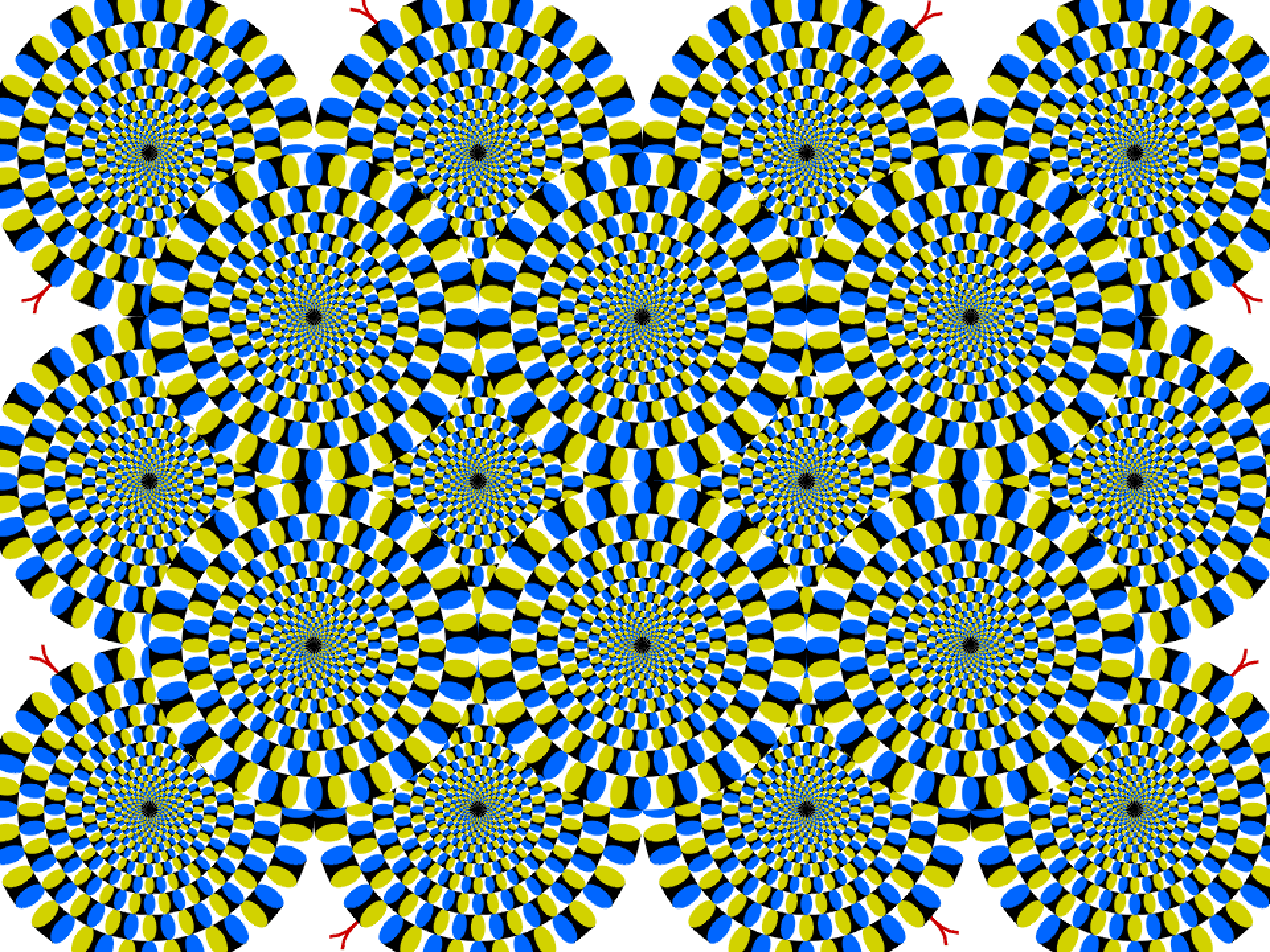


INF4300
Motion

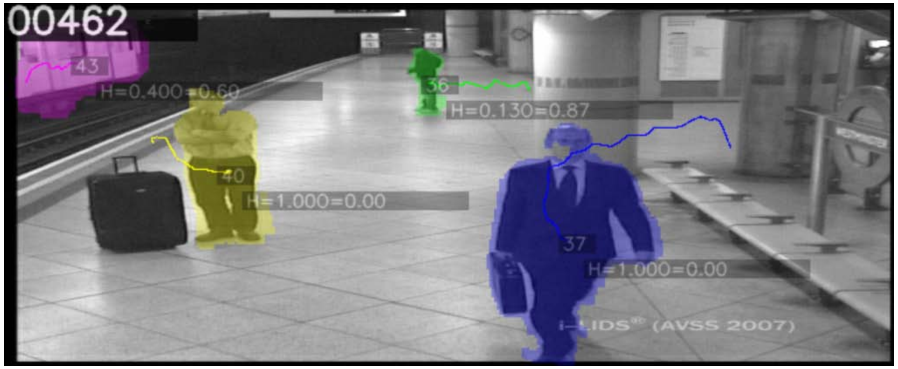
Asbjørn Berge 10-11-2010

Stuff to read or experiment with after this lecture

- OpenCV implementation lkdemo.cpp
  - http://opencv.willowgarage.com/wiki/
- Lucas Kanade affine template tracking
  - http://www.mathworks.com/matlabcentral/fileexchange/24677
- Jianbo Shi and Carlo Tomasi *Good Features to Track*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94), 1994, pp. 593 - 600.
- Simon Baker and Iain Matthews, Lucas-Kanade 20 Years On: A Unifying Framework, International Journal of Computer Vision 56(3), 221-255, 2004 http://dx.doi.org/10.1023/B:VISI.0000011205.11775.fd

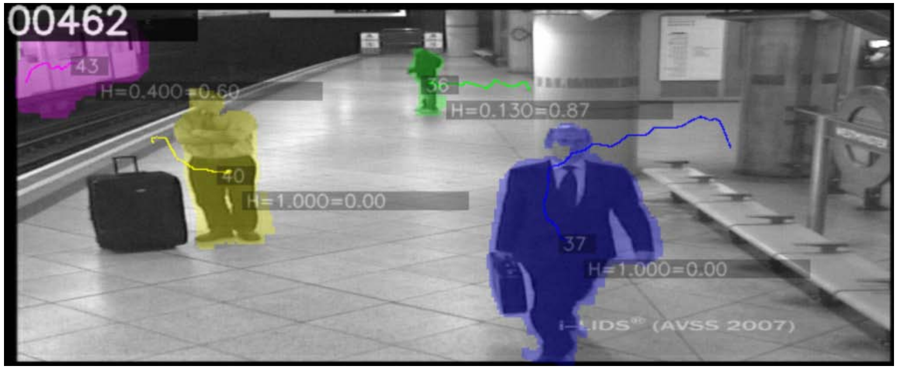
Applications of segmentation to video

- Background subtraction
  - A static camera is observing a scene
  - Goal: separate the static *background* from the moving *foreground*



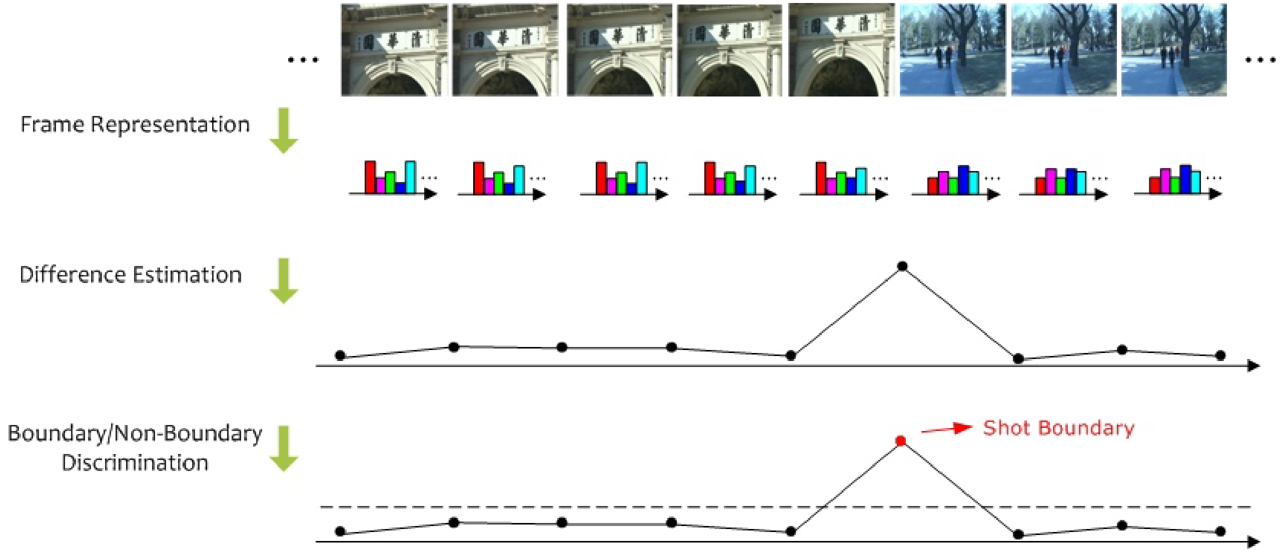
Applications of segmentation to video

- Background subtraction
  - Form an initial background estimate
  - For each frame:
    - Update estimate using a moving average
    - Subtract the background estimate from the frame
    - Label as foreground each pixel where the magnitude of the difference is greater than some threshold
    - Use median filtering, morphology or to “clean up” the results



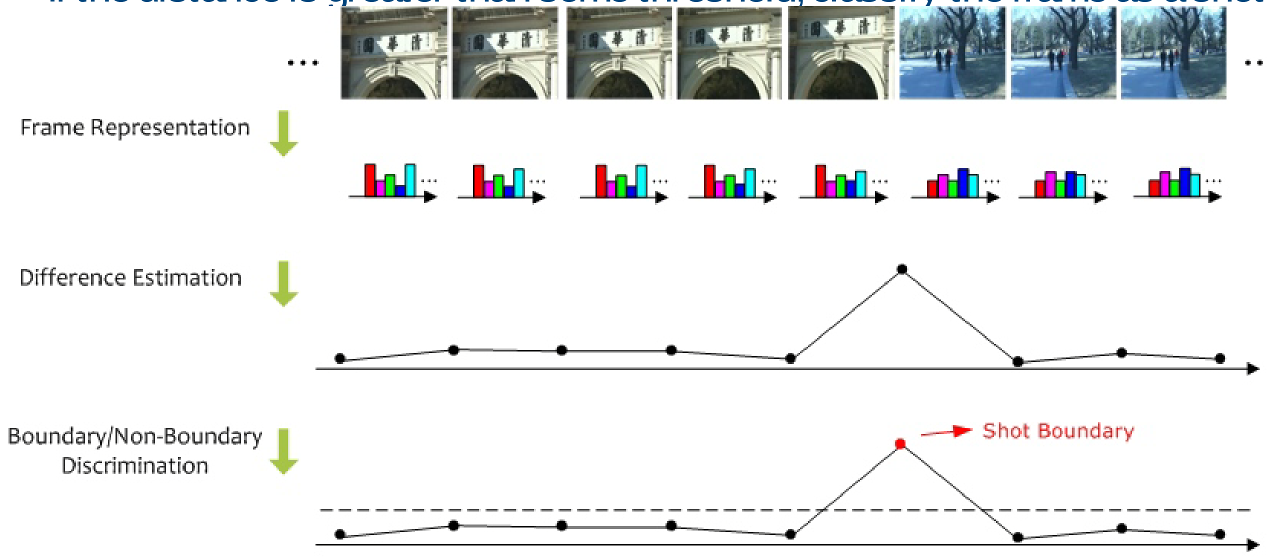
Applications of segmentation to video

- Shot boundary detection
  - Commercial video is usually composed of *shots* or sequences showing the same objects or scene
  - Goal: segment video into shots for summarization and browsing (each shot can be represented by a single keyframe in a user interface)
  - Difference from background subtraction: the camera is not necessarily stationary



Applications of segmentation to video

- Shot boundary detection
  - For each frame
    - Compute the distance between the current frame and the previous one
      - Pixel-by-pixel differences
      - Differences of color histograms
      - Block comparison
    - If the distance is greater than some threshold, classify the frame as a shot boundary

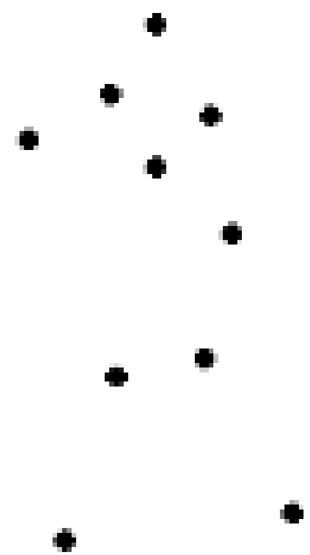


Applications of segmentation to video

- Motion segmentation
  - Segment the video into multiple *coherently* moving objects

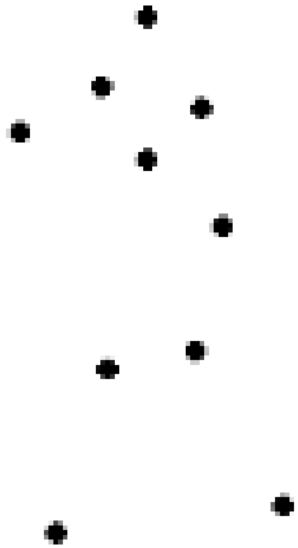
Motion is a strong feature

- Even "impoverished" motion data can evoke a strong percept



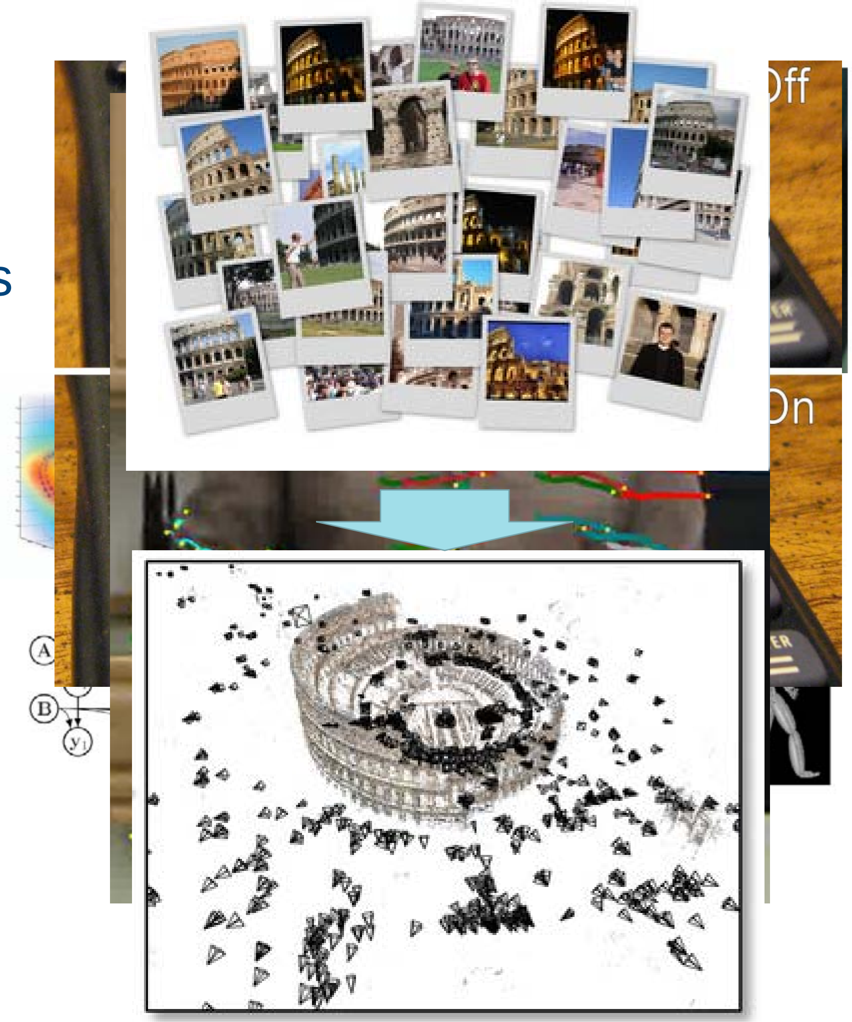
Motion is a strong feature

- Even “impoverished” motion data can evoke a strong percept



Uses of motion

- Estimating 3D structure
- Segmenting objects based on motion cues
- Learning dynamical models
- Recognizing events and activities
- Improving video quality (motion stabilization)



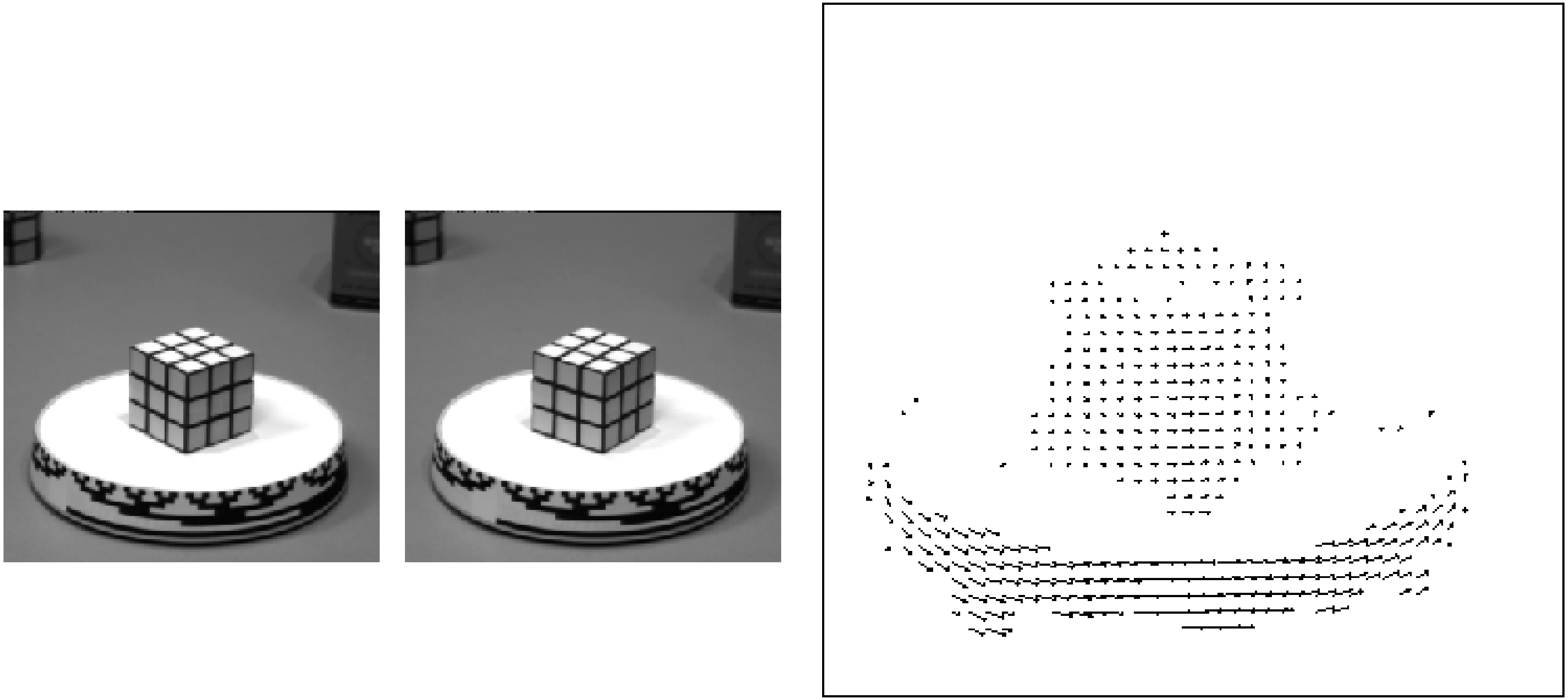
Motion estimation techniques

- Direct methods
  - Directly recover image motion at each pixel from spatio-temporal image brightness variations
  - Dense motion fields, but sensitive to appearance variations
  - Suitable for video when image motion is small
  - Computationally expensive

- Feature-based methods
  - Extract visual features (corners, textured areas) and track them over multiple frames
  - Sparse motion fields, but more robust tracking
  - Suitable when image motion is large (10s of pixels)
  - Usually sufficient

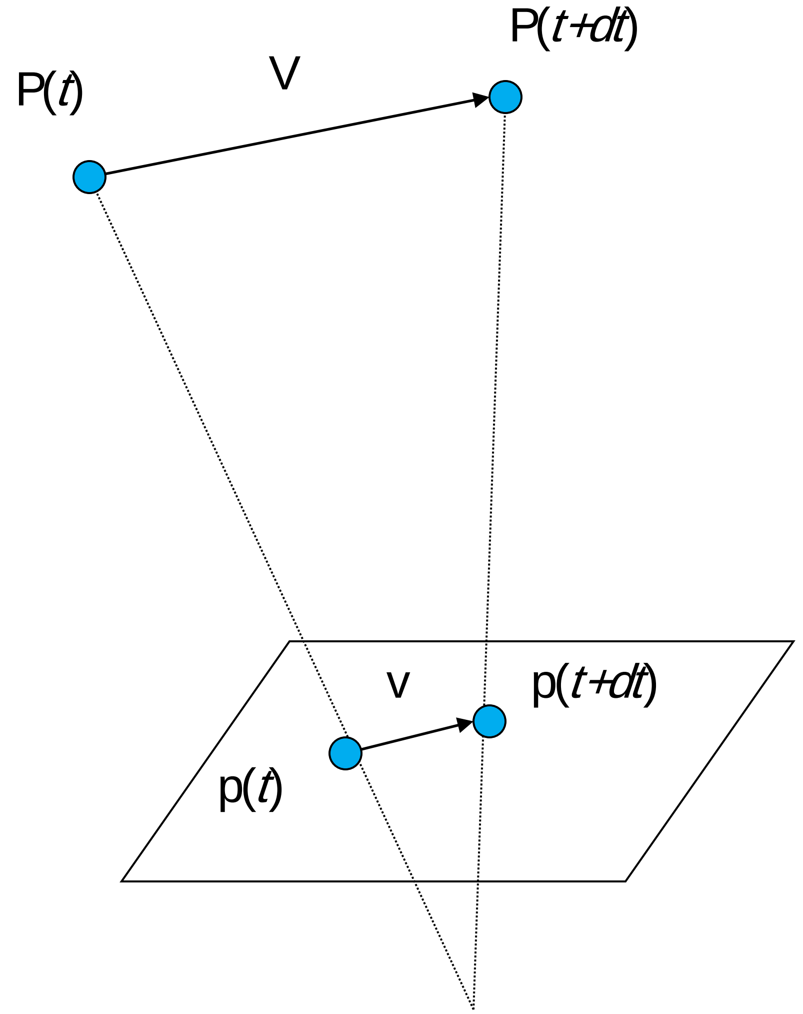
Motion field

- The motion field is the projection of the 3D scene motion into the image



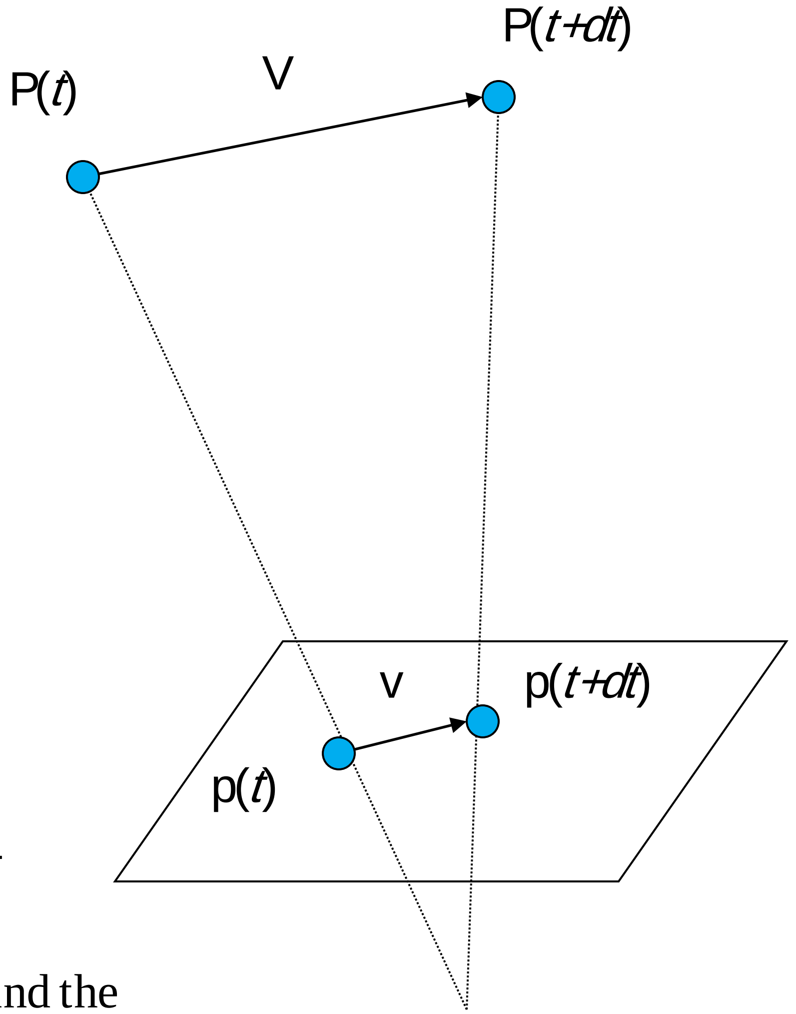
Motion field and parallax

- $\mathbf{P}(t)$ is a moving 3D point
- Velocity of scene point: $\mathbf{V} = d\mathbf{P}/dt$
- $\mathbf{p}(t) = (x(t), y(t))$ is the projection of $\mathbf{P}$ in the image
- Apparent velocity $\mathbf{v}$ in the image: given by components $v_x = dx/dt$ and $v_y = dy/dt$
- These components are known as the *motion field* of the image



Motion field and parallax

$$\mathbf{V} = (V_x, V_y, V_Z) \qquad \mathbf{p} = f\,\frac{\mathbf{P}}{Z}$$



To find image velocity v, differentiate
p with respect to *t* (using quotient rule):

$$\mathbf{v} = f\,\frac{Z\mathbf{V} - V_z\mathbf{P}}{Z^2}$$

$$v_x = \frac{fV_x - V_z x}{Z} \qquad v_y = \frac{fV_y - V_z y}{Z}$$

Image motion is a function of both the 3D motion (V) and the
depth of the 3D point (*Z*)

Motion field and parallax

- Pure translation: **V** is constant everywhere

$$v_x = \frac{fV_x - V_z x}{Z}$$

$$v_y = \frac{fV_y - V_z y}{Z}$$

$$\mathbf{v} = \frac{1}{Z}(\mathbf{v}_0 - V_z \mathbf{p}),$$

$$\mathbf{v}_0 = \left(fV_x, fV_y\right)$$

Motion field and parallax

- Pure translation: **V** is constant everywhere

$$\mathbf{v} = \frac{1}{Z}(\mathbf{v}_0 - V_z\mathbf{p}),$$

$$\mathbf{v}_0 = \left(fV_x, fV_y\right)$$

- V_z is nonzero:
  - Every motion vector points toward (or away from) $\mathbf{v}_0$, the vanishing point of the translation direction

Motion field and parallax

- Pure translation: **V** is constant everywhere

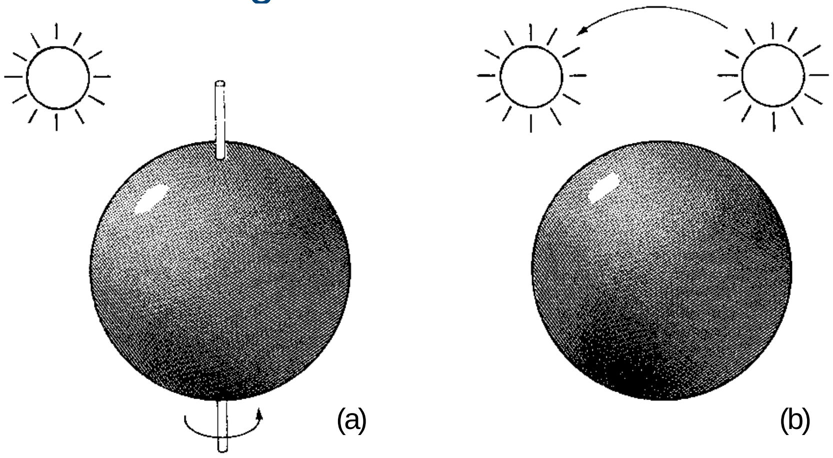
$$\mathbf{v} = \frac{1}{Z}(\mathbf{v}_0 - V_z\mathbf{p}),$$

$$\mathbf{v}_0 = \left(fV_x, fV_y\right)$$

- V_z is nonzero:
  - Every motion vector points toward (or away from) $\mathbf{v}_0$, the vanishing point of the translation direction
- V_z is zero:
  - Motion is parallel to the image plane, all the motion vectors are parallel
- The length of the motion vectors is inversely proportional to the depth Z

Optical flow

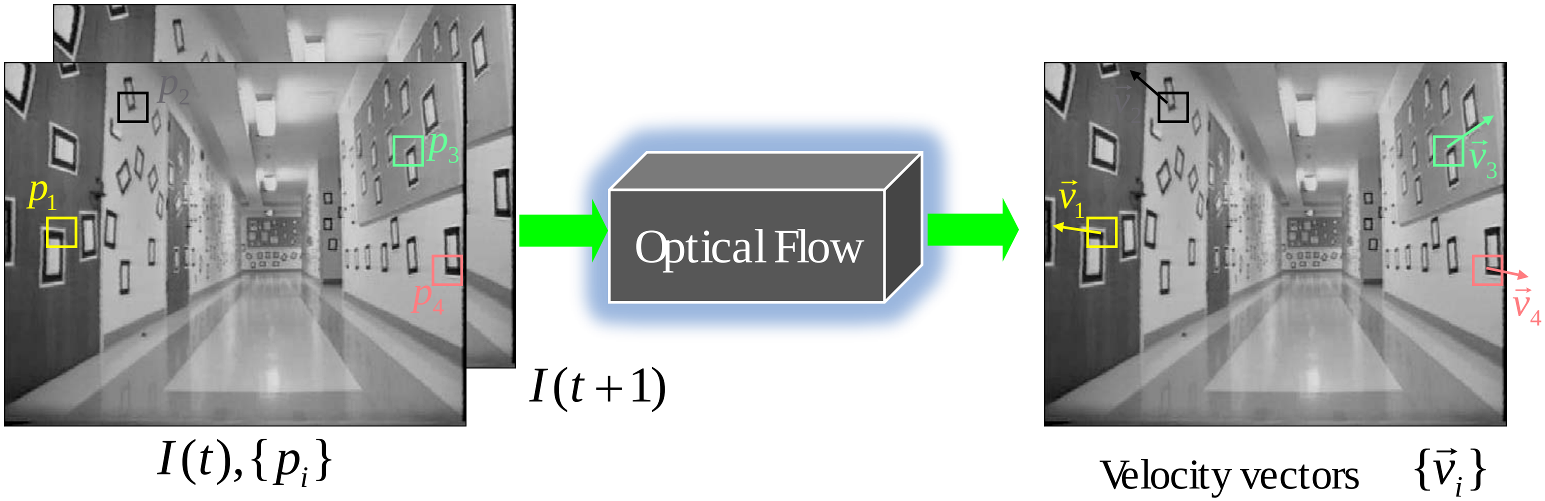
- Definition: optical flow is the *apparent* motion of brightness patterns in the image
- Ideally, optical flow would be the same as the motion field
- Have to be careful: apparent motion can be caused by lighting changes without any actual motion
  - Think of a uniform rotating sphere under fixed lighting vs. a stationary sphere under moving illumination



(a) A smooth sphere is rotating under constant illumination. Thus the optical flow field is zero, but the motion field is not.

(b) A fixed sphere is illuminated by a moving source—the shading of the image changes. Thus the motion field is zero, but the optical flow field is not.

What is Optical Flow?

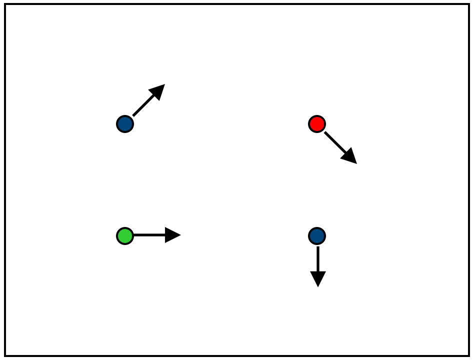


Common assumption:

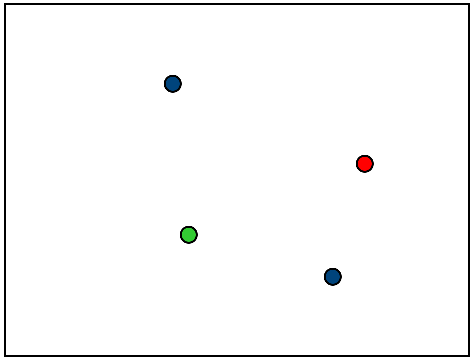
The appearance of the image patches do not change (brightness constancy)

$$I(p_i, t) = I(p_i + \vec{v}_i, t+1)$$

Problem definition: optical flow



$H(x, y)$



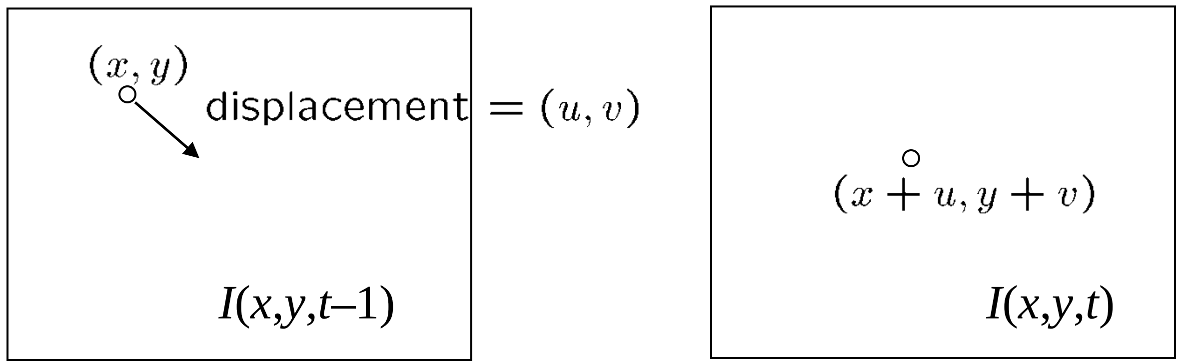
$I(x, y)$

- How to estimate pixel motion from image H to image I?
  - Solve pixel correspondence problem
    - given a pixel in H, look for nearby pixels of the same color in I

Key assumptions

- color constancy: a point in H looks the same in I
  - For grayscale images, this is brightness constancy
- small motion: points do not move very far

The brightness constancy constraint



- Brightness Constancy Equation:

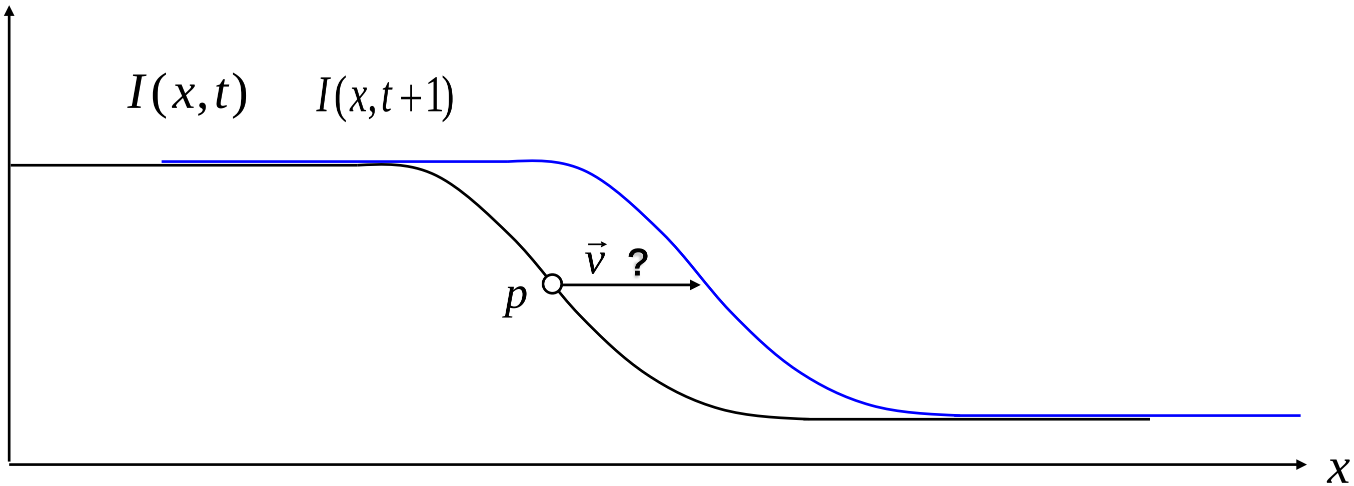
$$I(x, y, t-1) = I(x + u(x, y), y + v(x, y), t)$$

Linearizing the right side using Taylor expansion:

$$I(x, y, t-1) \approx I(x, y, t) + I_x \cdot u(x, y) + I_y \cdot v(x, y)$$

Hence, $$I_x \cdot u + I_y \cdot v + I_t \approx 0$$

Optical flow in the 1D-case



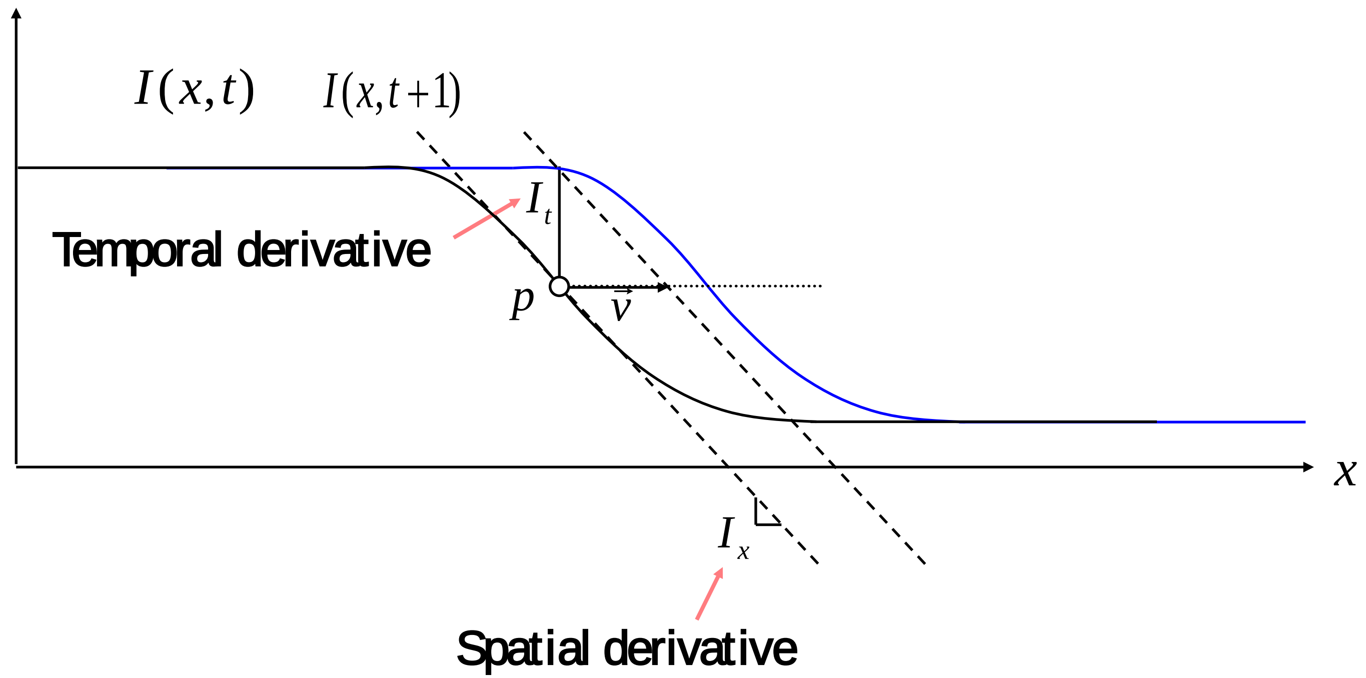
Brightness Constancy Assumption:

$$f(t) \equiv \underbrace{I(x(t),t)}_{} = I(x(t+dt), t+dt)$$

$$\frac{\partial f(x)}{\partial t} = 0$$ Because no change in brightness with time

$$\underbrace{\left.\frac{\partial I}{\partial x}\right|_{t}}_{I_x} \underbrace{\left(\frac{\partial x}{\partial t}\right)}_{v} + \underbrace{\left.\frac{\partial I}{\partial t}\right|_{x(t)}}_{I_t} = 0 \quad \Longrightarrow \quad v = \frac{I_t}{I_x}$$

Optical flow in the 1D-case



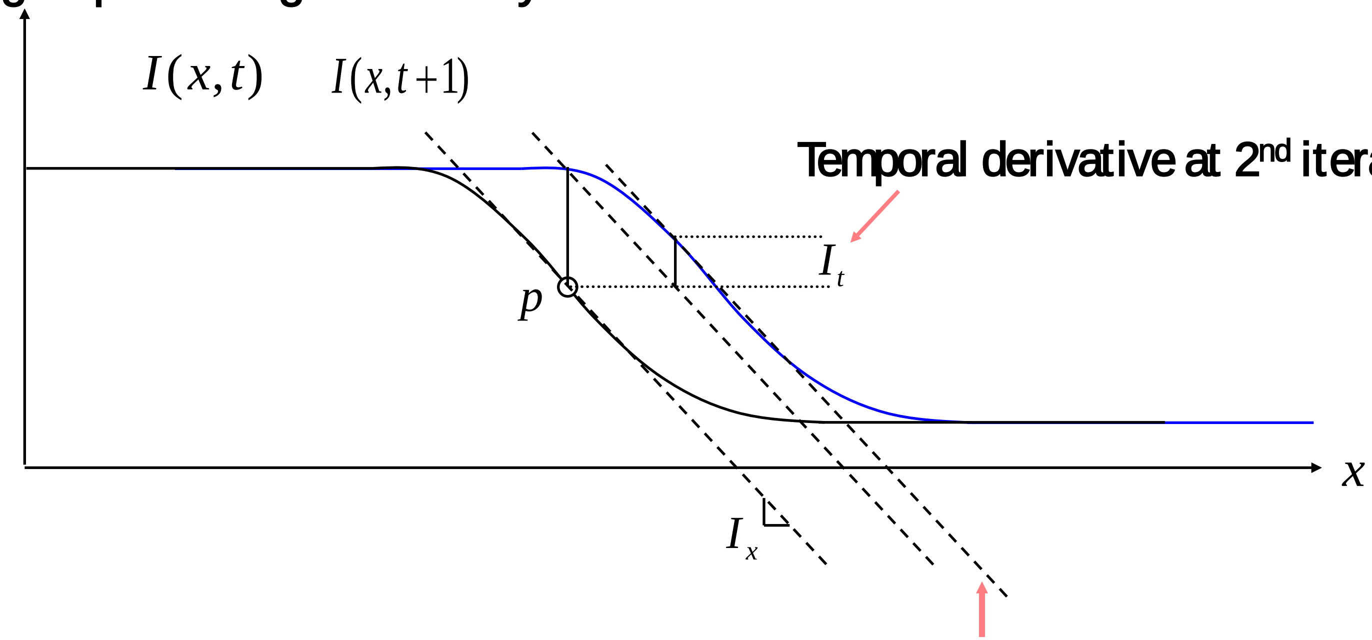
$$I_x = \left.\frac{\partial I}{\partial x}\right|_t \qquad I_t = \left.\frac{\partial I}{\partial t}\right|_{x=p} \qquad \Rightarrow \qquad \vec{v} \approx -\frac{I_t}{I_x}$$

Assumptions:

- Brightness constancy
- Small motion

Optical flow in the 1D-case

Iterating helps refining the velocity vector



$$\vec{v} \leftarrow \vec{v}_{previous} - \frac{I_t}{I_x}$$

Converges in about 5 iterations

Optical flow in the 1D-case: Algorithm

For all pixel of interest *p*:

- Compute local image derivative at *p*: I_x
- Initialize velocity vector: $\vec{v} \leftarrow 0$
- Repeat until convergence:
  - Compensate for current velocity vector: $I'(x,t+1) = I(x+\vec{v},t+1)$
  - Compute temporal derivative: $I_t = I'(p,t+1) - I(p,t)$
  - Update velocity vector: $\vec{v} \leftarrow \vec{v} - {I_t}/{I_x}$

Requirements:

- Need access to neighborhood pixels round p to compute I_x
- Need access to the second image patch, for velocity compensation:
  - The pixel data to be accessed in next image depends on current velocity estimate (bad?)
  - Compensation stage requires a bilinear interpolation (because *v* is not integer)
- The image derivative I_x needs to be kept in memory throughout the iteration

From 1D to 2D tracking

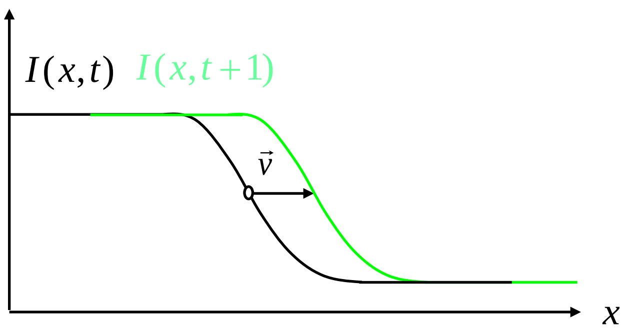
1D: $$\left.\frac{\partial I}{\partial x}\right|_{t}\left(\frac{\partial x}{\partial t}\right)+\left.\frac{\partial I}{\partial t}\right|_{x(t)}=0$$

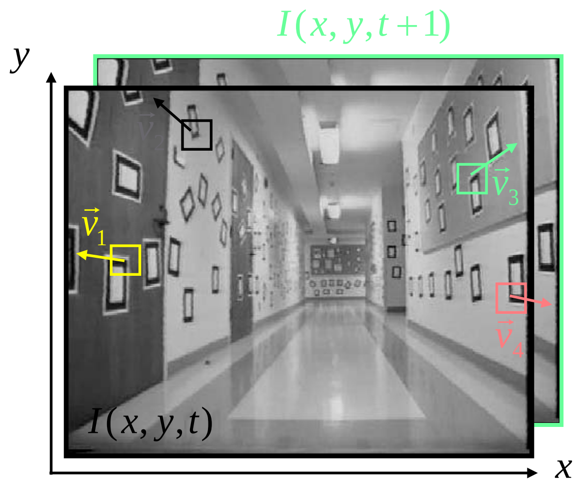
2D: $$\left.\frac{\partial I}{\partial x}\right|_{t}\left(\frac{\partial x}{\partial t}\right)+\left.\frac{\partial I}{\partial y}\right|_{t}\left(\frac{\partial y}{\partial t}\right)+\left.\frac{\partial I}{\partial t}\right|_{x(t)}=0$$

$$\left.\frac{\partial I}{\partial x}\right|_{t}u+\left.\frac{\partial I}{\partial y}\right|_{t}v+\left.\frac{\partial I}{\partial t}\right|_{x(t)}=0$$

! One equation, two velocity *(u,v)* unknowns…

From 1D to 2D optical flow





The math is very similar:

$$\vec{v} \approx -\frac{I_t}{I_x}$$

$$\begin{cases} \vec{v} \approx -G^{-1}b \\ G = \sum_{\text{window around } p} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \\ b = \sum_{\text{window around } p} \begin{bmatrix} I_x I_t \\ I_y I_t \end{bmatrix} \end{cases}$$

Window size here ~ 5x5 or 11x11

Optical Flow Constraint Equation

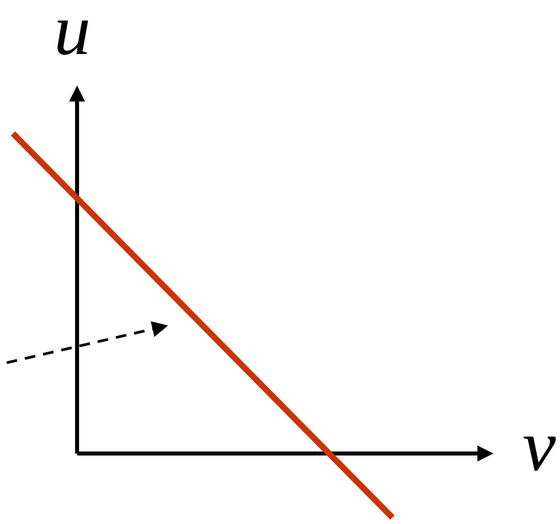
$$\delta x \frac{\partial E}{\partial x} + \delta y \frac{\partial E}{\partial y} + \delta t \frac{\partial E}{\partial t} = 0$$

Divide by δt and take the limit $\delta t \rightarrow 0$

$$\frac{dx}{dt}\frac{\partial E}{\partial x} + \frac{dy}{dt}\frac{\partial E}{\partial y} + \frac{\partial E}{\partial t} = 0$$

Constraint Equation

$$E_x u + E_y v + E_t = 0$$



NOTE: (u, v) must lie on a straight line

We can compute E_x , E_y , E_t using gradient operators!

But, *(u,v)* cannot be found uniquely with this constraint!

Computing Optical Flow

- Assumption 1: Brightness is constant.

$$H(x, y) = I(x+u, y+v)$$

- Assumption 2: Motion is small.

$$I(x+u, y+v) = I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$
$$\approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

(from Taylor series expansion)

Computing Optical Flow

- Combine

shorthand: $I_x = \frac{\partial I}{\partial x}$

$$
\begin{aligned}
0 &= I(x+u, y+v) - H(x,y) \\
&\approx I(x,y) + I_x u + I_y v - H(x,y) \\
&\approx \underbrace{(I(x,y) - H(x,y))}_{I_t} + I_x u + I_y v \\
&\approx I_t + I_x u + I_y v
\end{aligned}
$$

In the limit as u and v goes to zero, the equation becomes exact

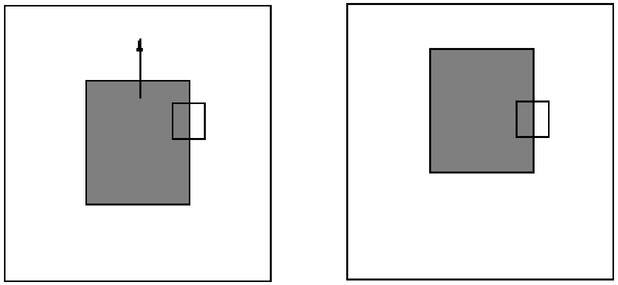
$$0 = I_t + I_x u + I_y v \qquad \text{(optical flow equation)}$$

Computing Optical Flow

- At each pixel, we have one equation, two unknowns.

$$0 = I_t + I_x u + I_y v \qquad \text{(optical flow equation)}$$

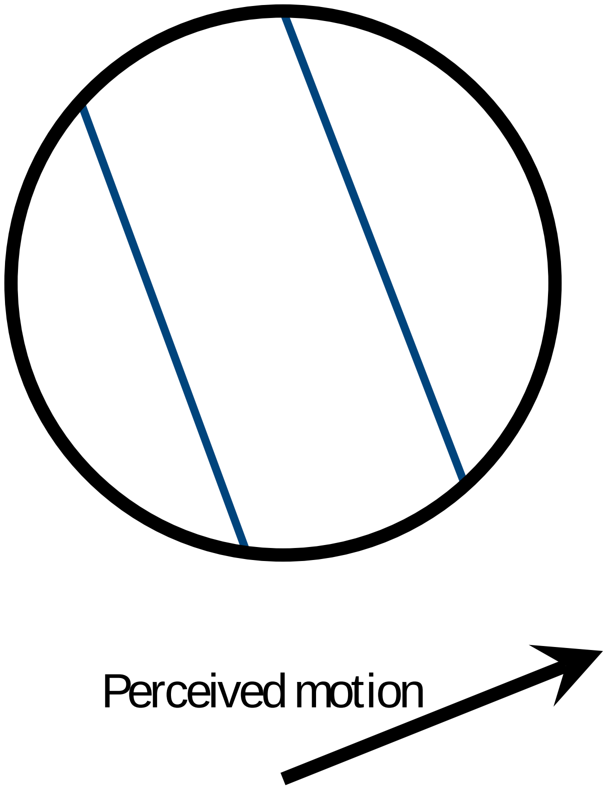
- This means that only the flow component in the gradient direction can be determined.



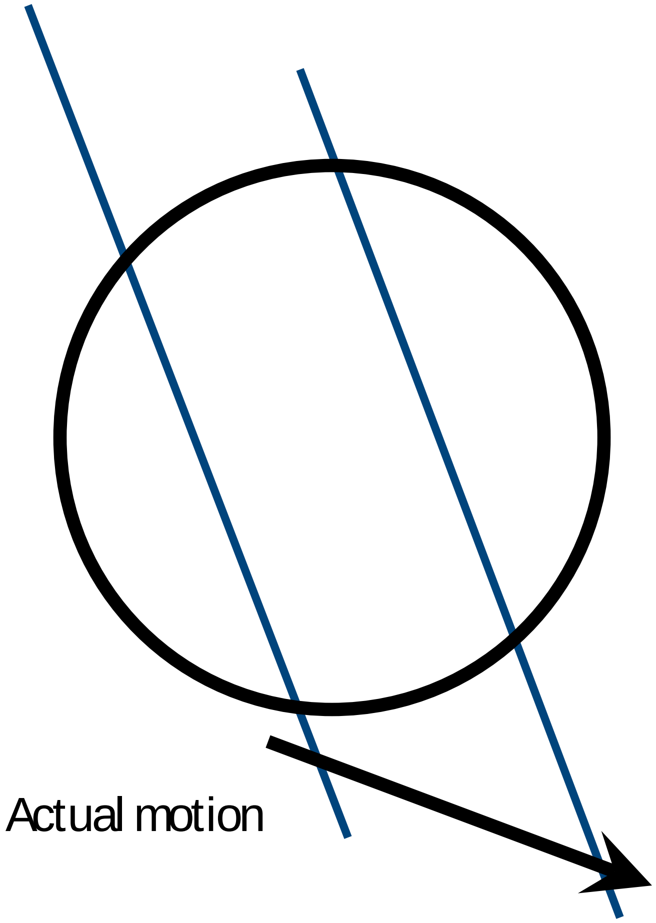
The motion is parallel to the edge, and it cannot be determined.

This is called the *aperture problem*.

The aperture problem



The aperture problem



The barber pole illusion

http://en.wikipedia.org/wiki/Barberpole_illusion

The brightness constancy constraint

$$I_x \cdot u + I_y \cdot v + I_t = 0$$

- How many equations and unknowns per pixel?
  - One equation, two unknowns

- Intuitively, what does this constraint mean?

$$\nabla I \cdot (u, v) + I_t = 0$$

- The component of the flow perpendicular to the gradient (i.e., parallel to the edge) is unknown

The brightness constancy constraint

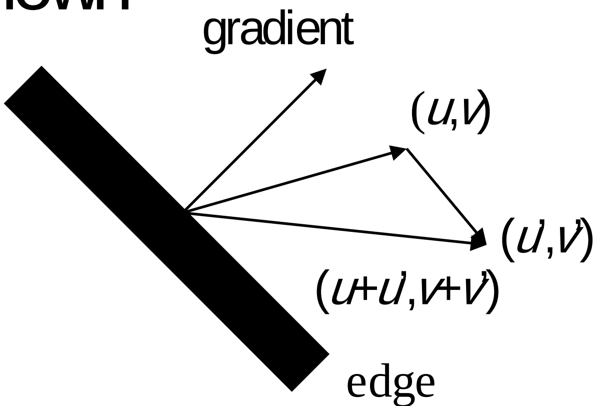
$$I_x \cdot u + I_y \cdot v + I_t = 0$$

- How many equations and unknowns per pixel?
  - One equation, two unknowns

- Intuitively, what does this constraint mean?

$$\nabla I \cdot (u, v) + I_t = 0$$

- The component of the flow perpendicular to the gradient (i.e., parallel to the edge) is unknown

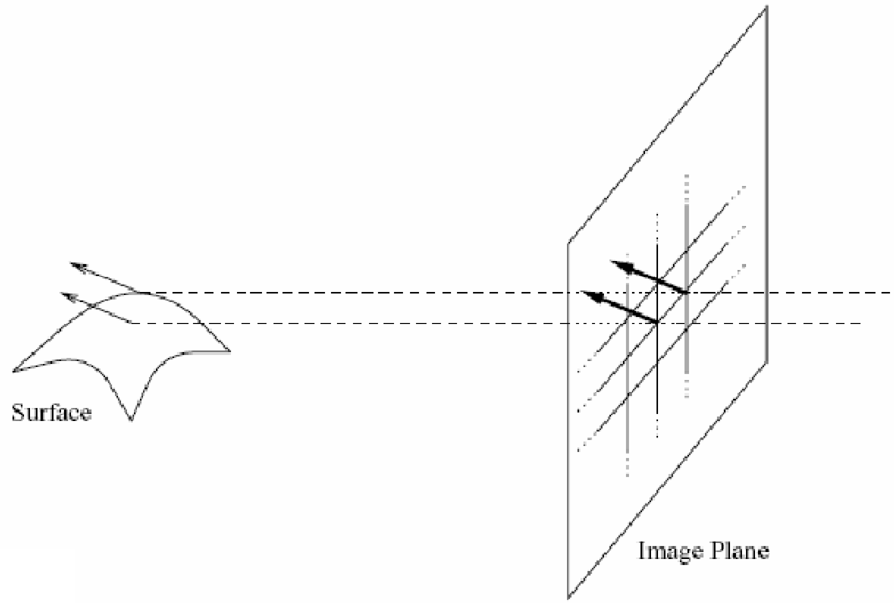


If (u, v) satisfies the equation,
so does $(u+u', v+v')$ if $\nabla I \cdot (u', v') = 0$

Computing Optical Flow

- We need more constraints.
- The most commonly used assumption is that optical flow changes smoothly locally.

Optical Flow Assumptions: Spatial Coherence



Assumption:

- Neighboring points in the scene typically belong to the same surface and thus have similar motions
- Neighboring points project to nearby points in the image, and we expect spatial coherence in image flow

Solving the aperture problem

- How to get more equations for a pixel?
- **Spatial coherence constraint:** pretend the pixel's neighbors have the same (u,v) $0 = I_t(\mathbf{p_i}) + \nabla I(\mathbf{p_i}) \cdot [u \;\; v]$
  - If we use a 5x5 window, that gives us 25 equations per pixel

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix}$$

Lukas-Kanade flow

- Problem: we have more equations than unknowns

$$\underset{25\text{x}2}{A}\;\underset{2\text{x}1}{d} = \underset{25\text{x}1}{b} \longrightarrow \text{minimize } \|Ad - b\|^2$$

- Solution: solve least squares problem
  - minimum least squares solution given by solution (in d) of:

$$\underset{2\text{x}2}{(A^T A)}\;\underset{2\text{x}1}{d} = \underset{2\text{x}1}{A^T b}$$

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b}$$

  - The summations are over all pixels in the K x K window
  - This technique was first proposed by Lukas & Kanade (1981

B. Lucas and T. Kanade. [An iterative image registration technique with an application to stereo vision.] In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

Conditions for solvability

– Optimal (u, v) satisfies Lucas-Kanade equation

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b}$$

When is This Solvable?

- A^TA should be invertible
- A^TA should not be too small due to noise
  - eigenvalues λ_1 and λ_2 of A^TA should not be too small
- A^TA should be well-conditioned
  - λ_1/λ_2 should not be too large (λ_1 = larger eigenvalue)

*From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Eigenvectors of A^TA

$$A^TA = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \; I_y] = \sum \nabla I (\nabla I)^T$$

- Suppose (x,y) is on an edge. What is A^TA?
  - gradients along edge all point the same direction
  - gradients away from edge have small magnitude

$$\left(\sum \nabla I (\nabla I)^T\right) \approx k \nabla I \nabla I^T$$

$$\left(\sum \nabla I (\nabla I)^T\right) \nabla I = k \|\nabla I\| \nabla I$$

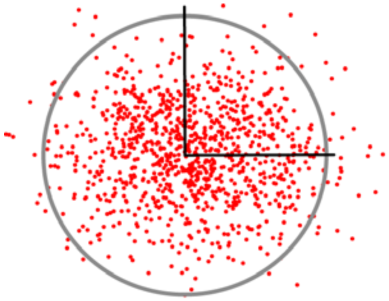
  - ∇I is an eigenvector with eigenvalue $k\|\nabla I\|$
  - What's the other eigenvector of A^TA?
    - let N be perpendicular to ∇I

$$\left(\sum \nabla I (\nabla I)^T\right) N = 0$$

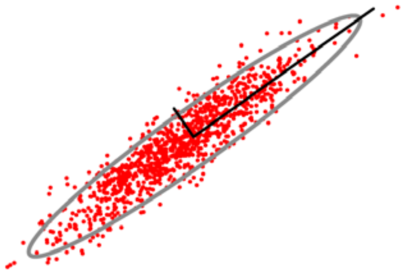
    - N is the second eigenvector with eigenvalue 0
- The eigenvectors of A^TA relate to edge direction and magnitude

Quality of Image Patch

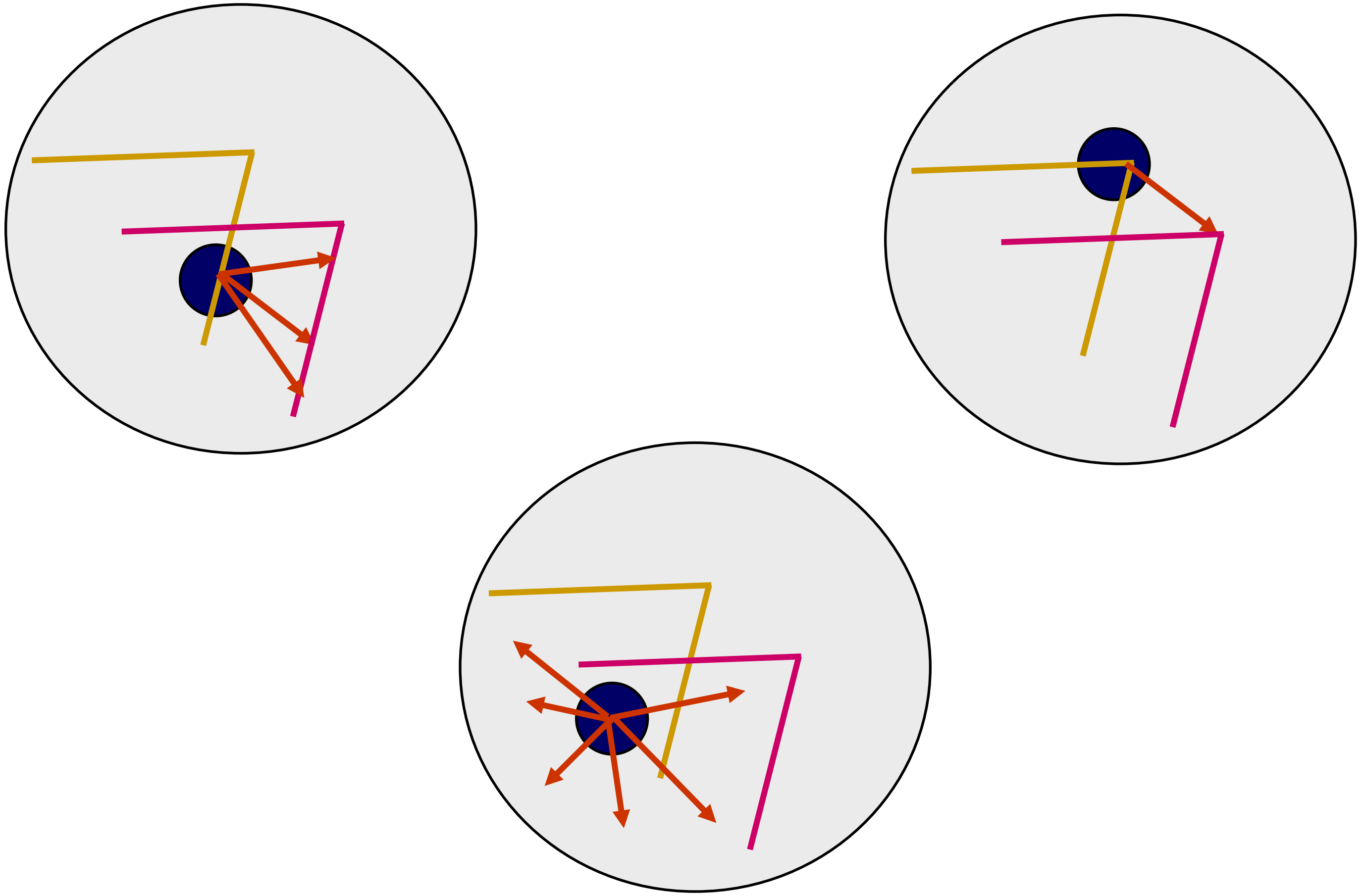
- Eigenvalues of the matrix contain information about local image structure
  - Both eigenvalues (close to) zero: Uniform area
  - One eigenvalue (close to) zero: Edge
  - No eigenvalues (close to) zero: Corner



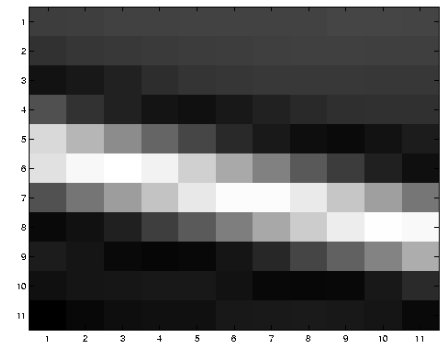
$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_y I_x & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{bmatrix}$$

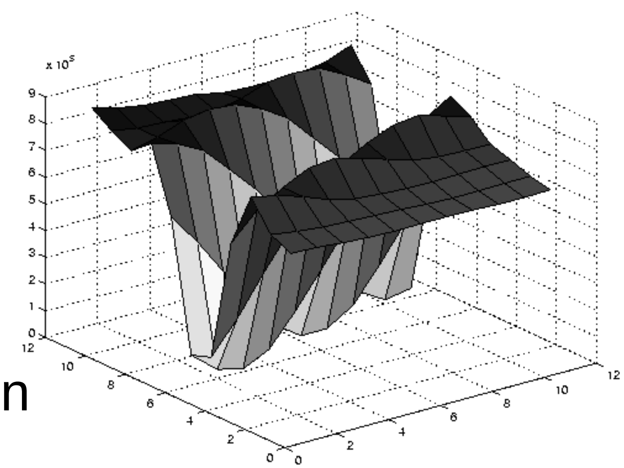


Local Patch Analysis



Edge



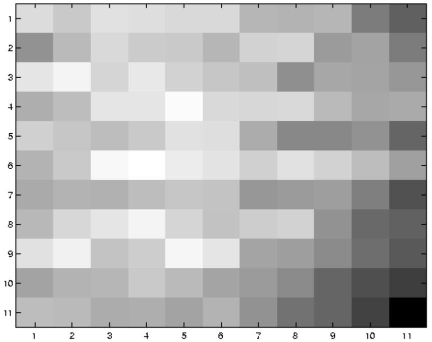


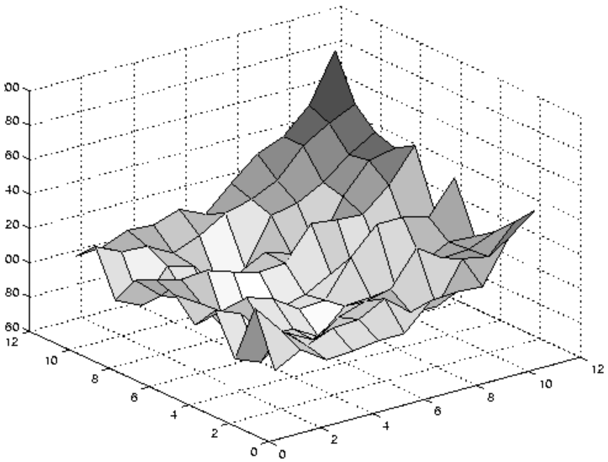
$$\sum \nabla I (\nabla I)^T$$

– large gradients in the same direction

– large λ_1, small λ_2

Low texture region



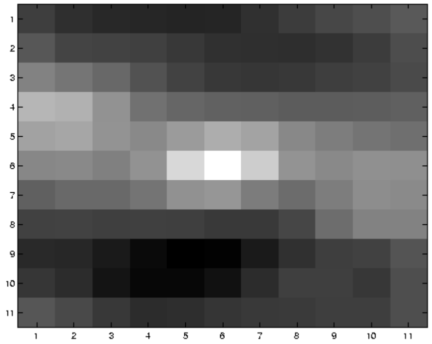


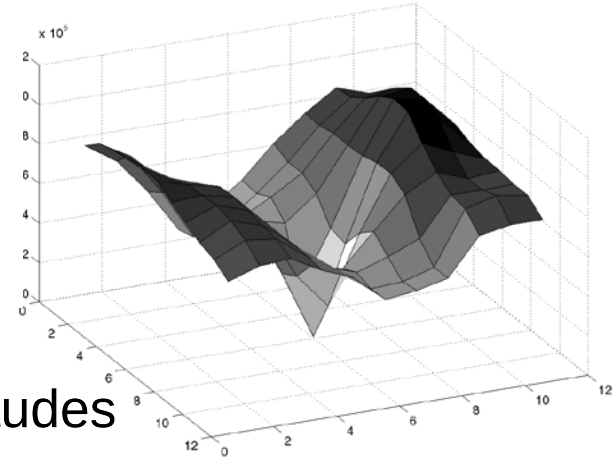
$$\sum \nabla I (\nabla I)^T$$

– gradients have small magnitude

– small λ_1, small λ_2

High textured region





$$\sum \nabla I (\nabla I)^T$$

– gradients are different, large magnitudes

– large λ_1, large λ_2

Computing Optical Flow

- What are the potential causes of errors in this procedure?
  - Brightness constancy is **not** satisfied
  - The motion is **not** small
  - A point does **not** move like its neighbors
    - window size is too large

Improving accuracy

- Recall our small motion assumption

$$0 = I(x+u, y+v) - H(x,y)$$
$$\approx I(x,y) + I_x u + I_y v - H(x,y)$$

- This is not exact
  - To do better, we need to add higher order terms back in:

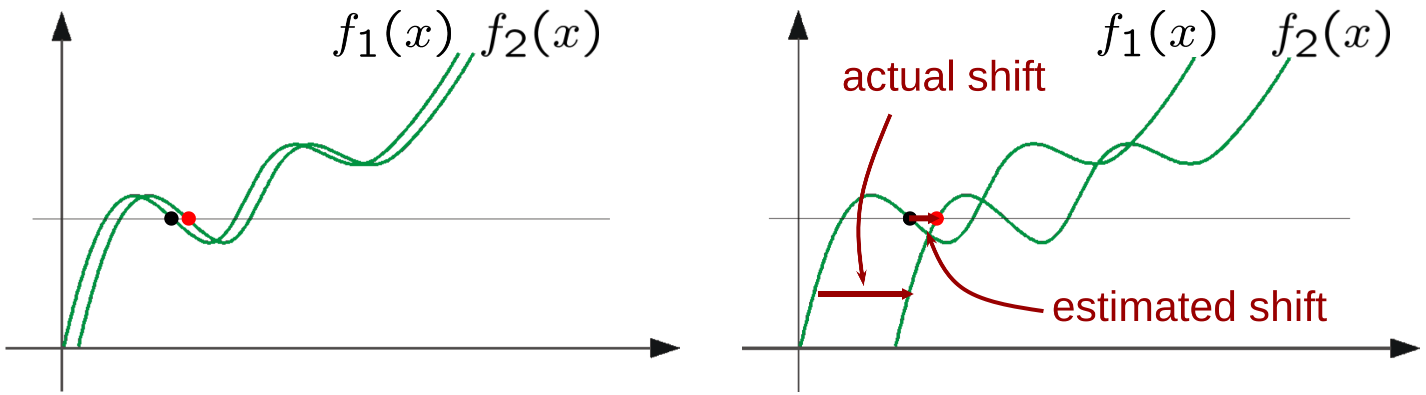
$$= I(x,y) + I_x u + I_y v + \text{higher order terms} - H(x,y)$$

- This is a polynomial root finding problem
  - Can solve using **Newton's method**
    - Also known as **Newton-Raphson** method
  - Lucas-Kanade method does one iteration of Newton's method
    - Better results are obtained via more iterations

Optical Flow: Aliasing

Temporal aliasing causes ambiguities in optical flow because images can have many pixels with the same intensity.

I.e., how do we know which ‘correspondence’ is correct?



nearest match is correct (no aliasing)

nearest match is incorrect (aliasing)

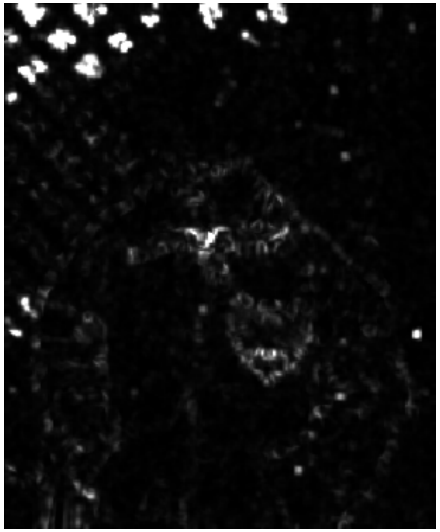
To overcome aliasing: coarse-to-fine estimation.

Corner metrics

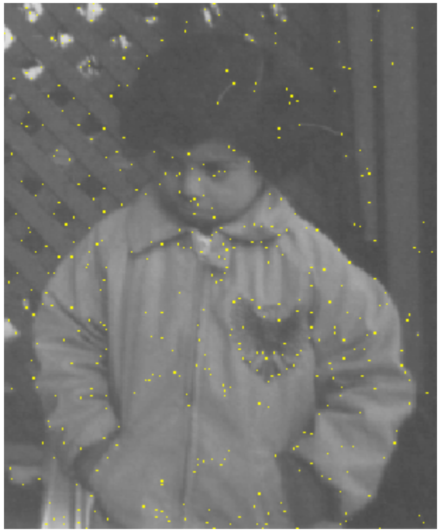
- Image processing toolbox in Matlab has the `cornermetric` function that has two detections
  - Harris corner detection
  - Shi-Tomasi (usually used in Lucas-Kanade optical flow algorithms)

Original

Corner Metric



Corner Points



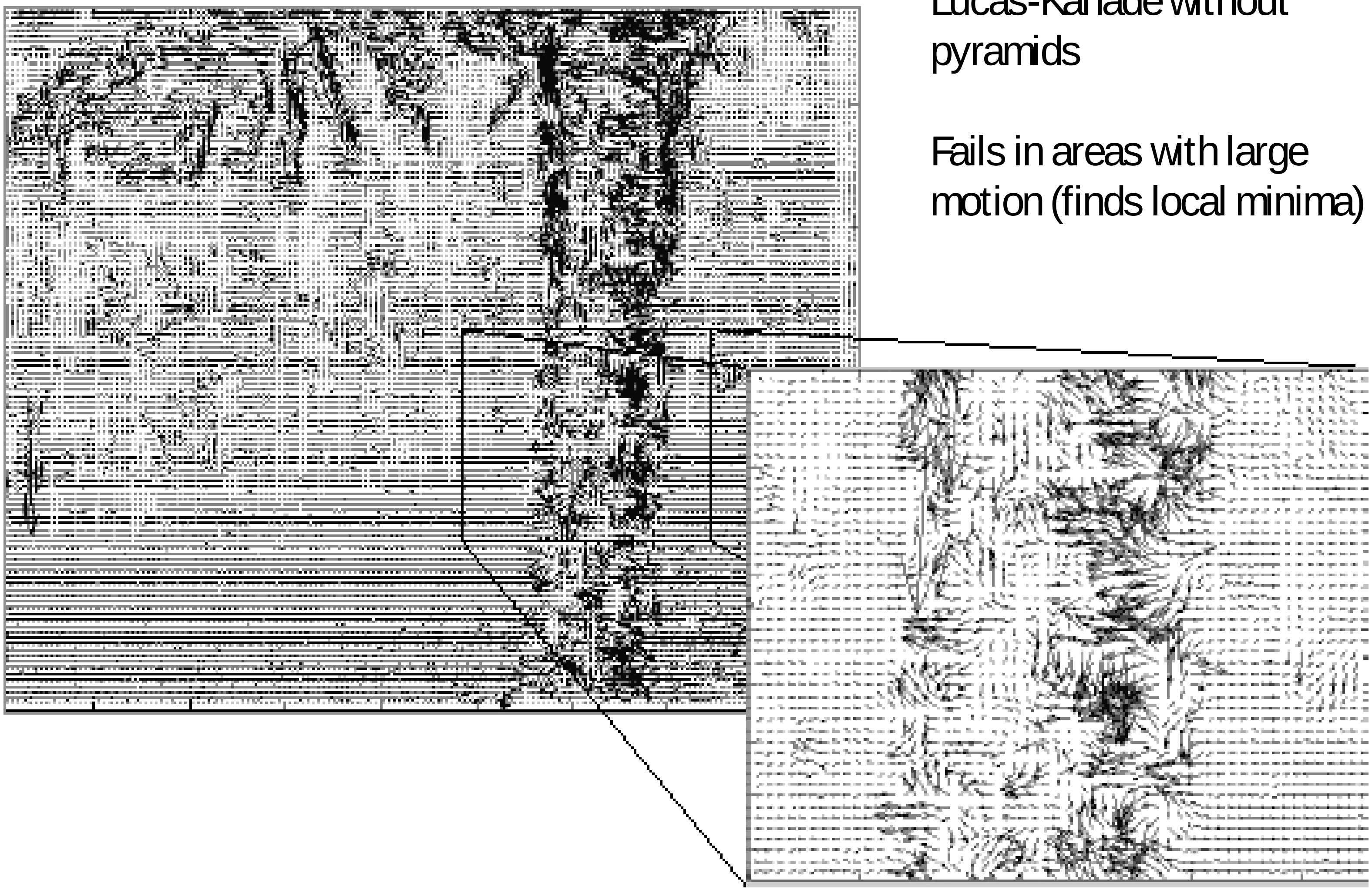
Shi-Tomasi feature tracker

1. Find good features (min eigenvalue of 2×2 Hessian)
2. Use Lucas-Kanade to track with pure translation
3. Use affine registration with first feature patch
4. Terminate tracks whose dissimilarity gets too large from first patch
5. Start new tracks when needed

Jianbo Shi and Carlo Tomasi *Good Features to Track*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94), 1994, pp. 593 - 600.

Dealing with large motions

Optical Flow Results

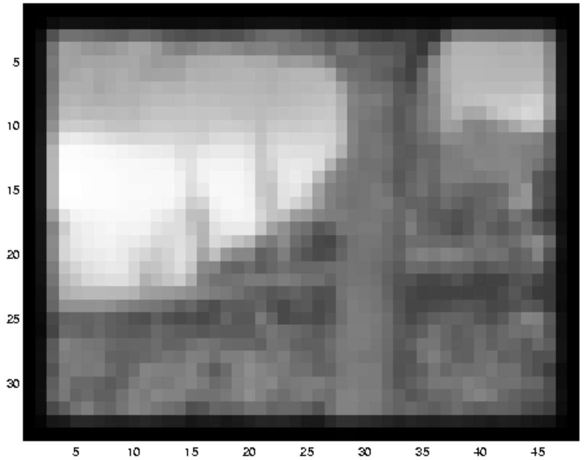


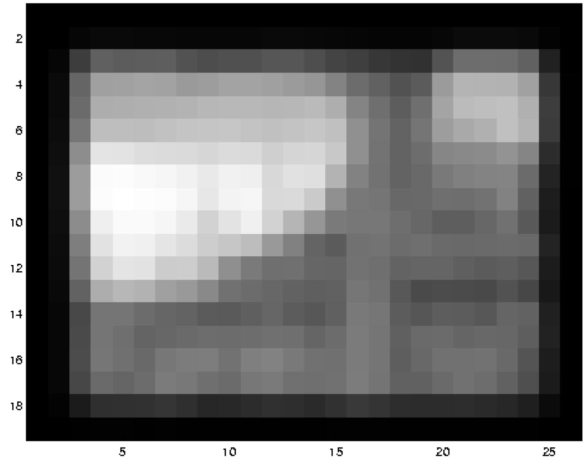
Lucas-Kanade without pyramids

Fails in areas with large motion (finds local minima)

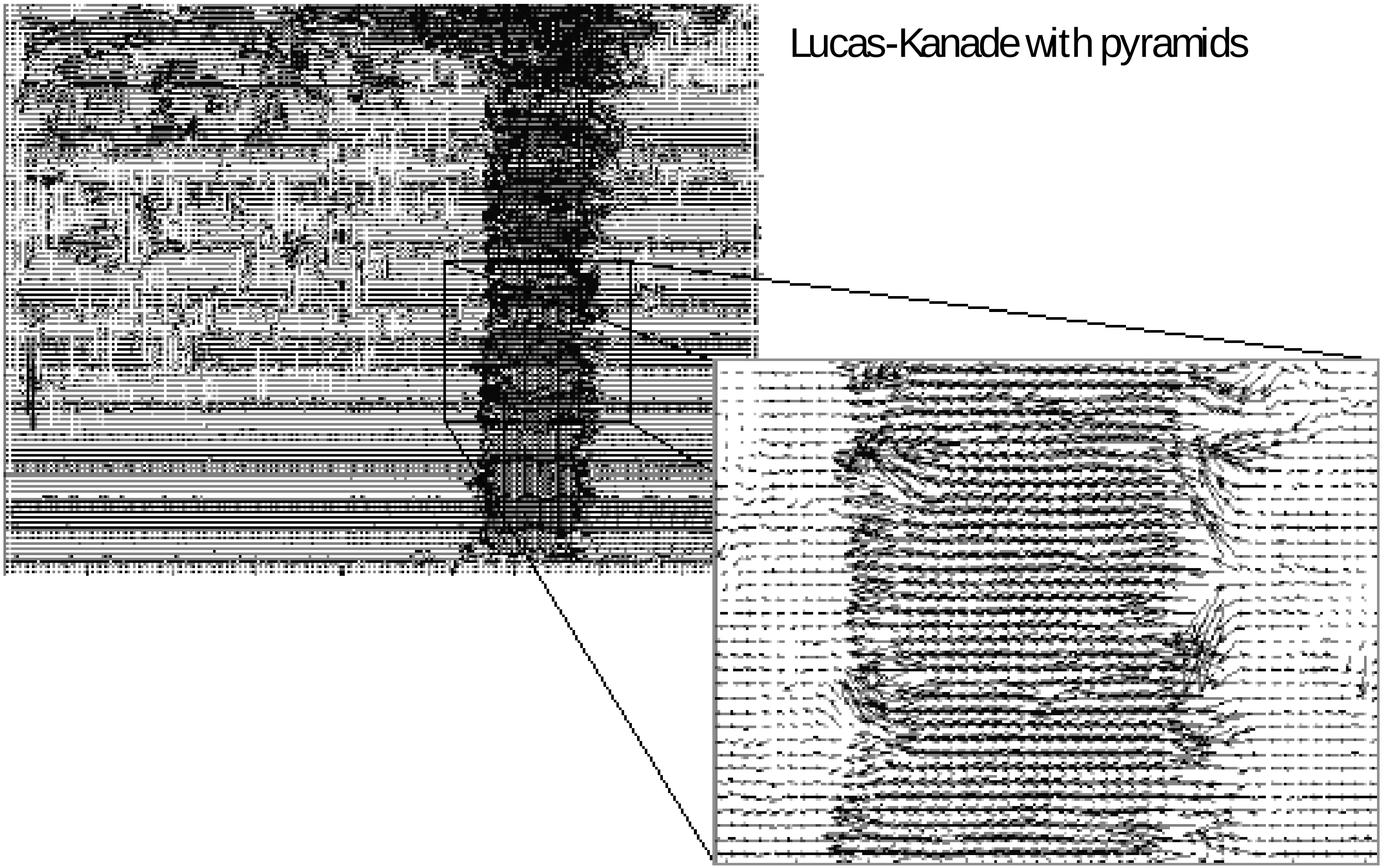
* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Reduce the resolution!





Optical Flow Results

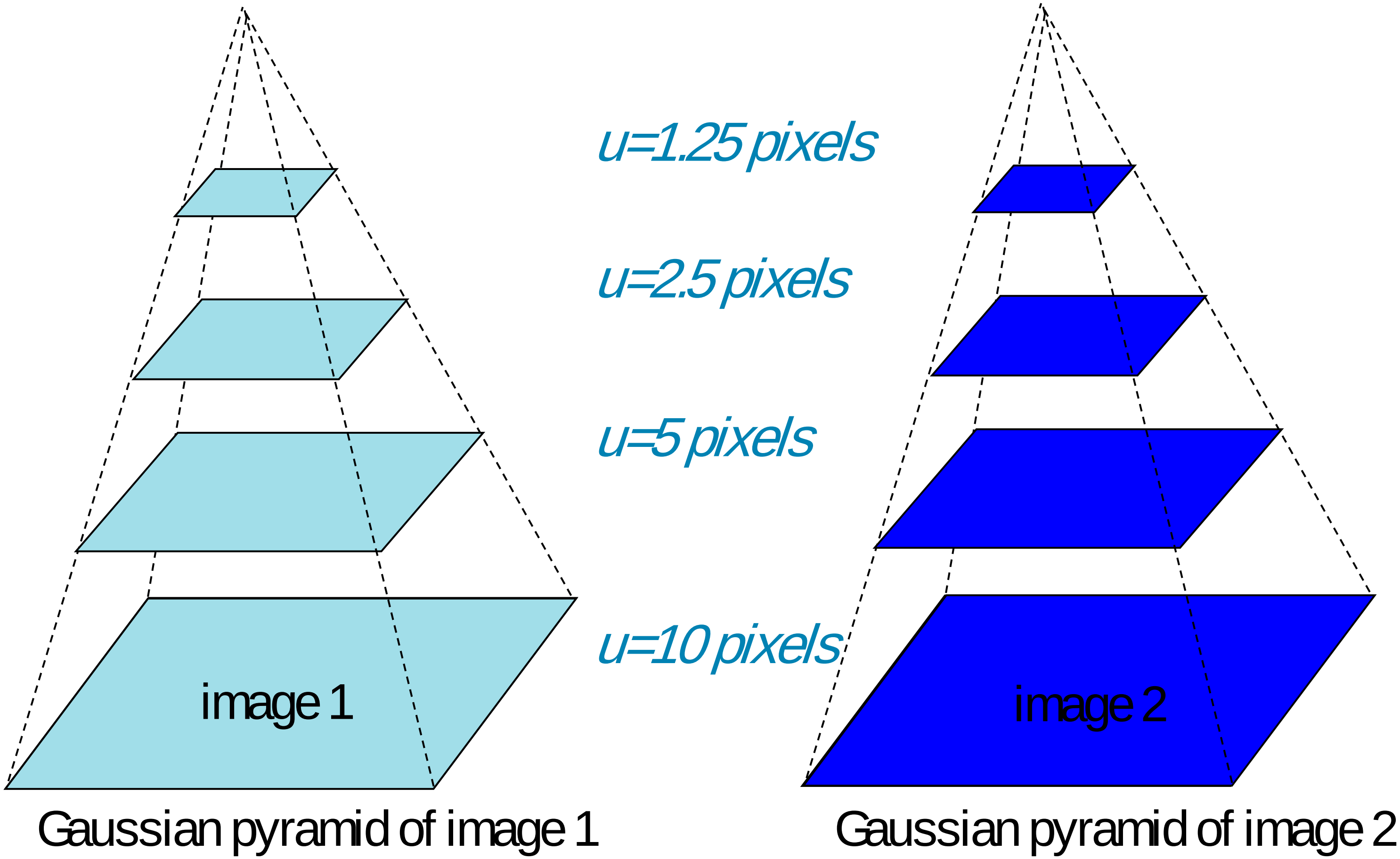


Lucas-Kanade with pyramids

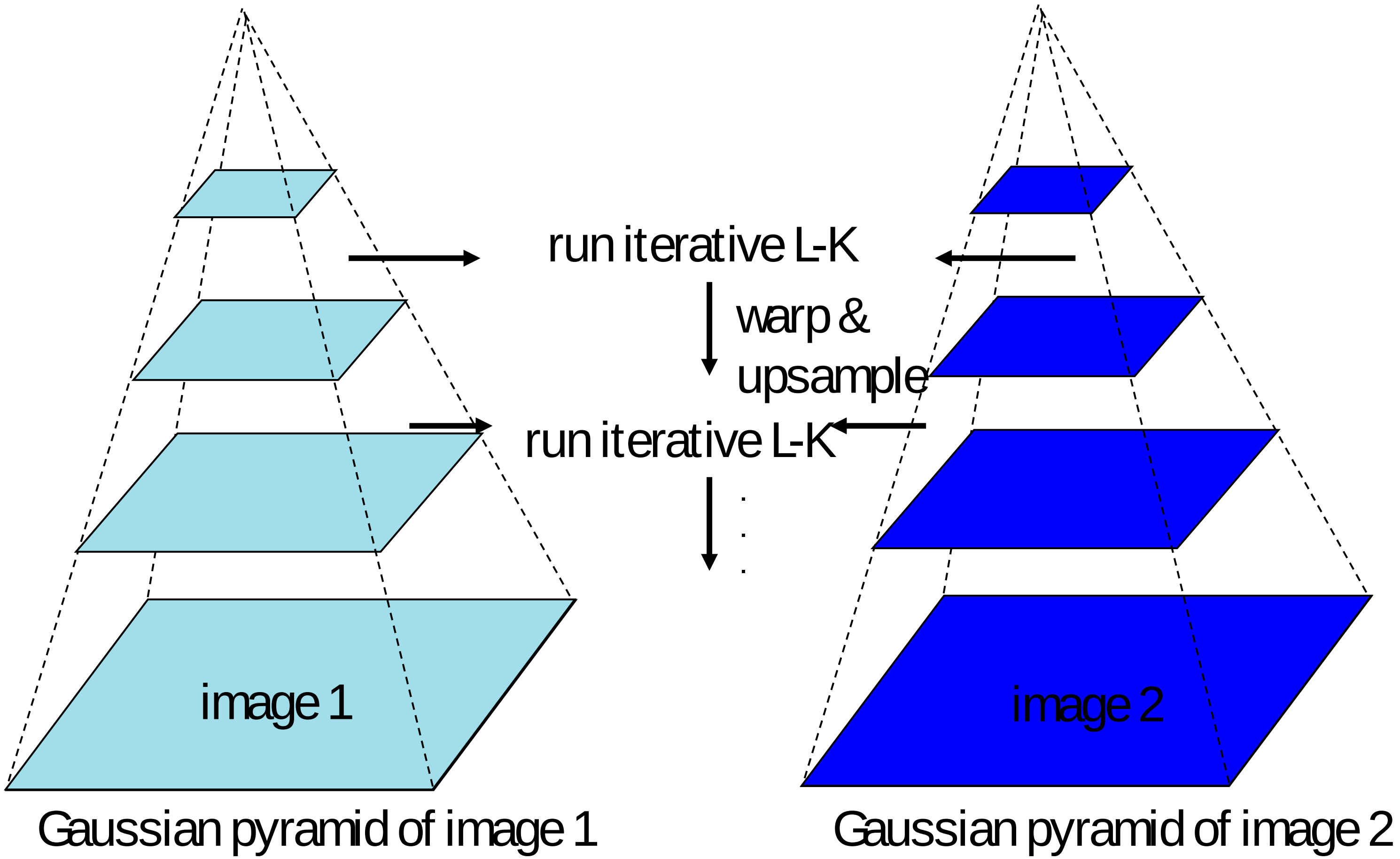
Multi-resolution Lucas Kanade Algorithm

- Compute ('simple') Lucas-Kanade at highest level
- At level *i*
  - Take flow u_{i-1}, v_{i-1} from level *i-1*
  - Bilinear interpolate it to create u_i^*, v_i^* matrices of twice the resoution of level *i*
  - Multiply u_i^*, v_i^* by 2
  - Compute f_t from a block displaced by $u_i^*(x,y), v_i^*(x,y)$
  - Apply LK to get u_i', v_i' the correction in flow.
  - Add corrections, u_i', v_i', i.e., $u_i = u_i^* + u_i'$, $v_i = v_i^* + v_i'$.

Coarse-to-fine optical flow estimation



Coarse-to-fine optical flow estimation



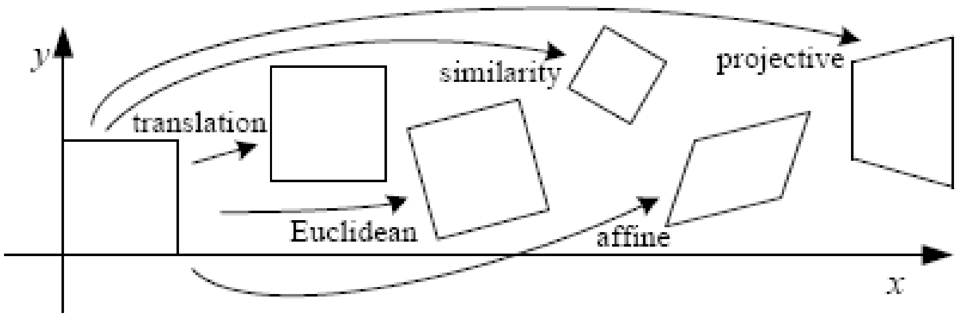
Errors in Lucas-Kanade

- The motion is large (larger than a pixel)
  - Iterative refinement, coarse-to-fine estimation
- A point does not move like its neighbors
  - Motion segmentation
- Brightness constancy does not hold
  - Do exhaustive neighborhood search with normalized correlation

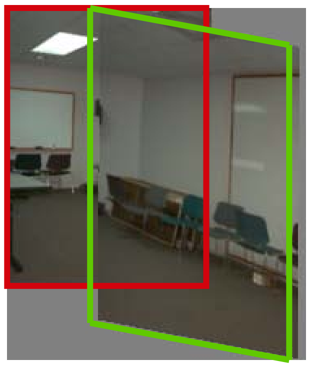
Optical flow: iterative estimation

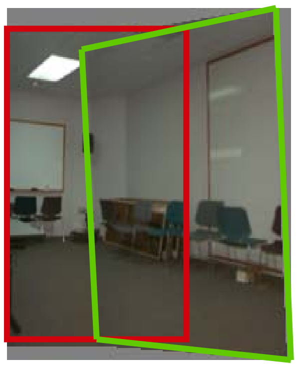
- Some implementation issues:
  - Warping is not easy (ensure that errors in warping are smaller than the estimate refinement)
  - Warp one image, take derivatives of the other so you don't need to re-compute the gradient after each iteration
  - Often useful to low-pass filter the images before motion estimation (for better derivative estimation, and linear approximations to image intensity)

Motion models



Translation | **Affine** | **Perspective** | **3D rotation**

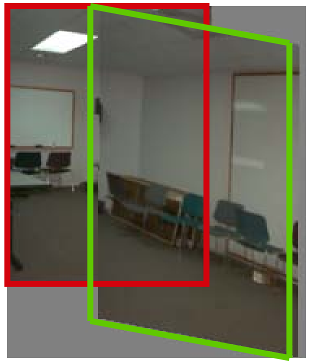


2 unknowns | **6 unknowns** | **8 unknowns** | **3 unknowns**

Affine motion

$$u(x, y) = a_1 + a_2 x + a_3 y$$

$$v(x, y) = a_4 + a_5 x + a_6 y$$



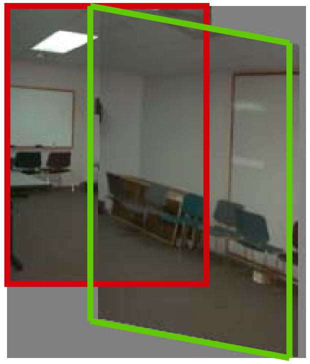
- Substituting into the brightness constancy equation:

$$\boxed{I_x \cdot u + I_y \cdot v + I_t \approx 0}$$

Affine motion

$$u(x,y) = a_1 + a_2 x + a_3 y$$

$$v(x,y) = a_4 + a_5 x + a_6 y$$



- Substituting into the brightness constancy equation:

$$\boxed{I_x(a_1 + a_2 x + a_3 y) + I_y(a_4 + a_5 x + a_6 y) + I_t \approx 0}$$

- Each pixel provides 1 linear constraint in 6 unknowns

- Least squares minimization:

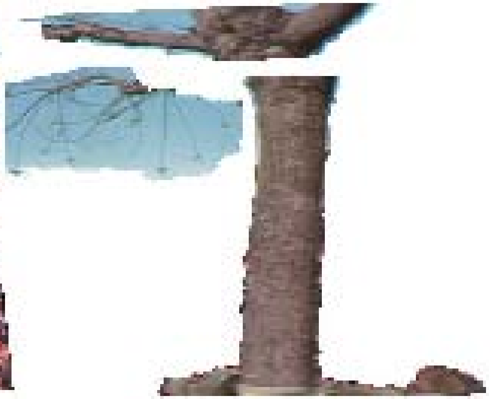
$$\boxed{Err(\vec{a}) = \sum \left[I_x(a_1 + a_2 x + a_3 y) + I_y(a_4 + a_5 x + a_6 y) + I_t \right]^2}$$

Using optical flow: Motion segmentation

- How do we represent the motion in this scene?

Using optical flow: Layered motion

- Break image sequence into "layers" each of which has a coherent motion

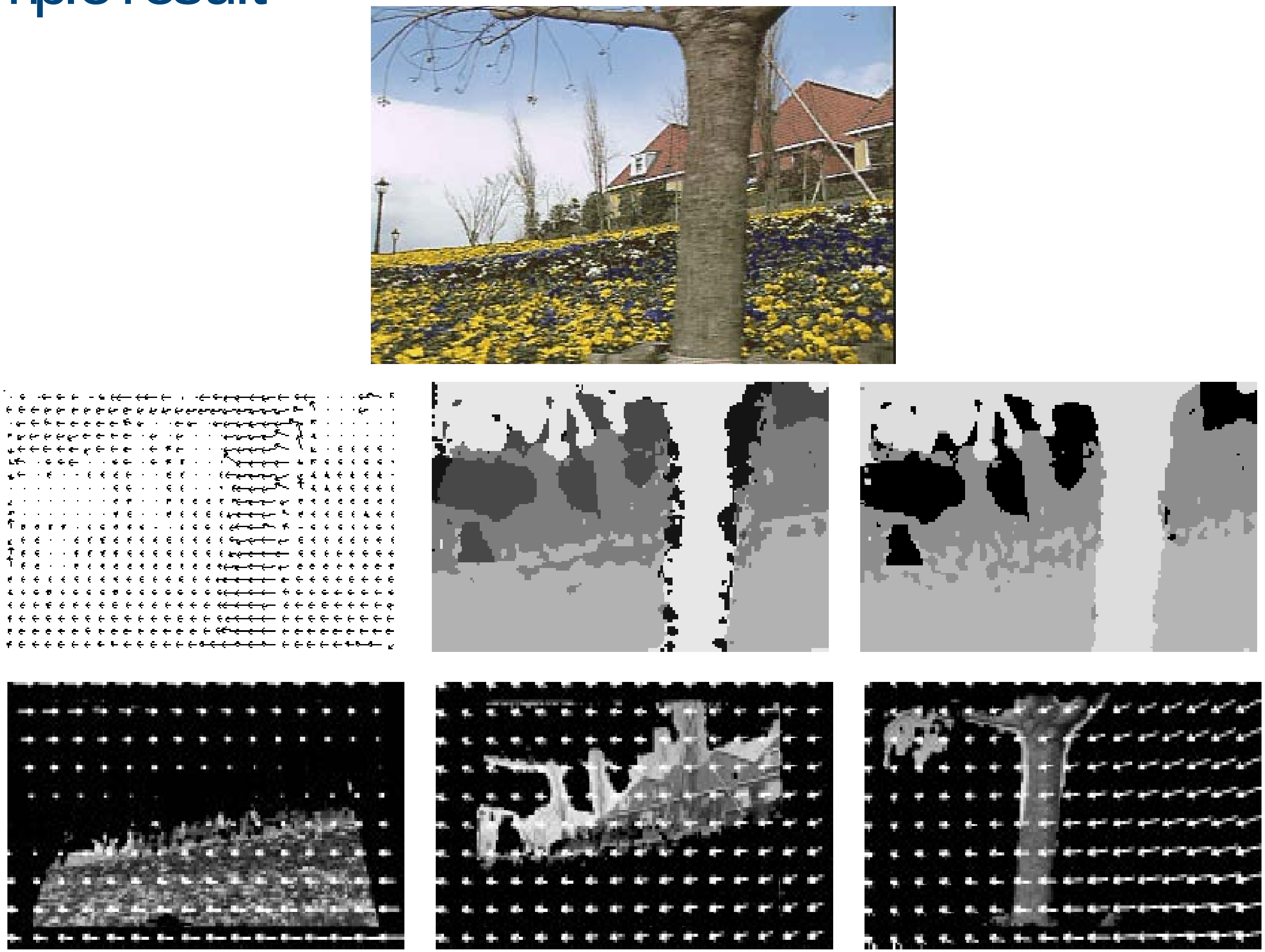


How do we estimate the layers?

- Compute local flow in a coarse-to-fine fashion
- Obtain a set of initial affine motion hypotheses
  - Divide the image into blocks and estimate affine motion parameters in each block by least squares
    - Eliminate hypotheses with high residual error
  - Perform *k-means* clustering on affine motion parameters
    - Merge clusters that are close and retain the largest clusters to obtain a smaller set of hypotheses to describe all the motions in the scene
- Iterate until convergence:
  - Assign each pixel to best hypothesis
    - Pixels with high residual error remain unassigned
  - Perform region filtering to enforce spatial constraints
  - Re-estimate affine motions in each region

J. Wang and E. Adelson. Layered Representation for Motion Analysis. *CVPR 1993*.

Example result



Motion segmentation with an affine model

$$\boxed{u(x,y)} = a_1 + a_2 x + a_3 y$$
$$\boxed{v(x,y)} = a_4 + a_5 x + a_6 y$$

Local flow estimates

Motion segmentation with an affine model

$$\boxed{u(x,y) = a_1 + a_2 x + a_3 y}$$

$$v(x,y) = a_4 + a_5 x + a_6 y$$

Equation of a plane
(parameters a_1, a_2, a_3 can be found by least squares)

Motion segmentation with an affine model

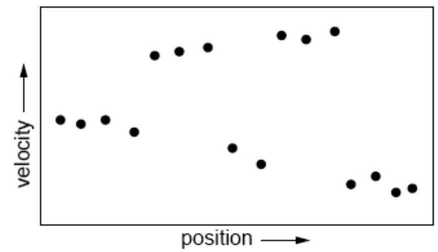
$$\boxed{u(x,y) = a_1 + a_2 x + a_3 y}$$

$$v(x,y) = a_4 + a_5 x + a_6 y$$

Equation of a plane
(parameters a_1, a_2, a_3 can be found by least squares)

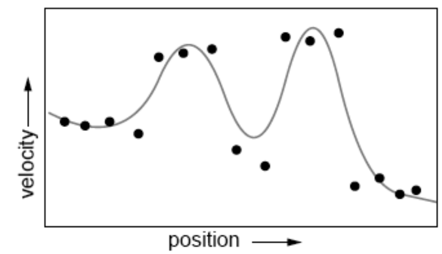
1D example

$u(x,y)$



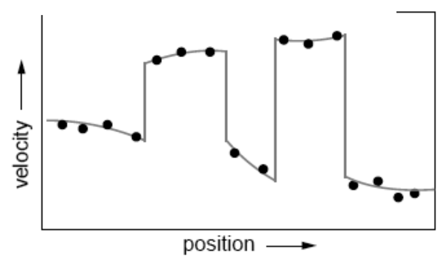
(a) velocity estimates

True flow

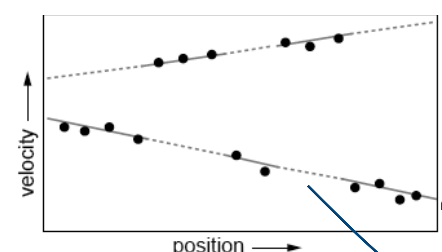


(b) velocity smoothing

Local flow estimate



Segmented estimate



"Foreground"

"Background"

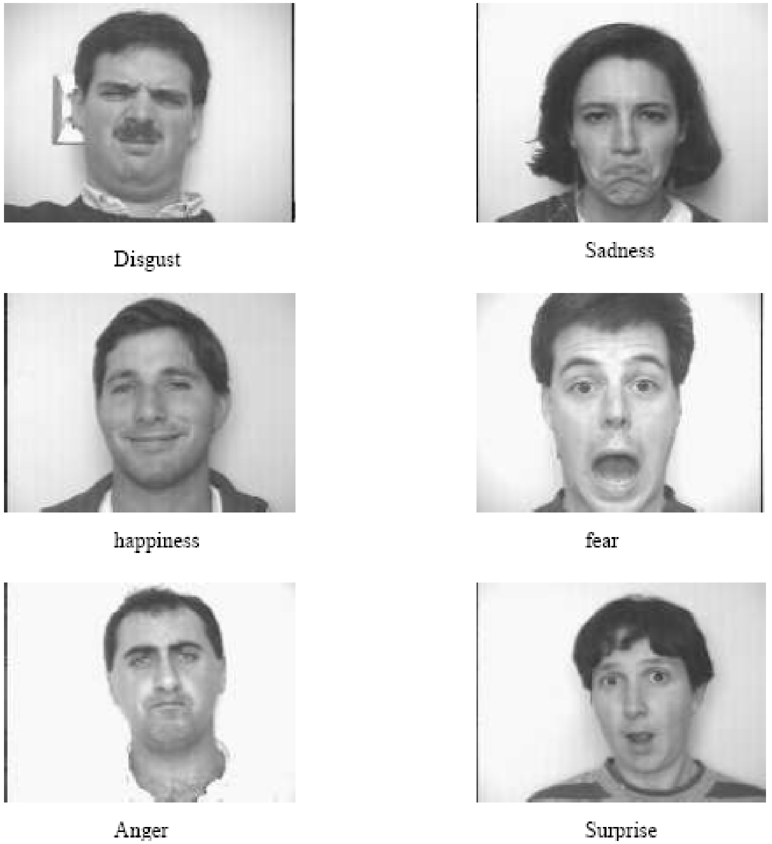
Line fitting

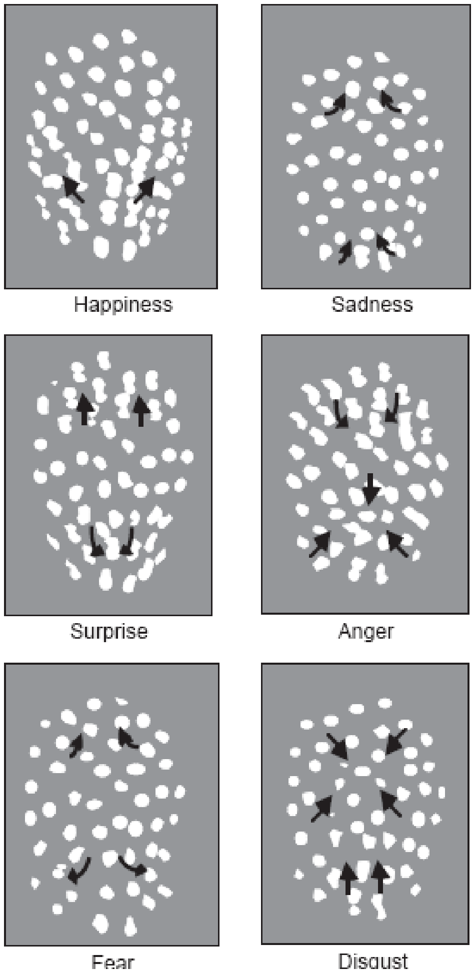
Occlusion

Block-based motion prediction

- Break image up into square blocks
- Estimate translation for each block
- Use this to predict next frame, code difference (MPEG-2)

Using optical flow: recognizing facial expressions

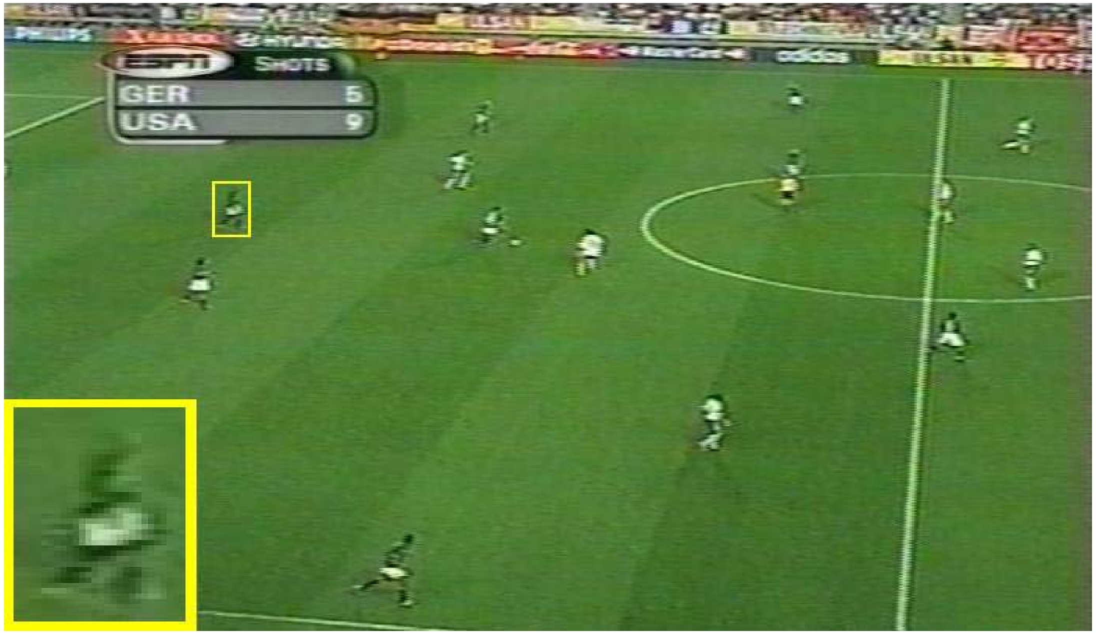




Recognizing Human Facial Expression (1994)
by Yaser Yacoob, Larry S. Davis

Using optical flow: action recognition at a distance

- Features = optical flow within a region of interest
- Classifier = nearest neighbors

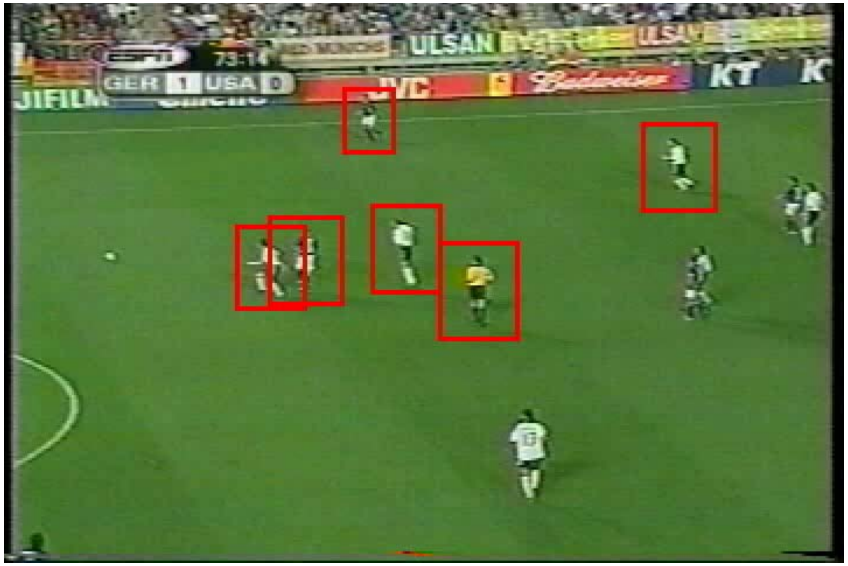


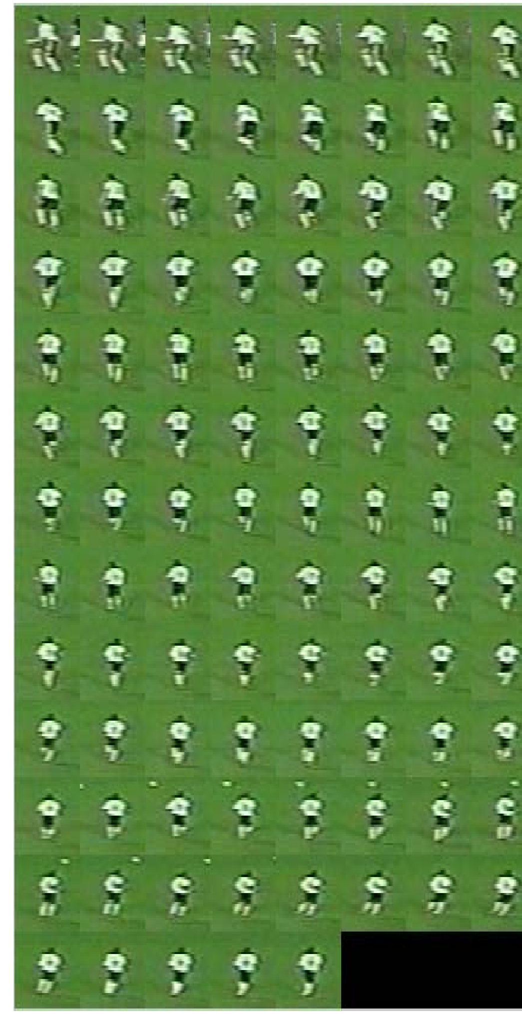
Challenge: low-res data, not going to be able to track each limb.

The 30-Pixel Man

[Efros, Berg, Mori, & Malik 2003]
http://graphics.cs.cmu.edu/people/efros/research/action/

Using optical flow:
action recognition at a distance

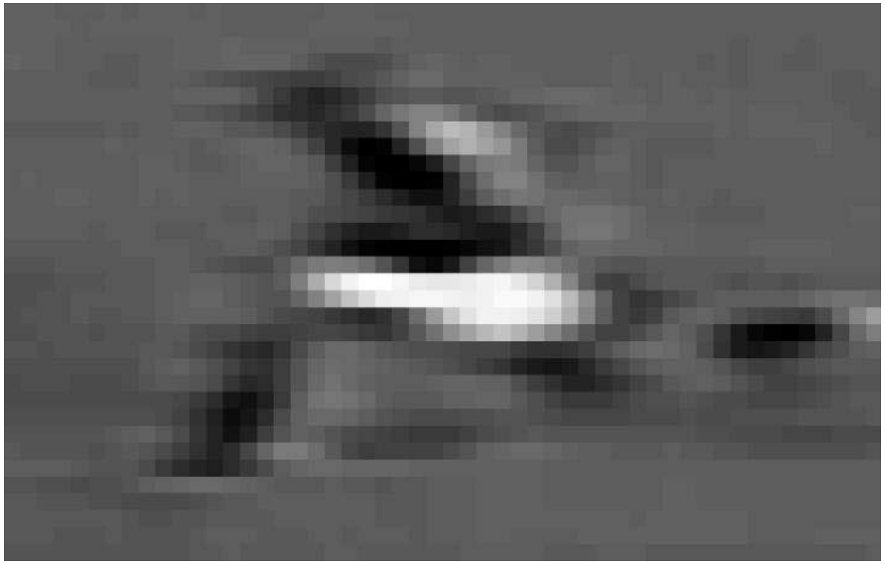




Correlation-based tracking
Extract person-centered frame window

[Efros, Berg, Mori, & Malik 2003]
http://graphics.cs.cmu.edu/people/efros/research/action/

Using optical flow: action recognition at a distance



Extract optical flow to describe the region's motion.

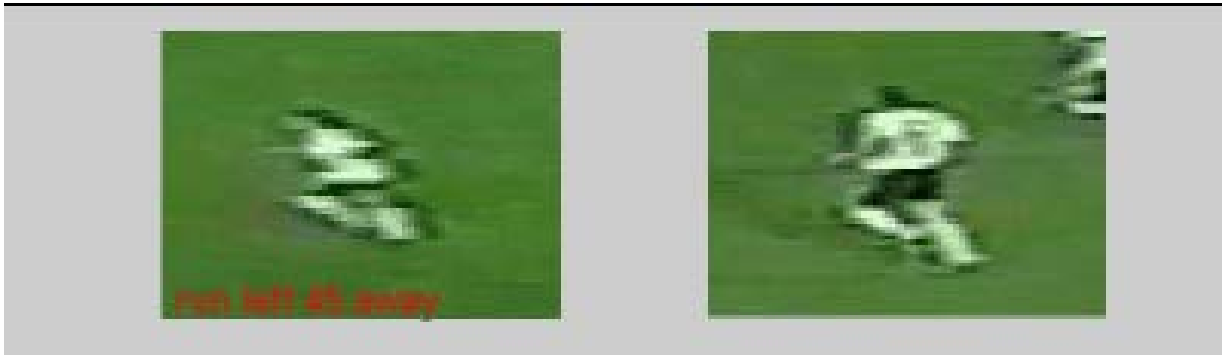
Using optical flow: action recognition at a distance

Input Sequence

Matched Frames

Use **nearest neighbor** classifier to name the actions occurring in new video frames.

Using optical flow: action recognition at a distance



Input Sequence

Matched NN Frame

Use **nearest neighbor** classifier to name the actions occurring in new video frames.

Using optical flow: 3D Structure from Motion

If you can estimate the motion (optical flow) between pairs of images, you could calculate your way back to the relative positions of features in 3D. Now take a bazillion images from flickr of the same object. Repeat the process, and show off your results to much amazement.

F. Schaffalitzky and A. Zisserman, Multi-view matching for unordered image sets, or "How do I organize my holiday snaps?", ECCV 02. Also check out http://phototour.cs.washington.edu/ or the Microsoft Live Labs Photosynth