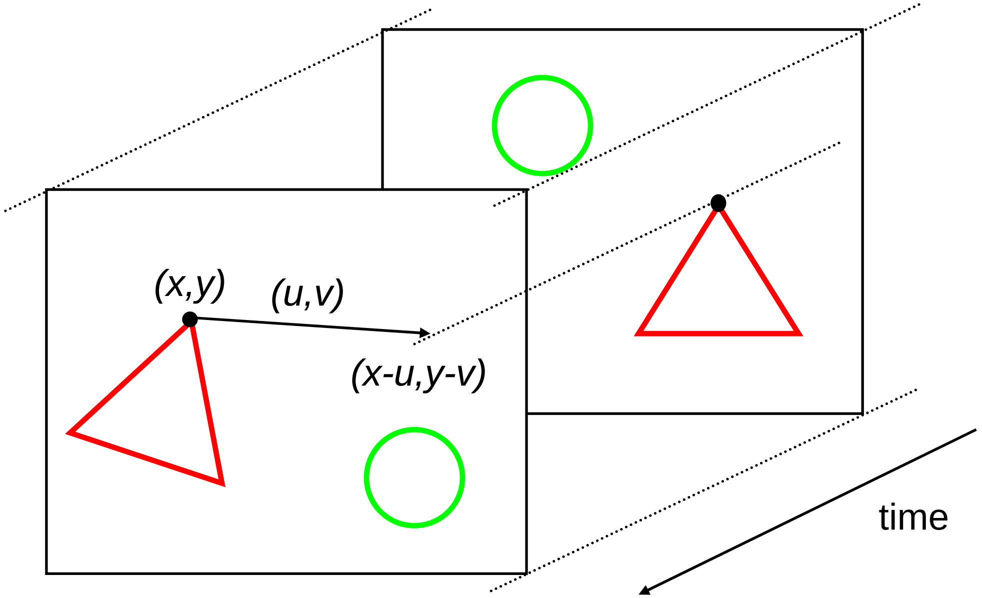
time t+1

Motion estimation

- goal: given images *I(x,y,t)* and *I(x,y,t+1)*, for each pixel find *(u,v)* which minimizes difference

$$D(x,y) = \left[I(x-u, y-v, t) - I(x, y, t+1) \right]^2$$

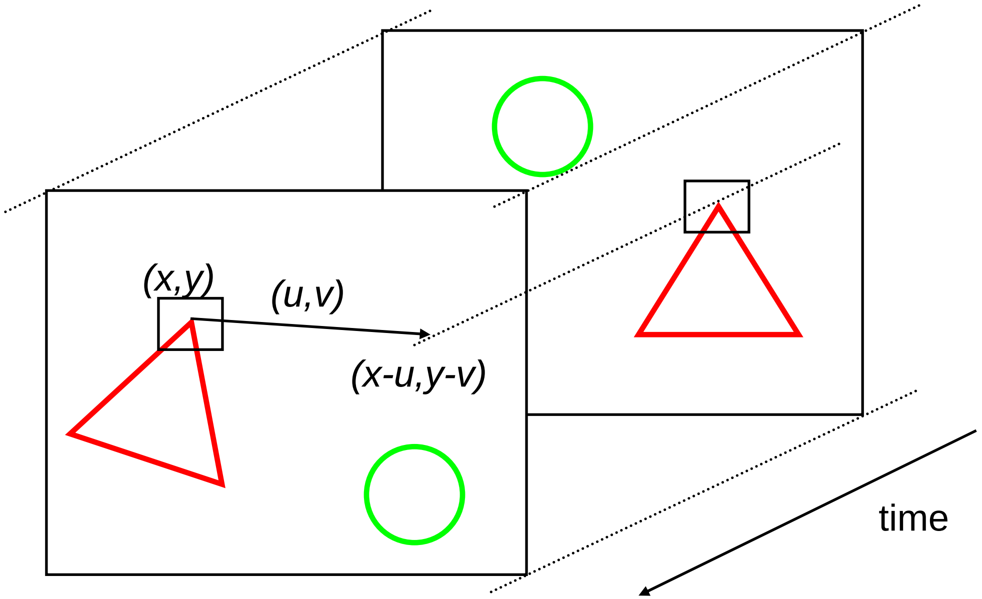
- problem: impossible to solve from one pixel alone
  - two unknowns *(u,v)*, one equation



Fundamental law

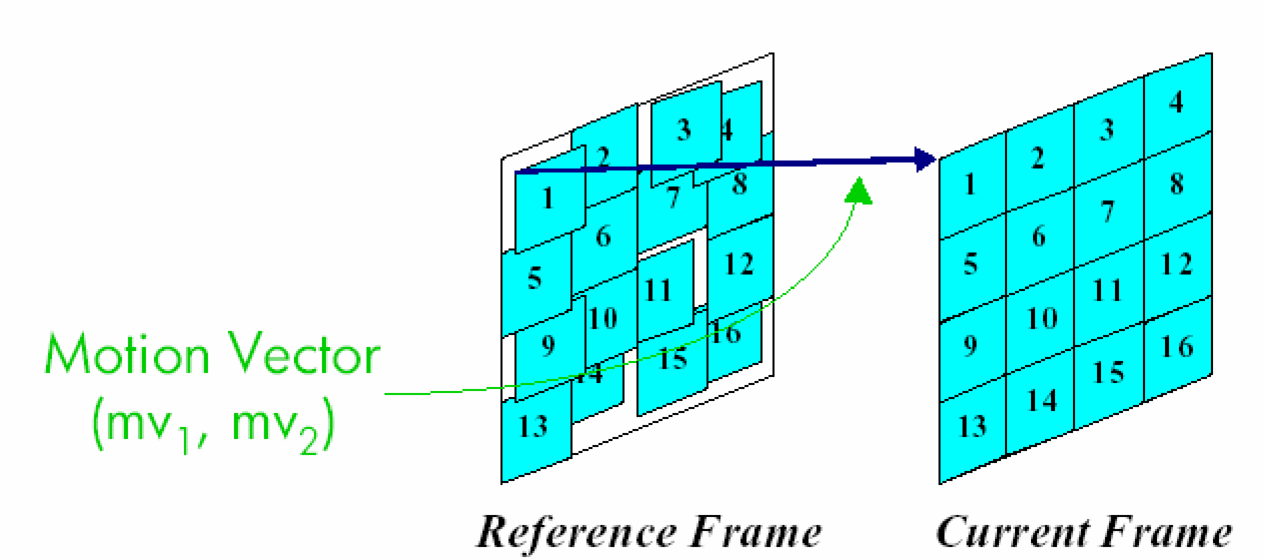
- motion can only be solved over a neighborhood
  - need at least two pixels
  - makes sense to consider more and minimize the average error
- this is least squares

$$\varepsilon = \sum_{x,y \in R} \left[I(x-u, y-v, t) - I(x, y, t+1) \right]^2$$



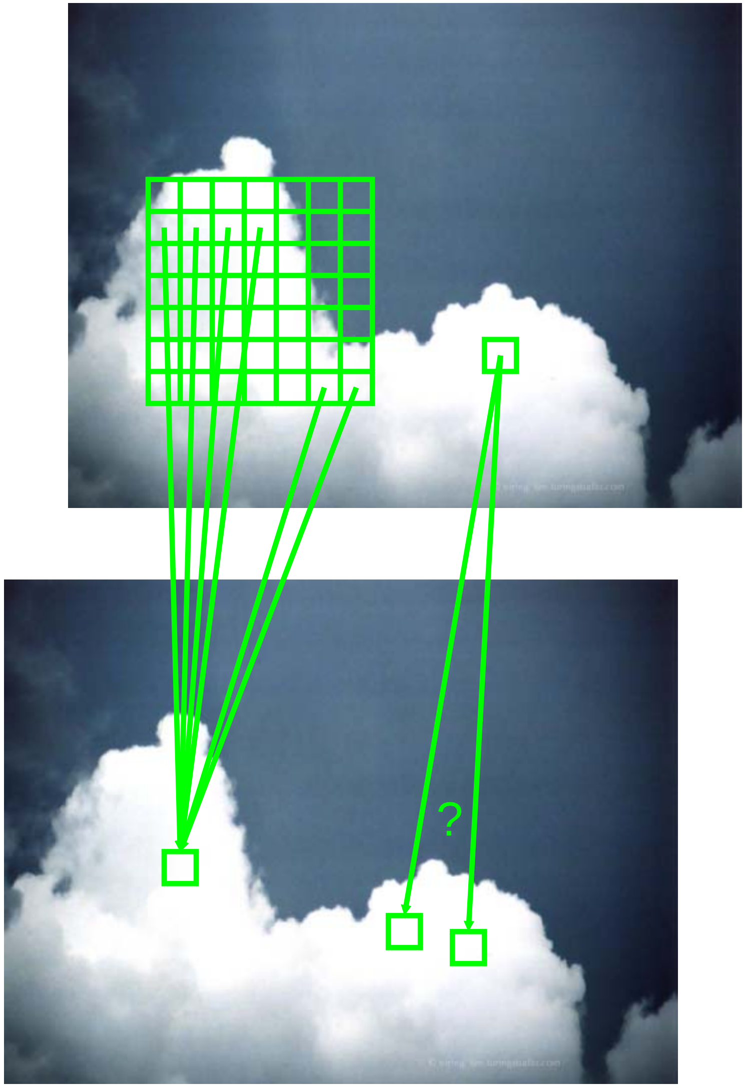
Block matching

- in fact, it is a non-linear least squares problem, since *I(x-u,y-v,t)* is a non-linear function of *(u,v)*
- solution I: block matching
  - for each block in *I(x,y,t+1)* do an exhaustive search in *I(x,y,t)* for the closest match
  - very common in compression, e.g. MPEG



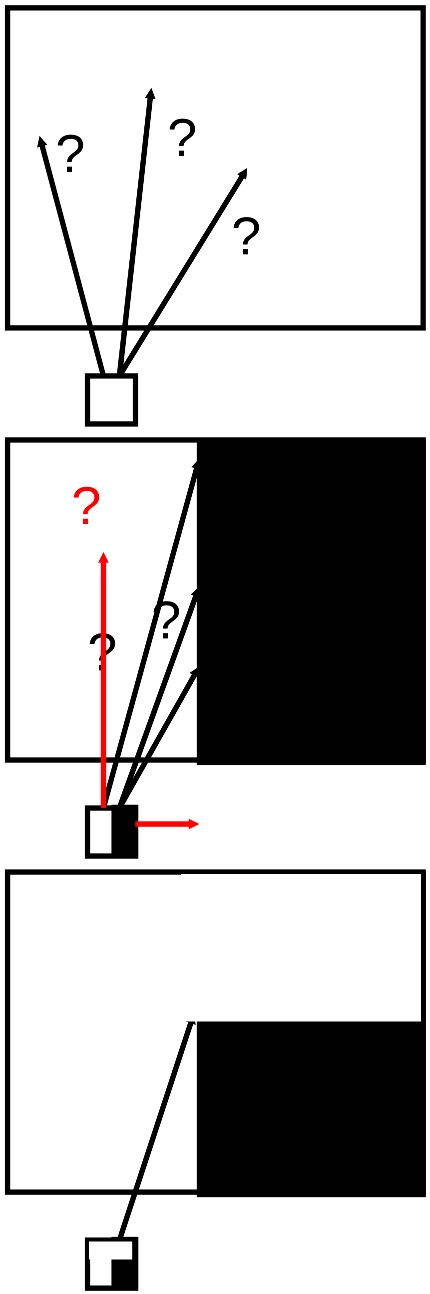
Block matching

- is computationally intensive
  - need to compute the squared error between the block and a collection of blocks in the previous image
- does not always produce good motion estimates
  - e.g. many matches can be equally good
- this is a problem for all motion estimation methods:
  - motion can be ambiguous when measured locally (e.g. by matching windows)



Motion ambiguities

- clearly we cannot determine the motion of a flat neighborhood
- for an edge neighborhood, we can only determine one of the two components
- the two components are uniquely defined only when the neighborhood contains 2D image structure
- this is called the "aperture problem"



Differential methods

- we can at least eliminate the complexity problem, by looking for a closed-form solution to

$$\varepsilon^* = \min_{d_x, d_y} \sum_{x,y \in R} \left[I(x-u, y-v, t) - I(x, y, t+1) \right]^2$$

- problem: this is a non-linear function of *(u,v)*
- solution: clearly, the problem is due to

$$\Delta = I\left(x, y, t+1\right) - I\left(x-u, y-v, t\right)$$

- this equation can be made linear on *(u,v)* by a Taylor series approximation

$$I\left(x-u, y-v, t\right) = I(x, y, t) - u\frac{\partial I(x, y, t)}{\partial x} - v\frac{\partial I(x, y, t)}{\partial y}$$

Differential methods

- which leads to

$$\Delta = \underbrace{I\left(x,y,t+1\right) - I\left(x,y,t\right)}_{A} + \underbrace{u\frac{\partial I\left(x,y,t\right)}{\partial x} + v\frac{\partial I\left(x,y,t\right)}{\partial y}}_{B}$$

- note: we know how to compute these terms
  - A is the difference between consecutive frames

$$A = I_t\left(x,y,t\right) = I\left(x,y,t+1\right) - I\left(x,y,t\right)$$

  - B is

$$B = \nabla I\left(x,y,t\right)^T \begin{bmatrix} u \\ v \end{bmatrix}$$

  i.e. a function of the image gradient

$$\nabla I\left(x,y,t\right) = \left(I_x\left(x,y,t\right), I_y\left(x,y,t\right)\right)^T$$
$$= \left(\frac{\partial I\left(x,y,t\right)}{\partial x}, \frac{\partial I\left(x,y,t\right)}{\partial y}\right)^T$$

Differential methods

- we thus have

$$\Delta = I_t(x,y,t) + uI_x(x,y,t) + vI_y(x,y,t)$$

- and the least squares problem is

$$\varepsilon^* = \sum_{x,y \in R} \left[I_t(x,y) + uI_x(x,y) + vI_y(x,y) \right]^2$$

(note: since t is constant, we omit it)

- this is now linear least squares, we can just use our formula
- recall that

Linear least squares

- if

$$\varepsilon^* = \min_{\Phi} \sum_i \left[y_i - f(x_i, \Phi) \right]^2$$

- then the LS solution is:
  - write

$$\begin{bmatrix} f(x_1, \Phi) \\ \vdots \\ f(x_n, \Phi) \end{bmatrix} = \Gamma(x_1, \cdots, x_n)\Phi, \qquad y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

  - solution is given by normal equations

$$\Phi = \left(\Gamma^T \Gamma\right)^{-1} \Gamma^T y$$

Least squares solution

- for motion, instead of

$$\varepsilon^* = \sum_i \left[y_i - f(x_i, \Phi) \right]^2$$

- we have

$$\varepsilon^* = \sum_{x,y \in R} \left[I_t(x,y) + uI_x(x,y) + vI_y(x,y) \right]^2$$

- and write

$$\begin{bmatrix} f(x_1, y_1, \Phi) \\ \vdots \\ f(x_n, y_n, \Phi) \end{bmatrix} = -\begin{bmatrix} I_x(x_1, y_1) & & I_y(x_1, y_1) \\ & \vdots & \\ I_x(x_n, y_n) & & I_y(x_n, y_n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}, \quad y = \begin{bmatrix} I_t(x_1, y_1) \\ \vdots \\ I_t(x_n, y_n) \end{bmatrix}$$

Least squares solution

- the normal equations are

$$\begin{bmatrix} u \\ v \end{bmatrix} = -\left(\begin{bmatrix} I_x(x_1,y_1) & \cdots & I_x(x_n,y_n) \\ I_y(x_1,y_1) & \cdots & I_y(x_n,y_n) \end{bmatrix} \begin{bmatrix} I_x(x_1,y_1) & & I_y(x_1,y_1) \\ & \vdots & \\ I_x(x_n,y_n) & & I_y(x_n,y_n) \end{bmatrix} \right)^{-1} \times$$

$$\times \begin{bmatrix} I_x(x_1,y_1) & \cdots & I_x(x_n,y_n) \\ I_y(x_1,y_1) & \cdots & I_y(x_n,y_n) \end{bmatrix} \begin{bmatrix} I_t(x_1,y_1) \\ \vdots \\ I_t(x_n,y_n) \end{bmatrix}$$

- leading to the solution

$$\begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum_{x,y \in R} I_x^2(x,y) & \sum_{x,y \in R} I_x(x,y) I_y(x,y) \\ \sum_{x,y \in R} I_x(x,y) I_y(x,y) & \sum_{x,y \in R} I_y^2(x,y) \end{bmatrix}^{-1} \begin{bmatrix} \sum_{x,y \in R} I_x(x,y) I_t(x,y) \\ \sum_{x,y \in R} I_y(x,y) I_t(x,y) \end{bmatrix}$$

Least squares solution

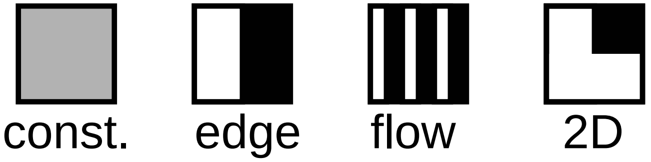
- when is this well defined?
- note that

$$\begin{bmatrix} \sum\limits_{x,y\in R} I_x^2(x,y) & \sum\limits_{x,y\in R} I_x(x,y)I_y(x,y) \\ \sum\limits_{x,y\in R} I_x(x,y)I_y(x,y) & \sum\limits_{x,y\in R} I_y^2(x,y) \end{bmatrix}$$

- has to be invertible
- it turns out that this is a function of the image structure within the window R

Orientation representations

- more general question: what sorts of structure are there?
- It is common to describe image patches by the variation of the gradient orientation



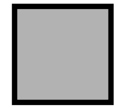
- important types:
  - constant window
    - small gradient mags
  - edge window
    - few large gradient mags in one direction
  - flow window
    - many large gradient mags in one direction (e.g. hair)
  - corner window
    - large gradient mags that swing (e.g. corner)

Representing Windows

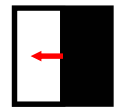
- how can we **detect** these types of windows?
- the key is the matrix

$$H = \sum_{window} (\nabla I)(\nabla I)^T$$

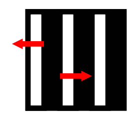
- how does it relate to edges?
- the answer is in the **rank**



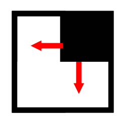
$$H = \sum 00^T = 0$$



$$H = \sum_{edge\ pts} \begin{pmatrix} -k \\ 0 \end{pmatrix} \begin{pmatrix} -k & 0 \end{pmatrix} = \begin{pmatrix} nk^2 & 0 \\ 0 & 0 \end{pmatrix}$$



$$H = \#\{edges\}\ x \begin{pmatrix} nk^2 & 0 \\ 0 & 0 \end{pmatrix}$$



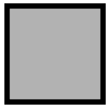
$$H = \sum_{vert\ edge} \begin{pmatrix} -k \\ 0 \end{pmatrix} \begin{pmatrix} -k & 0 \end{pmatrix} + \sum_{horiz\ edge} \begin{pmatrix} 0 \\ -k \end{pmatrix} \begin{pmatrix} 0 & -k \end{pmatrix}$$

$$= \begin{pmatrix} \frac{nk^2}{2} & 0 \\ 0 & \frac{nk^2}{2} \end{pmatrix}$$

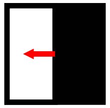
Representing Windows

- recall: the eigenvalues of a diagonal matrix are the diagonal entries
- hence:
  - constant window
    - small eigenvalues
  - edge window
    - one medium, one small
  - flow window
    - one large, one small
  - corner window
    - two large eigenvalues

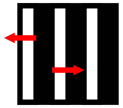
$$H = \sum_{window} (\nabla I)(\nabla I)^T$$



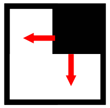
$$H = 0$$



$$H \propto \begin{pmatrix} k^2 & 0 \\ 0 & 0 \end{pmatrix}$$



$$H \propto \begin{pmatrix} k^2 & 0 \\ 0 & 0 \end{pmatrix}$$



$$H \propto \begin{pmatrix} k^2 & 0 \\ 0 & k^2 \end{pmatrix}$$

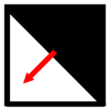
Representing Windows

- what about other orientations?
- useful property
  - if A is a 2 x 2 matrix
  - then

$$\boxed{\begin{aligned}\lambda_1\lambda_2 &= \det(A) \\ \lambda_1 + \lambda_2 &= a_{11} + a_{22} = trace(A)\end{aligned}}$$

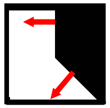
- to have full rank we need diversity in the component matrices
- i.e. need edges of different orientation

$$H = \sum_{window} (\nabla I)(\nabla I)^T$$



$$H = \sum_{edge} \begin{pmatrix} -a \\ -b \end{pmatrix} \begin{pmatrix} -a & -b \end{pmatrix} = n \begin{pmatrix} a^2 & ab \\ ab & b^2 \end{pmatrix}$$

$$\lambda_1 = n(a^2 + b^2);\ \lambda_2 = 0$$



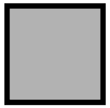
$$H = n_1 \begin{pmatrix} a^2 & ab \\ ab & b^2 \end{pmatrix} + n_2 \begin{pmatrix} c^2 & 0 \\ 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} n_1 a^2 + n_2 c^2 & ab \\ ab & b^2 \end{pmatrix}$$

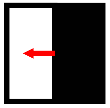
$$\lambda_1, \lambda_2 > 0$$

Representing Windows

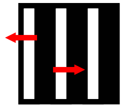
- in summary:
  - constant window
    - small eigenvalues
  - edge window
    - one medium, one small
  - flow window
    - one large, one small
  - corner window
    - two large eigenvalues



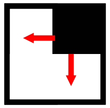
$$H = 0$$



$$H \propto \begin{pmatrix} k^2 & 0 \\ 0 & 0 \end{pmatrix}$$



$$H \propto \begin{pmatrix} k^2 & 0 \\ 0 & 0 \end{pmatrix}$$



$$H \propto \begin{pmatrix} k^2 & 0 \\ 0 & k^2 \end{pmatrix}$$

- this confirms what we had already seen:
  - motion can only be computed unambiguously when the neighborhood contains 2D information (e.g. corners)

In summary

- $[U,V]$ = lsme(I,I',w)
  - compute gradients $I_x, I_y, I_t = I' - I$
  - for each pixel (x,y)
    - let window $R = \{(x_i, y_i) \mid x - w \le x_i \le x + w,\ y - w \le y_i \le y + w\}$
    - compute

$$\begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum_{x,y\in R} I_x^2(x,y) & \sum_{x,y\in R} I_x(x,y) I_y(x,y) \\ \sum_{x,y\in R} I_x(x,y) I_y(x,y) & \sum_{x,y\in R} I_y^2(x,y) \end{bmatrix}^{-1} \begin{bmatrix} \sum_{x,y\in R} I_x(x,y) I_t(x,y) \\ \sum_{x,y\in R} I_y(x,y) I_t(x,y) \end{bmatrix}$$

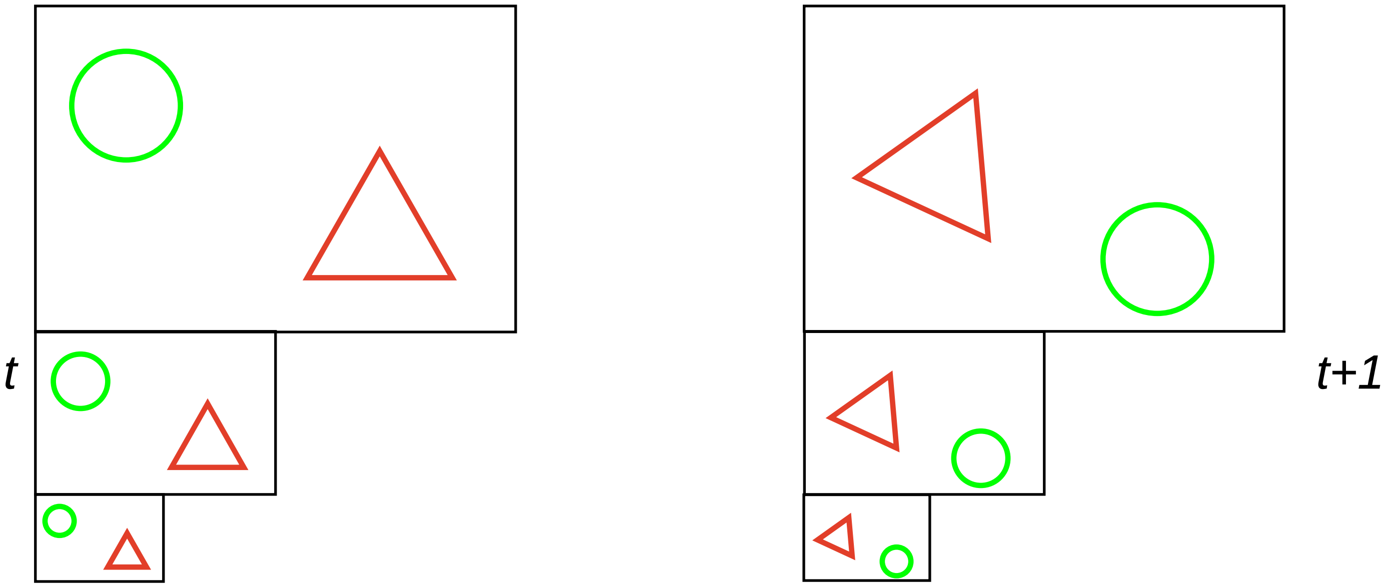
    - make $U(x,y) = u$, $V(x,y) = v$
  - return U,V

Problems

- recall we used the Taylor series approximation

$$I(x-u, y-v, t) = I(x,y,t) - u\frac{\partial I(x,y,t)}{\partial x} - v\frac{\partial I(x,y,t)}{\partial y}$$

- this is a good approximation only for small *(u,v)*
- to avoid this problem we need to use pyramids

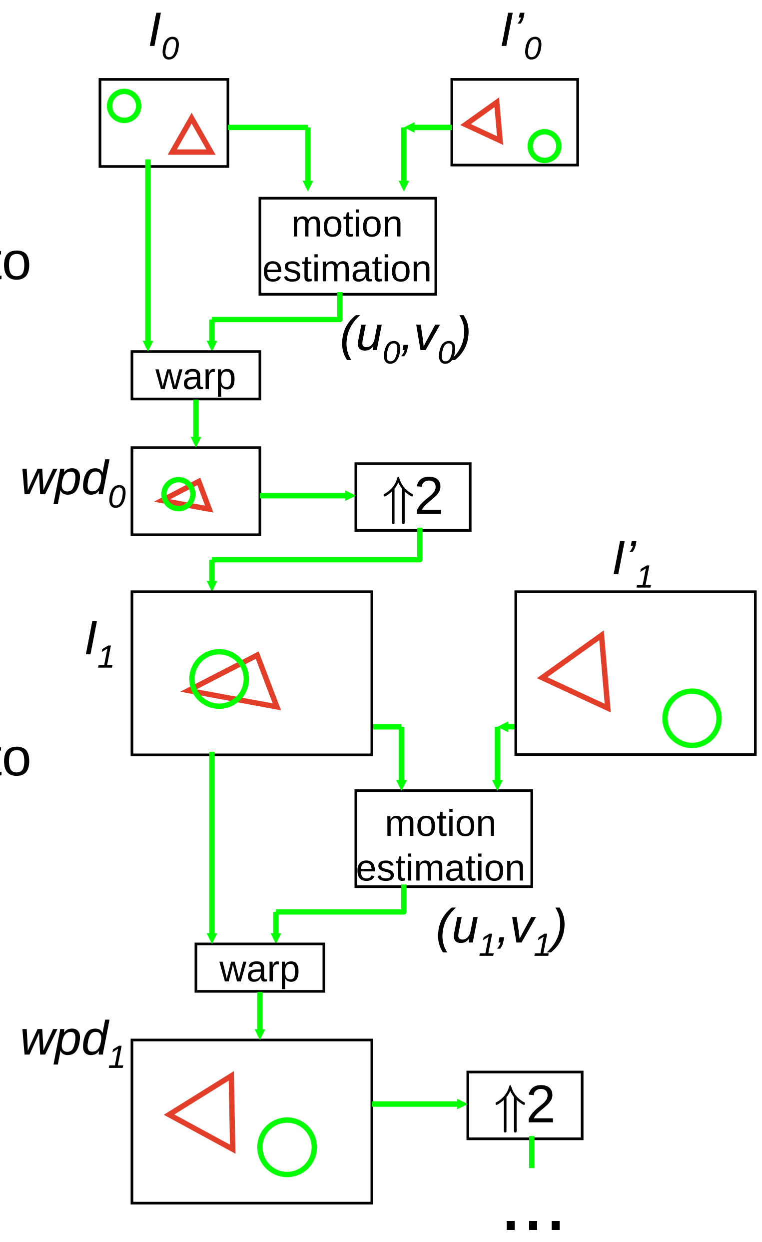


Hierarchical estimation

- algorithm:
  - do motion estimation using I_0 and I'_0 to obtain (u_0,v_0)
  - warp I_0 with (u_0,v_0) :

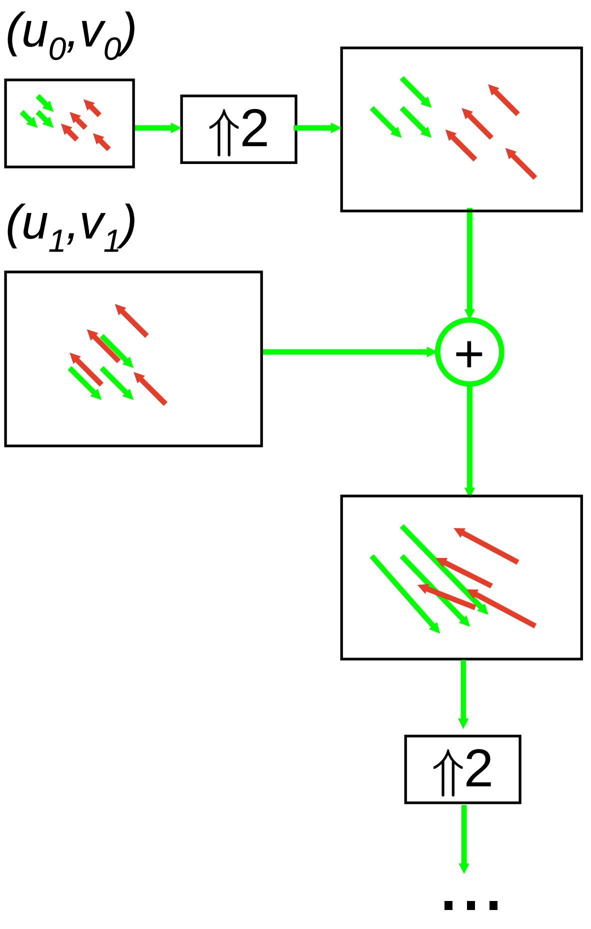
    $$wpd_0(x,y) = I_0(x-u_0,y-v_0)$$

  - up-sample by 2 to get I_1
  - do motion estimation using I_1 and I'_1 to obtain (u_1,v_1)
  - warp I_1 with (u_1,v_1)
  - etc.



Hierarchical estimation

- each stage improves the match
- solution:
  - upsample all *(u_i,v_i)* to full resolution
  - add to obtain *(u,v)*
- note that
  - small displacements at low resolution
  - are large displacements at full resolution
- combines linearity with ability to estimate large displacements

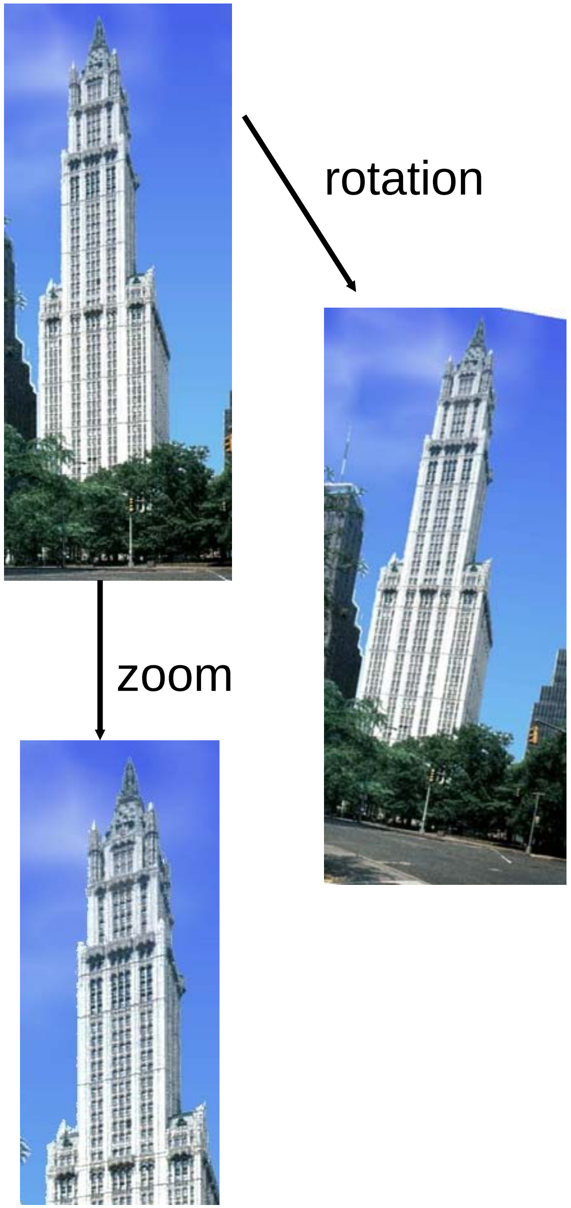


Motion models

- so far we have dealt
  - local motion (each pixel moves by itself)
  - translation

$$I(x, y, t+1) = I(x-u, y-v, t)$$

- local motion is the most generic (e.g. tree leaves blowing in the wind)
- one important alternative case is that of global motion
  - motion of all pixels satisfies one common equation
  - usually due to camera motion: panning, rotation, zooming



Important cases

- point *(x,y)* at time t warped into point *(x',y')* at time *t+1*
- important global motions are
  - translation by *(u,v)*

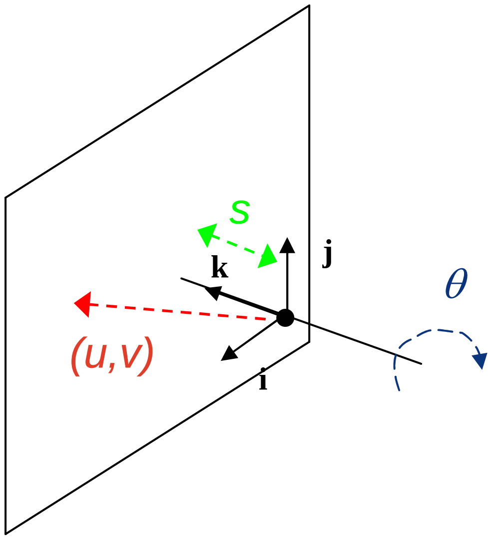
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} u \\ v \end{bmatrix}$$

  - rotation by θ

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

  - scaling by (s_x, s_y)

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

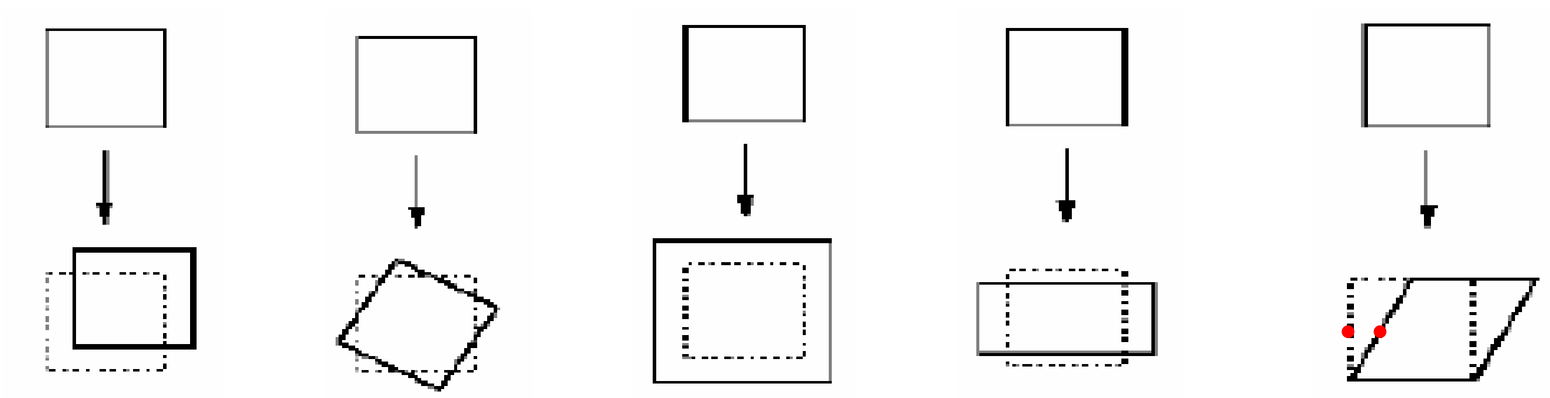


Affine transformations

- these are all special cases of the affine transformation

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

- motion of entire image described by $\Phi = (a,b,c,d,e,f)^T$
- can account for translation, rotation, scaling, and shear



translation rotation uniform scale nonuniform scale shearing

Any questions?