Motion

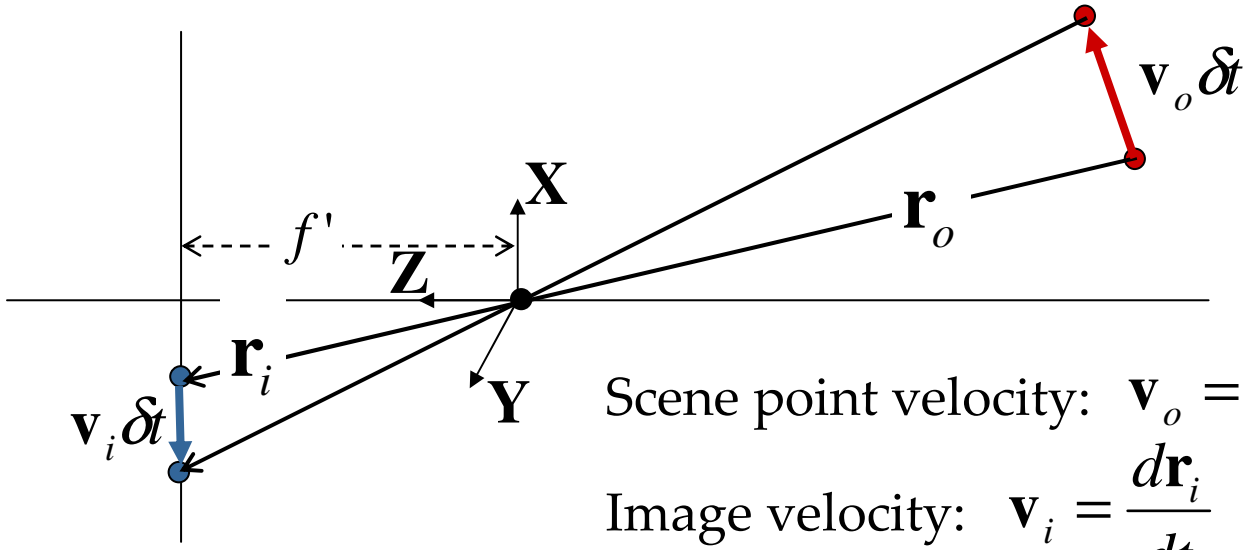
Reading: Robot Vision Chapter 12, Lucas-Kanade

Why estimate motion?

- Lots of uses
  - Track object behavior
  - Correct for camera jitter (stabilization)
  - Align images (mosaics)
  - 3D shape reconstruction
  - Special effects

Motion Field

- Image velocity of a point moving in the scene



Scene point velocity: $\mathbf{v}_o = \dfrac{d\mathbf{r}_o}{dt}$

Image velocity: $\mathbf{v}_i = \dfrac{d\mathbf{r}_i}{dt}$

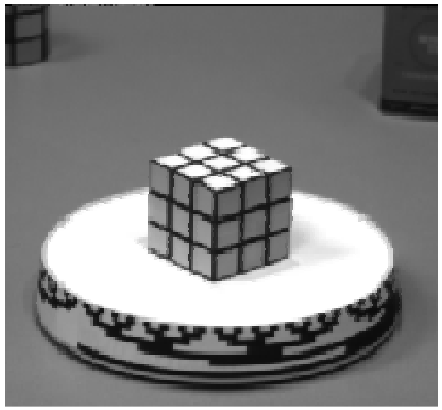
Perspective projection: $\dfrac{1}{f'}\mathbf{r}_i = \dfrac{\mathbf{r}_o}{\mathbf{r}_o \cdot \mathbf{Z}}$

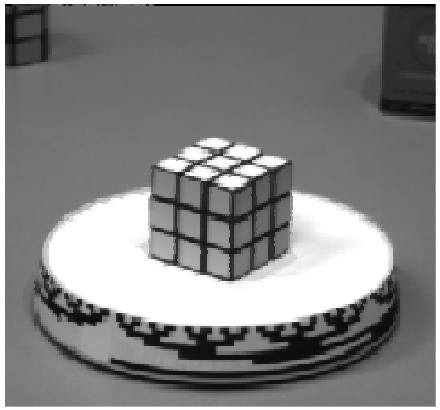
Motion field

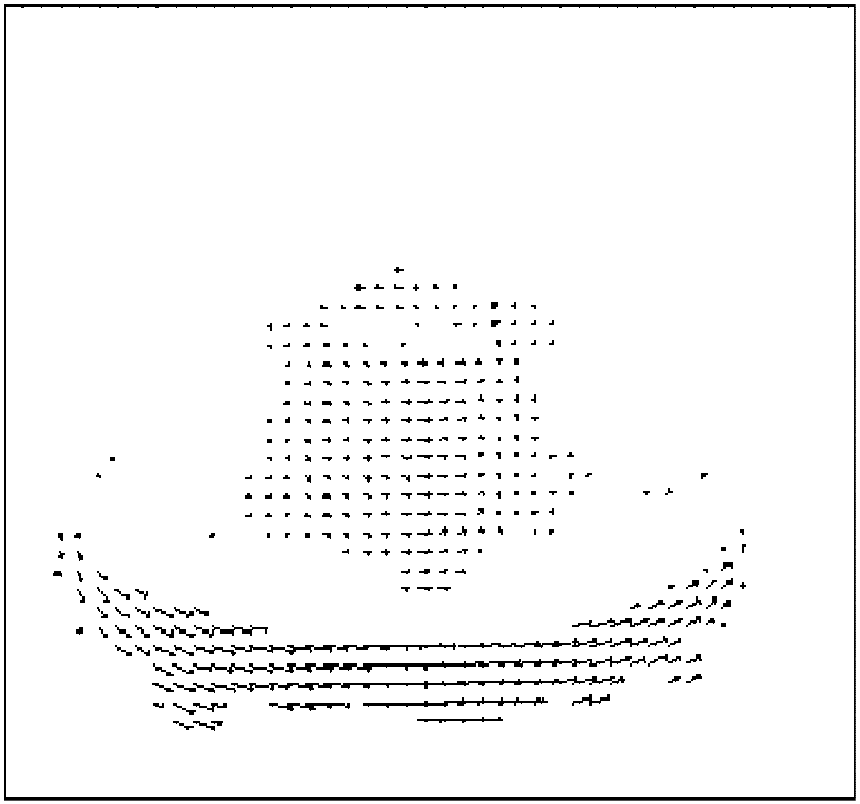
$$\mathbf{v}_i = \frac{d\mathbf{r}_i}{dt} = f' \frac{(\mathbf{r}_o \cdot \mathbf{Z})\mathbf{v}_o - (\mathbf{v}_o \cdot \mathbf{Z})\mathbf{r}_o}{(\mathbf{r}_o \cdot \mathbf{Z})^2} = f' \frac{(\mathbf{r}_o \times \mathbf{v}_o) \times \mathbf{Z}}{(\mathbf{r}_o \cdot \mathbf{Z})^2}$$

Optical Flow

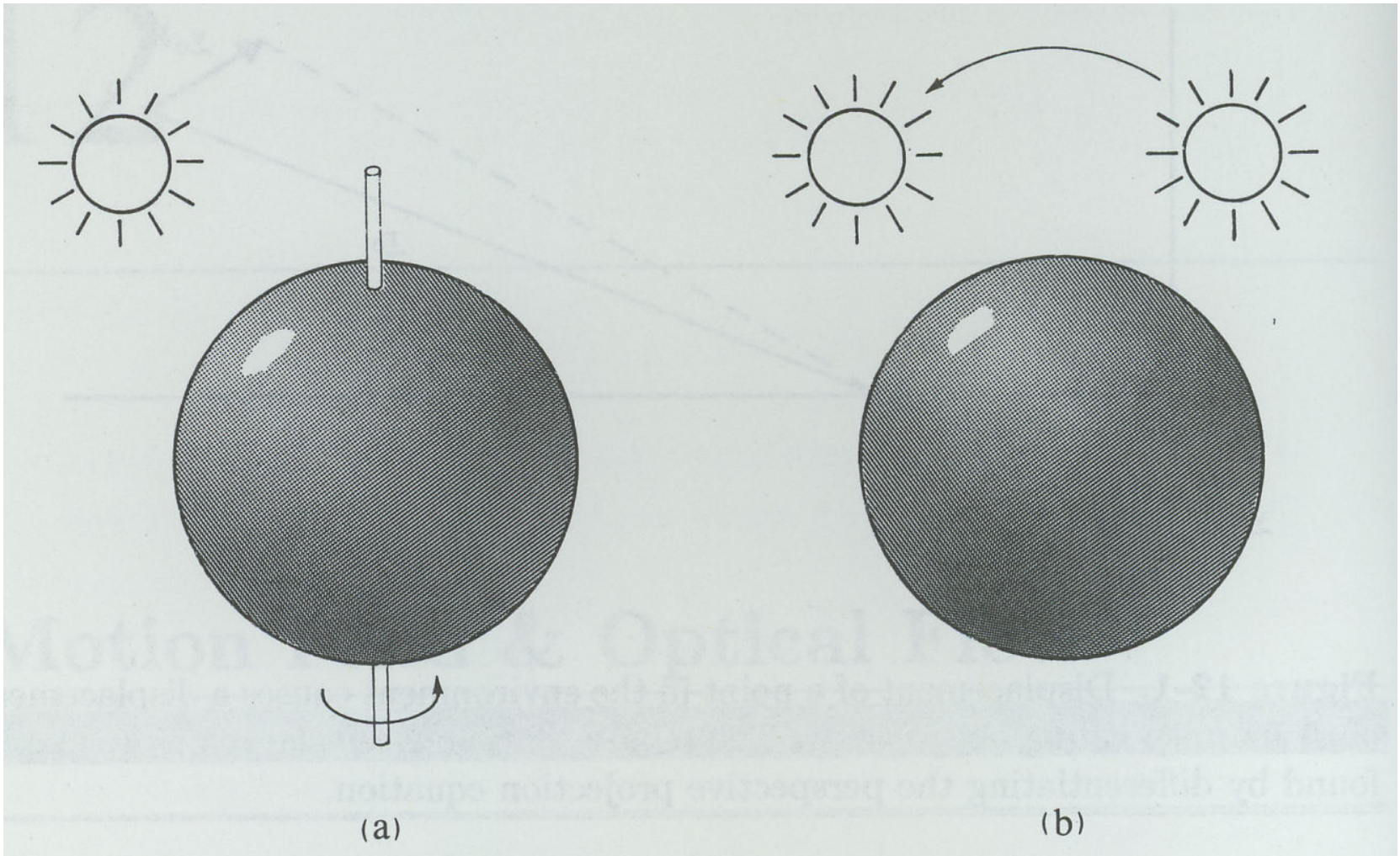
- Motion of brightness pattern in the image
- **Ideally** Optical flow = Motion field



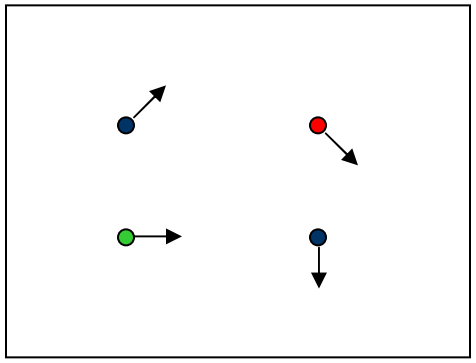




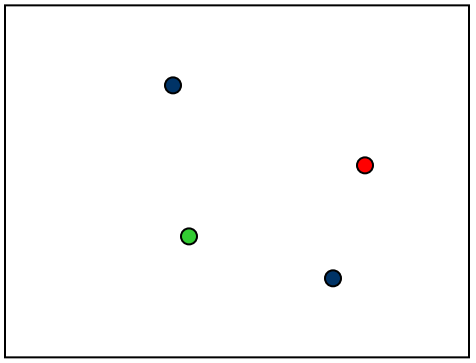
Optical Flow ≠ Motion Field



Problem Definition: Optical Flow



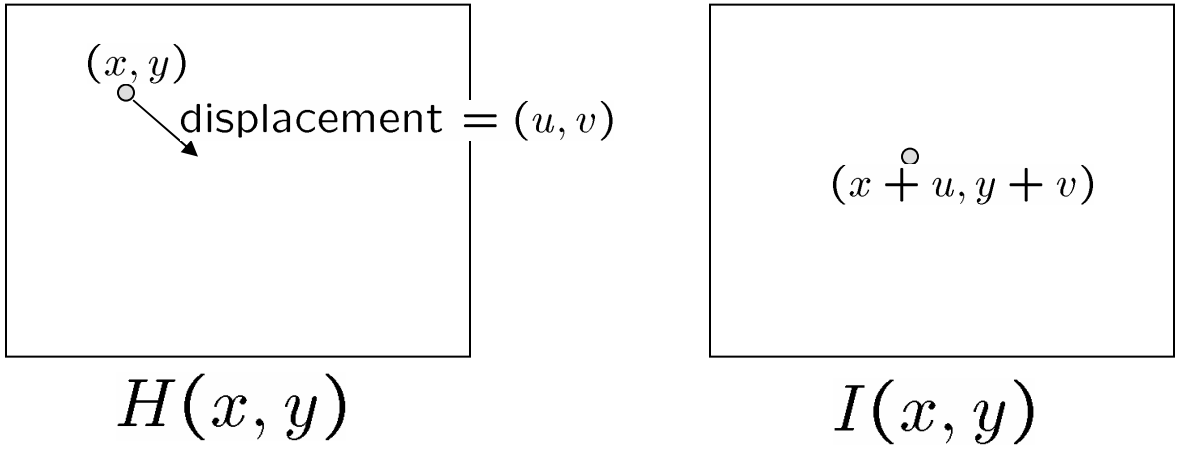
$H(x, y)$



$I(x, y)$

- 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

Optical Flow Constraints



- Let's look at these constraints more closely
  - brightness constancy: Q: what's the equation?

$$0 = I(x+u, y+v) - H(x, y)$$

  - small motion: (u and v are less than 1 pixel)

suppose we take the Taylor series expansion of I:

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

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

Optical Flow Equation

$$0 = I(x+u, y+v) - H(x,y)$$

Plugging in the Taylor expansion gives us

$$\approx I(x,y) + I_x u + I_y v - H(x,y) \quad \text{shorthand: } I_x = \frac{\partial I}{\partial x}$$

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

$$\approx I_t + I_x u + I_y v$$

$$\approx I_t + \nabla I \cdot [u \;\; v]$$

- In the limit as u and v go to zero, this becomes exact

$$0 = I_t + \nabla I \cdot [\frac{\partial x}{\partial t} \;\; \frac{\partial y}{\partial t}]$$

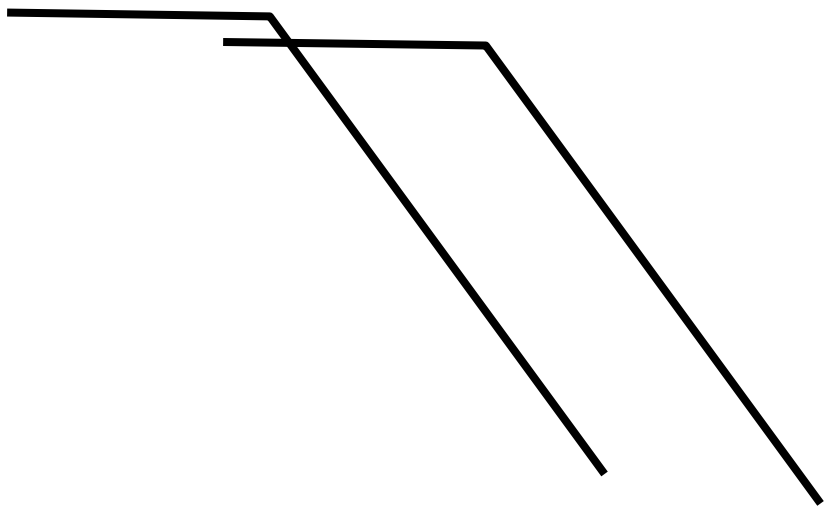
Optical Flow Equation

$$0 = I_t + \nabla I \cdot [u \ \ v]$$

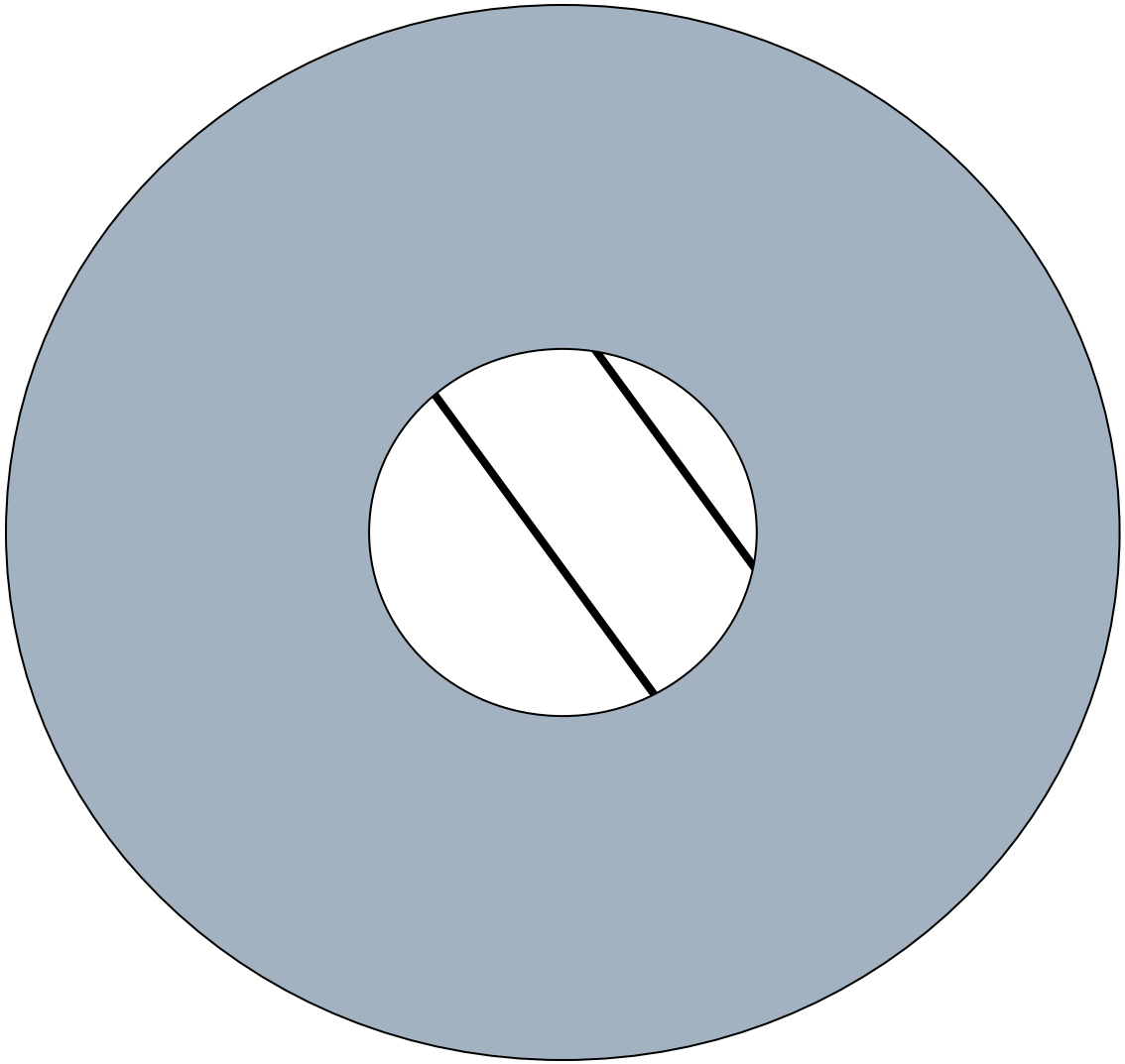
- Q: how many unknowns and equations per pixel?
- Intuitively, what does this constraint mean?
  - The component of the flow in the gradient direction is determined
  - The component of the flow parallel to an edge is unknown

barberpole illusion

Aperture Problem



Aperture Problem



Estimating Optical Flow

$$0 = I_t(\mathbf{p_i}) + \nabla I(\mathbf{p_i}) \cdot [u \ v]$$

- How to get more equations for a pixel?
  - Basic idea: impose additional constraints
    - most common is to assume that the flow field is smooth locally
    - one method: pretend the pixel's neighbors have the same (u,v)
      - If we use a 5x5 window, that gives us 25 equations per pixel!

$$\underset{A}{\begin{bmatrix} I_x(\mathbf{p_1}) & I_y(\mathbf{p_1}) \\ I_x(\mathbf{p_2}) & I_y(\mathbf{p_2}) \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}}) & I_y(\mathbf{p_{25}}) \end{bmatrix}}_{25\times2} \underset{d}{\begin{bmatrix} u \\ v \end{bmatrix}}_{2\times1} = - \underset{b}{\begin{bmatrix} I_t(\mathbf{p_1}) \\ I_t(\mathbf{p_2}) \\ \vdots \\ I_t(\mathbf{p_{25}}) \end{bmatrix}}_{25\times1}$$

RGB version

- How to get more equations for a pixel?
  - Basic idea: impose additional constraints
    - most common is to assume that the flow field is smooth locally $0 = I_t(\mathbf{p_i})[0,1,2] + \nabla I(\mathbf{p_i})[0,1,2] \cdot [u\ v]$
    - one method: pretend the pixel's neighbors have the same (u,v)
      - If we use a 5x5 window, that gives us 25*3 equations per pixel!

$$\begin{bmatrix} I_x(\mathbf{p_1})[0] & I_y(\mathbf{p_1})[0] \\ I_x(\mathbf{p_1})[1] & I_y(\mathbf{p_1})[1] \\ I_x(\mathbf{p_1})[2] & I_y(\mathbf{p_1})[2] \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}})[0] & I_y(\mathbf{p_{25}})[0] \\ I_x(\mathbf{p_{25}})[1] & I_y(\mathbf{p_{25}})[1] \\ I_x(\mathbf{p_{25}})[2] & I_y(\mathbf{p_{25}})[2] \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p_1})[0] \\ I_t(\mathbf{p_1})[1] \\ I_t(\mathbf{p_1})[2] \\ \vdots \\ I_t(\mathbf{p_{25}})[0] \\ I_t(\mathbf{p_{25}})[1] \\ I_t(\mathbf{p_{25}})[2] \end{bmatrix}$$

A (75x2) $\quad d$ (2x1) $\quad b$ (75x1)

Lucas-Kanade Flow

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

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

$A^T A$ $\qquad\qquad$ $A^T b$

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

Conditions for Solvability

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

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

$A^T A$ $\qquad$ $A^T b$

When is this solvable?

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

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

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

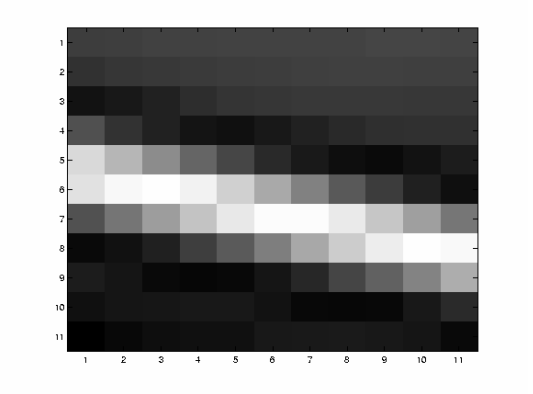
    $$\left(\sum \nabla I (\nabla I)^T\right) \nabla I = k \|\nabla I\|^2 \nabla I$$

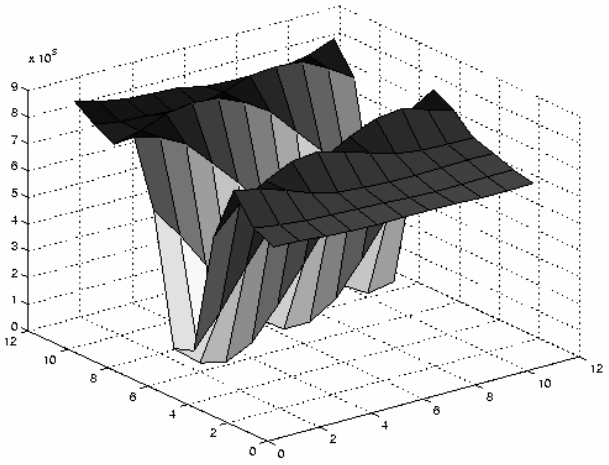
  - ∇I is an eigenvector with eigenvalue $k\|\nabla I\|^2$
  - What's the other eigenvector of A^TA?
    - let N be perpendicular to ∇I

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

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

Edge

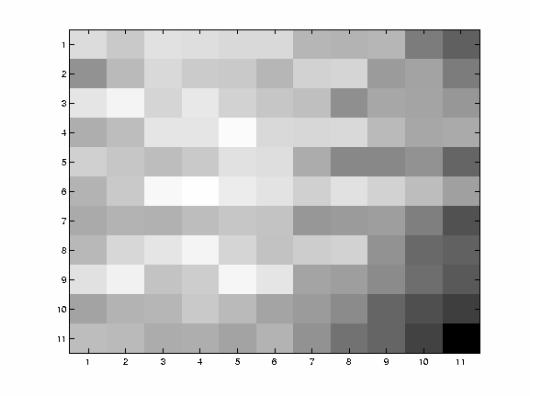


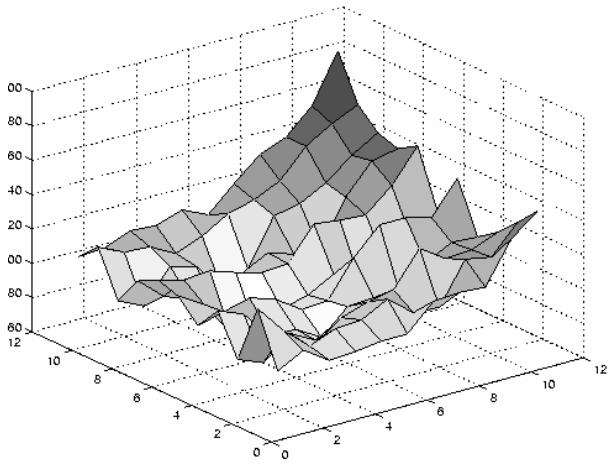


$$\sum \nabla I (\nabla I)^T$$

- large gradients, all the same
- large λ_1, small λ_2

Low Texture Region



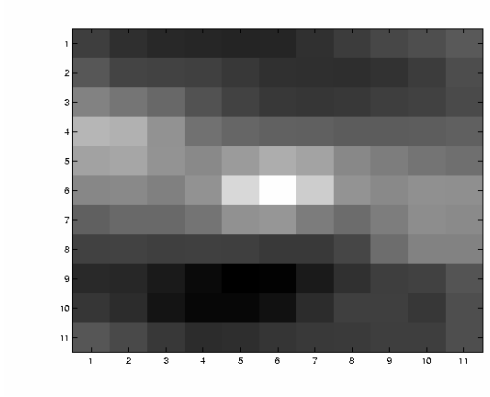


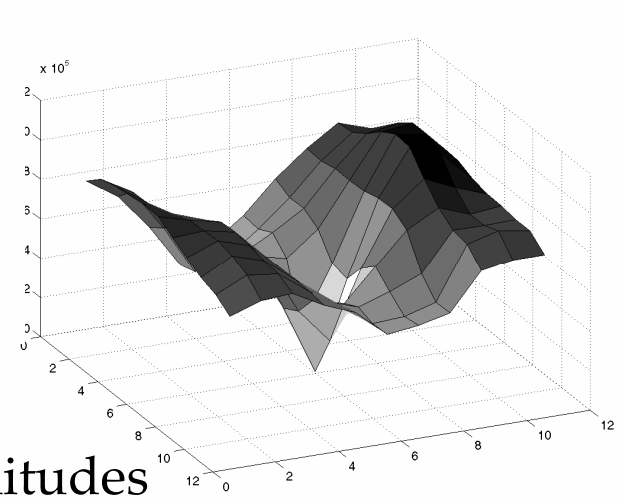
$$\sum \nabla I (\nabla I)^T$$

– 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

Observation

- This is a two image problem BUT
  - Can measure sensitivity by just looking at one of the images!
  - This tells us which pixels are easy to track, which are hard
    - very useful later on when we do feature tracking...

Read “A Combined Corner and Edge Detector” by C. Harris and M. Stephens (local copy on class page)

Errors in Lucas-Kanade

- What are the potential causes of errors in this procedure?
  - Suppose A^TA is easily invertible
  - Suppose there is not much noise in the image
- When our assumptions are violated
  - Brightness constancy is **not** satisfied
  - The motion is **not** small
  - A point does **not** move like its neighbors
    - window size is too large
    - what is the ideal window size?

Improving Accuracy

Recall our small motion assumption

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

- 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
    - Read first four pages of
      - http://www.library.cornell.edu/nr/bookcpdf/c9-4.pdf
  - Approach so far does one iteration of Newton's method
    - Better results are obtained via more iterations

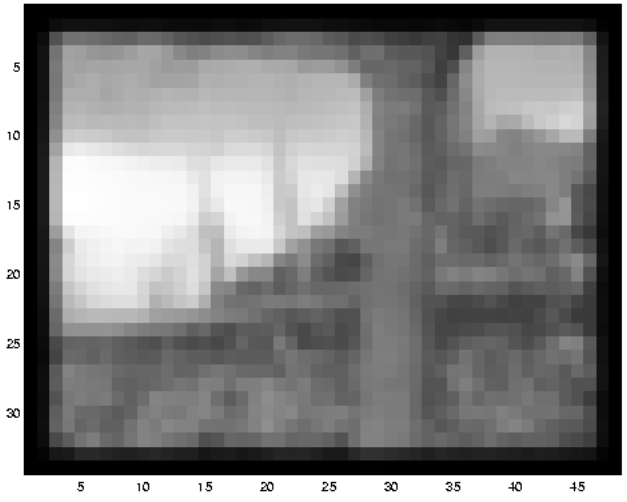
Iterative Refinement

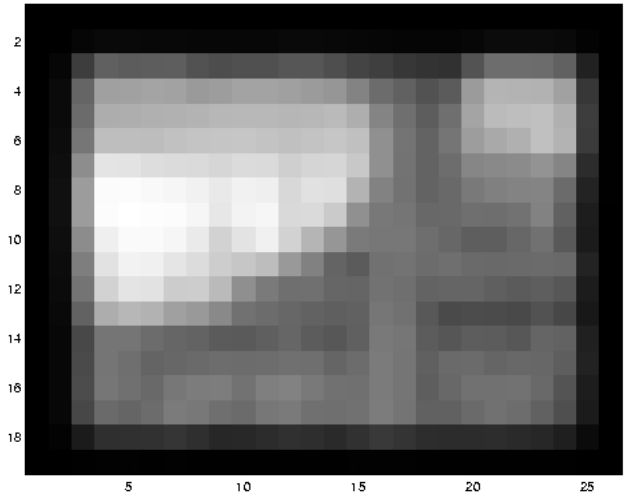
- Iterative Lucas-Kanade Algorithm
  1. Estimate velocity at each pixel by solving Lucas-Kanade equations
  2. Warp H towards I using the estimated flow field
     - *use image warping techniques*
  3. Repeat until convergence

Revisiting the Small Motion Assumption

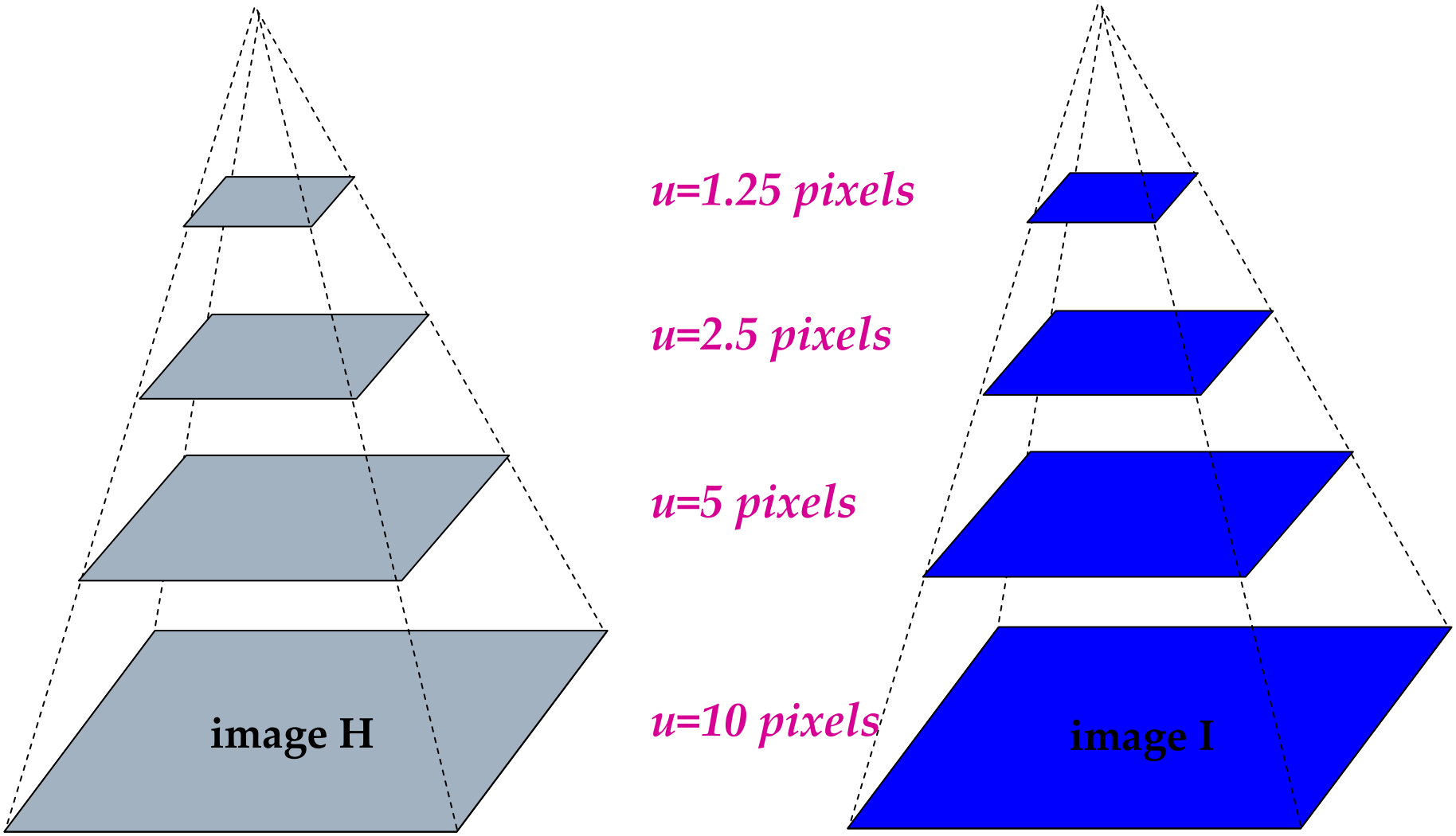
- Is this motion small enough?
  - Probably not—it's much larger than one pixel (2nd order terms dominate)
  - How might we solve this problem?

Reduce the Resolution!





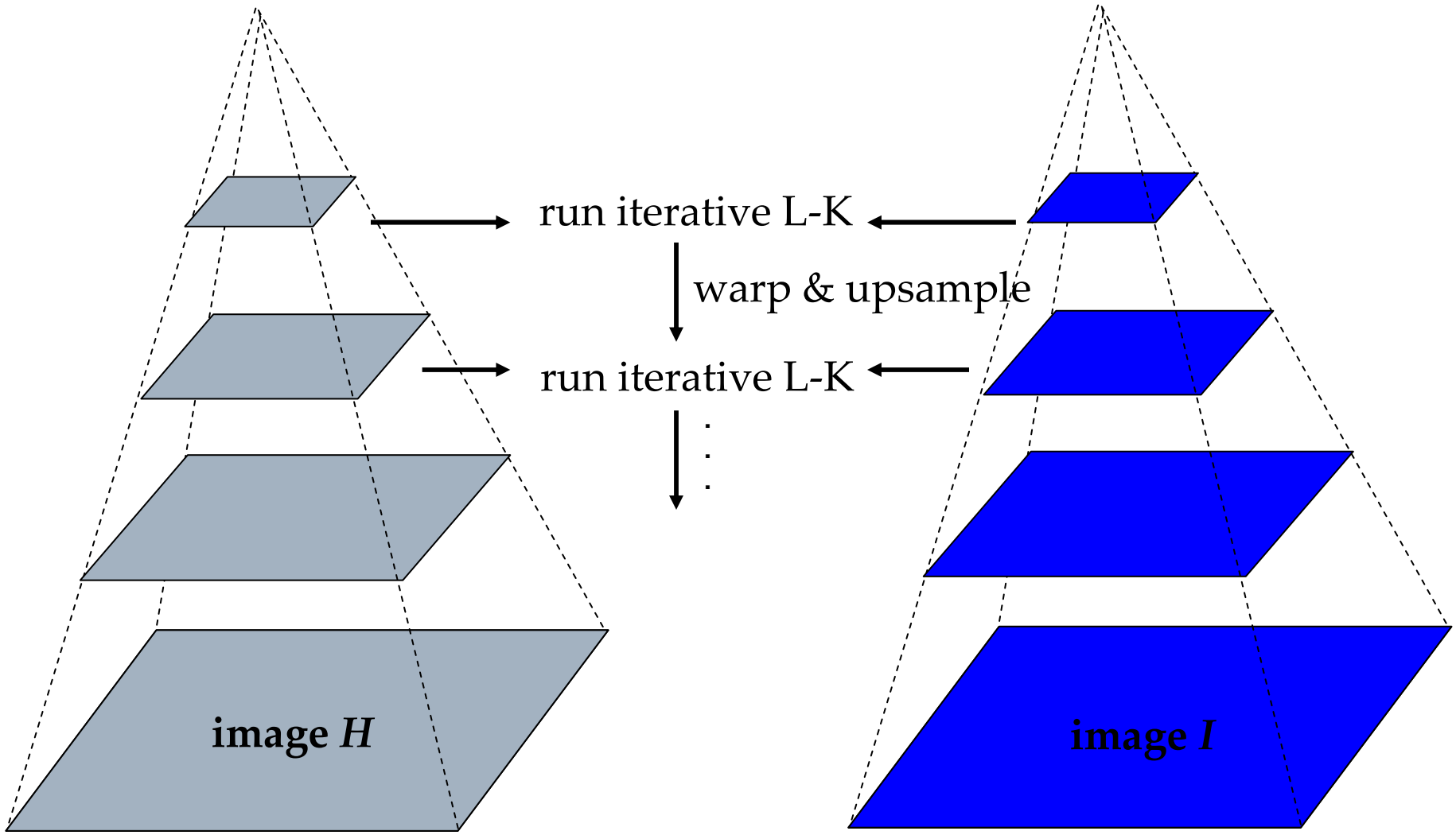
Coarse-to-fine Optical Flow Estimation



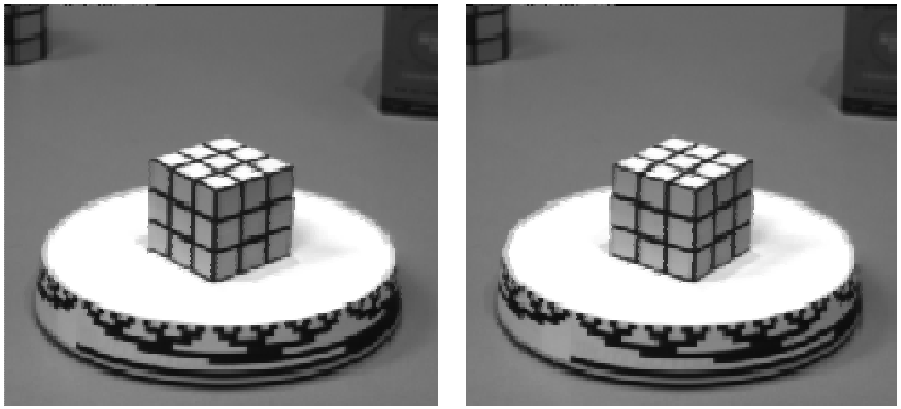
Gaussian pyramid of image *H*

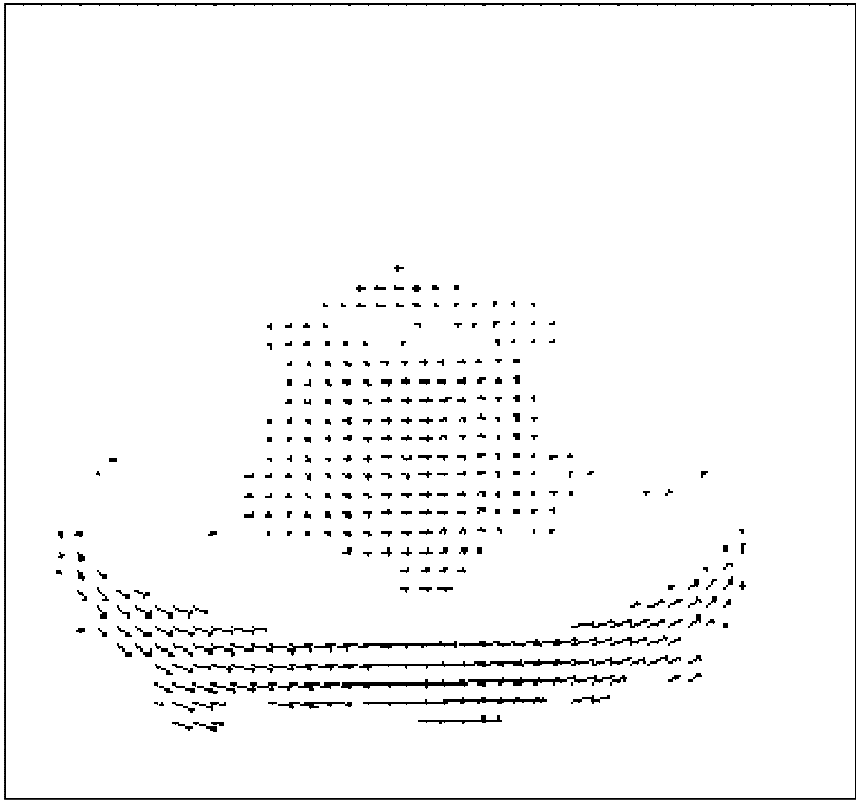
Gaussian pyramid of image *I*

Coarse-to-fine Optical Flow Estimation



Optical Flow Result





Motion Tracking

- Suppose we have more than two images
  - How to track a point through all of the images?
    - In principle, we could estimate motion between each pair of consecutive frames
    - Given point in first frame, follow arrows to trace out it's path
    - Problem: DRIFT
      - small errors will tend to grow and grow over time—the point will drift way off course
- Feature Tracking
  - Choose only the points ("features") that are easily tracked
  - How to find these features?
    - windows where $\sum \nabla I (\nabla I)^T$ as two large eigenvalues
  - Called the Harris Corner Detector

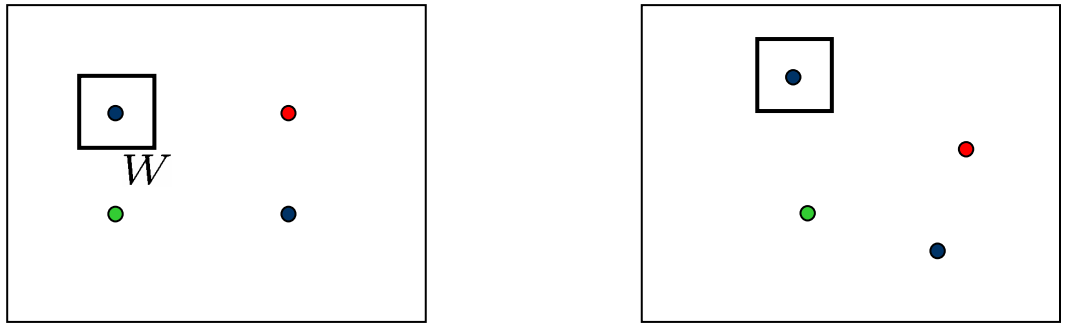
Feature Detection

Tracking Features

- Feature tracking
  - Compute optical flow for that feature for each consecutive *H, I*
- When will this go wrong?
  - Occlusions—feature may disappear
    - need mechanism for deleting, adding new features
  - Changes in shape, orientation
    - allow the feature to deform
  - Changes in color
  - Large motions
    - will pyramid techniques work for feature tracking?

Handling Large Motions

- L-K requires small motion
  - If the motion is much more than a pixel, use discrete **search** instead



$H(x, y)$ $I(x, y)$

  - Given feature window *W* in *H*, find best matching window in *I*
  - Minimize sum squared difference (SSD) of pixels in window

$$min_{(u,v)} \left\{ \sum_{(x,y)\in W} |I(x+u, y+v) - H(x, y)|^2 \right\}$$

  - Solve by doing a search over a specified range of *(u,v)* values
    - this *(u,v)* range defines the **search window**

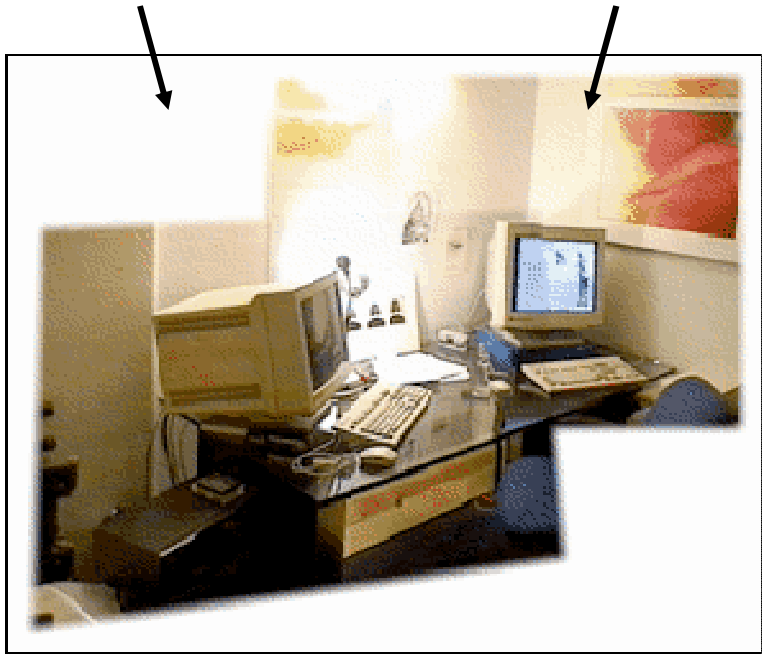
Tracking Over Many Frames

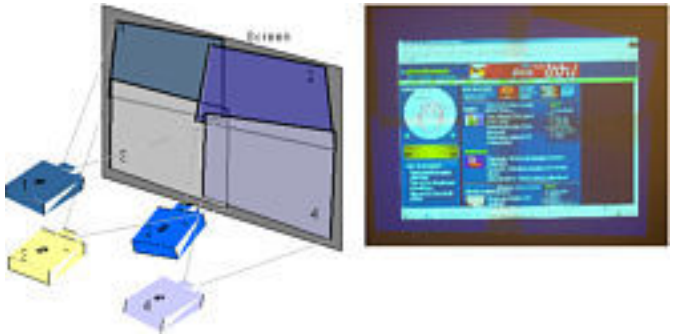
- Feature tracking with *m* frames
  1. Select features in first frame
  2. Given feature in frame *i*, compute position in *i*+1
  3. Select more features if needed
  4. $i = i + 1$
  5. If $i < m$, go to step 2

Issues

- Discrete search vs. Lucas Kanade?
  - depends on expected magnitude of motion
  - discrete search is more flexible
- Compare feature in frame *i* to *i+1* or frame *1* to *i+1*?
  - affects tendency to drift..
- How big should search window be?
  - too small: lost features. Too large: slow

Image Alignment





- Goal: estimate single *(u,v)* translation for entire image
  - Easier subcase: solvable by pyramid-based Lucas-Kanade

Next Week

- Midterm and Voting
  - Closed book
  - Pay attention to the key equations and their derivations