Motion and Optical Flow

Moving to Multiple Images

- So far, we've mostly looked at processing a single image
- Multiple images
  - Multiple cameras at one time: stereo
  - Single camera at many times: video
    - Moving camera
    - Moving objects
    - Changing environment (e.g., lighting)
  - (Multiple cameras at multiple times)

Applications of Multiple Images

- 2D
  - Feature / object tracking
  - Segmentation based on motion
  - Image fusion (extending field of view, dynamic range, other parameters)
- 3D
  - Shape extraction
  - Motion capture

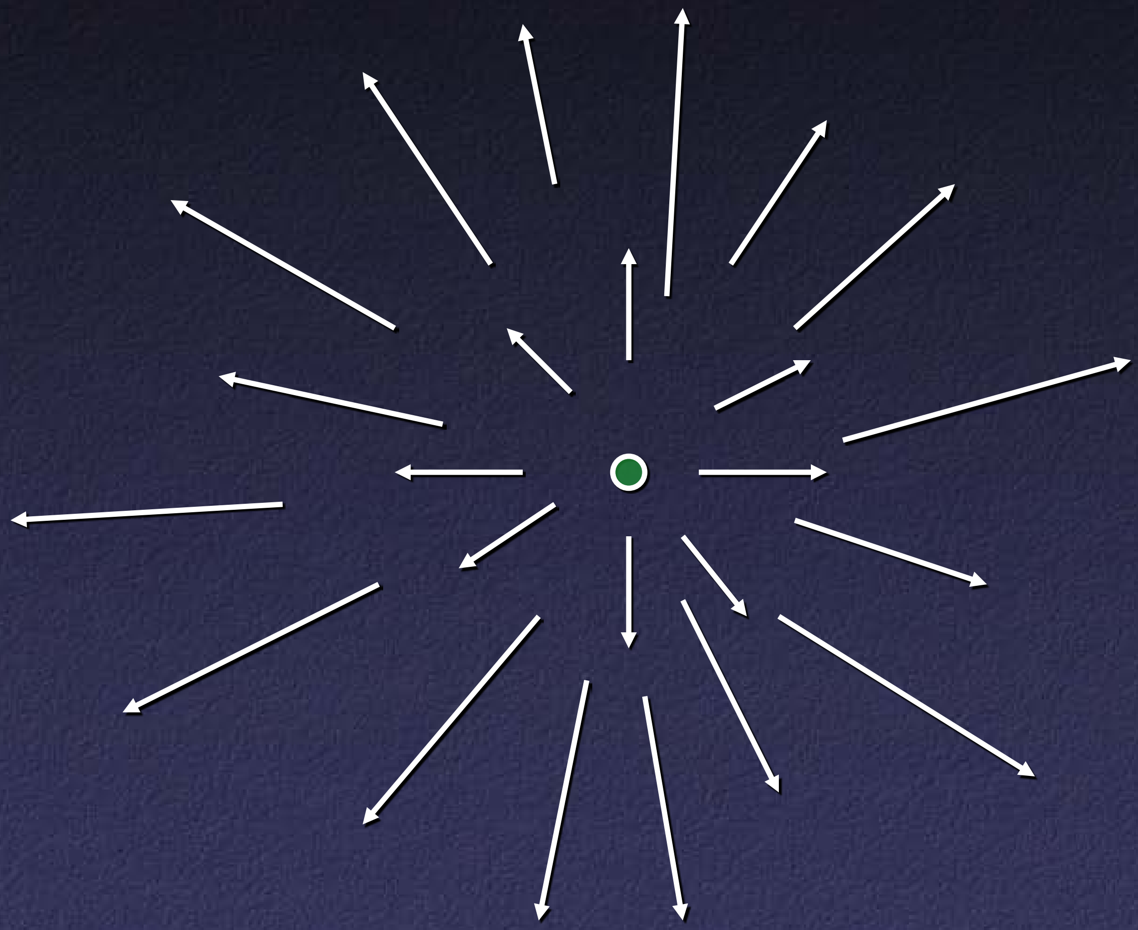
Applications of Multiple Images in Graphics

- Stitching images into panoramas
- Automatic image morphing
- Reconstruction of 3D models for rendering
- Capturing articulated motion for animation

Applications of Multiple Images in Biological Systems

- Shape inference
- Peripheral sensitivity to motion (low-level)
- Looming field – obstacle avoidance
- Very similar applications in robotics

Looming Field



- Pure translation: motion looks like it originates at a point – focus of expansion

Key Problem

- Main problem in most multiple-image methods: correspondence

Correspondence

- Small displacements
  - Differential algorithms
  - Based on gradients in space and time
  - Dense correspondence estimates
  - Most common with video
- Large displacements
  - Matching algorithms
  - Based on correlation or features
  - Sparse correspondence estimates
  - Most common with multiple cameras / stereo

Result of Correspondence

- For points in image *i*, displacements to corresponding locations in image *j*
- In video, usually called motion field
- In stereo, usually called disparity

Computing Motion Field

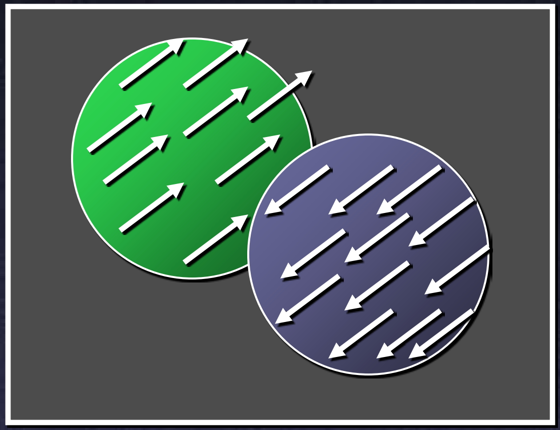
- Basic idea: a small portion of the image ("local neighborhood") shifts position
- Assumptions
  - No / small changes in reflected light
  - No / small changes in scale
  - No occlusion or disocclusion
  - Neighborhood is correct size: aperture problem

Actual and Apparent Motion

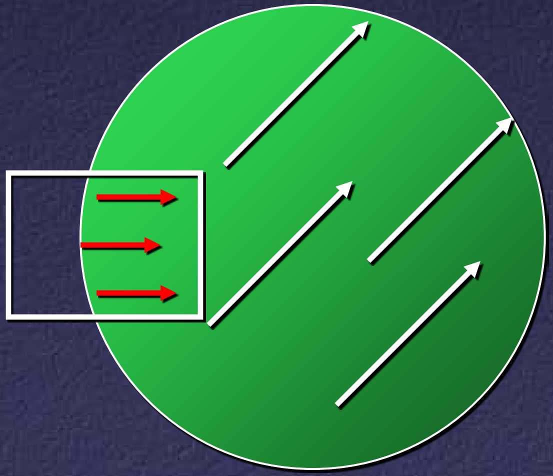
- If these assumptions violated, can still use the same methods – apparent motion
- Result of algorithm is optical flow (vs. ideal motion field)
- Most obvious effects:
  - Aperture problem: can only get motion perpendicular to edges
  - Errors near discontinuities (occlusions)

Aperture Problem

- Too big:
  confused by
  multiple motions



- Too small:
  only get motion
  perpendicular
  to edge



Computing Optical Flow: Preliminaries

- Image sequence I(x,y,t)
- Uniform discretization along x,y,t – "cube" of data
- Differential framework: compute partial derivatives along *x,y,t* by convolving with derivative of Gaussian

Computing Optical Flow: Image Brightness Constancy

- Basic idea: a small portion of the image ("local neighborhood") shifts position
- Brightness constancy assumption:

$$\frac{dI}{dt} = 0$$

Computing Optical Flow: Image Brightness Constancy

- This does not say that a position in the image remains the same brightness!
- $\frac{dI}{dt}$ vs. $\frac{\partial I}{\partial t}$: total vs. partial derivative
- Use chain rule

$$\frac{dI\left(x(t), y(t), t\right)}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t}$$

Computing Optical Flow: Image Brightness Constancy

- Given optical flow **v**(x,y)

$$\frac{dI\big(x(t), y(t), t\big)}{dt} = 0$$

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

$$(\nabla I)^{\mathrm{T}}\mathbf{v} + I_t = 0$$

Image brightness constancy equation

Computing Optical Flow: Discretization

- Look at some neighborhood N:

$$\forall_{(i,j)\in \mathrm{N}} \left(\nabla I(i,j)\right)^{\mathrm{T}} \mathbf{v} + I_t(i,j) \overset{\text{want}}{=} 0$$

$$\mathbf{A}\mathbf{v} + \mathbf{b} \overset{\text{want}}{=} 0$$

$$\mathbf{A} = \begin{bmatrix} \nabla I(i_1, j_1) \\ \nabla I(i_2, j_2) \\ \vdots \\ \nabla I(i_n, j_n) \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} I_t(i_1, j_1) \\ I_t(i_2, j_2) \\ \vdots \\ I_t(i_n, j_n) \end{bmatrix}$$

Computing Optical Flow: Least Squares

- In general, overconstrained linear system
- Solve by least squares

$$\mathbf{A}\mathbf{v} + \mathbf{b} \overset{\text{want}}{=} 0$$

$$\Rightarrow (\mathbf{A}^{\mathrm{T}}\mathbf{A})\,\mathbf{v} = -\mathbf{A}^{\mathrm{T}}\mathbf{b}$$

$$\mathbf{v} = -(\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}$$

Computing Optical Flow: Stability

- Has a solution unless $\mathbf{C} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$ is singular

$$\mathbf{C} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$$

$$\mathbf{C} = \begin{bmatrix} \nabla I(i_1, j_1) & \nabla I(i_2, j_2) & \cdots & \nabla I(i_n, j_n) \end{bmatrix} \begin{bmatrix} \nabla I(i_1, j_1) \\ \nabla I(i_2, j_2) \\ \vdots \\ \nabla I(i_n, j_n) \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} \sum_N I_x{}^2 & \sum_N I_x I_y \\ \sum_N I_x I_y & \sum_N I_y{}^2 \end{bmatrix}$$

Computing Optical Flow: Stability

- Where have we encountered **C** before?
- Corner detector!
- **C** is singular if constant intensity or edge
- Use eigenvalues of **C**:
  - to evaluate stability of optical flow computation
  - to find good places to compute optical flow (corners!)
  - [Shi-Tomasi]

Computing Optical Flow: Improvements

- Assumption that optical flow is constant over neighborhood not always good
- Decreasing size of neighborhood $\Rightarrow$ **C** more likely to be singular
- Alternative: weighted least-squares
  - Points near center = higher weight
  - Still use larger neighborhood

Computing Optical Flow: Weighted Least Squares

- Let **W** be a diagonal matrix of weights

$$\mathbf{A} \rightarrow \mathbf{WA}$$

$$\mathbf{b} \rightarrow \mathbf{Wb}$$

$$\mathbf{v} = -(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$$

$$\Rightarrow \mathbf{v}_w = -(\mathbf{A}^T\mathbf{W}^2\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}^2\mathbf{b}$$

Computing Optical Flow: Improvements

- What if windows are still bigger?
- Adjust motion model: no longer constant within a window
- Popular choice: affine model

Computing Optical Flow: Affine Motion Model

- Translational model

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- Affine model

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

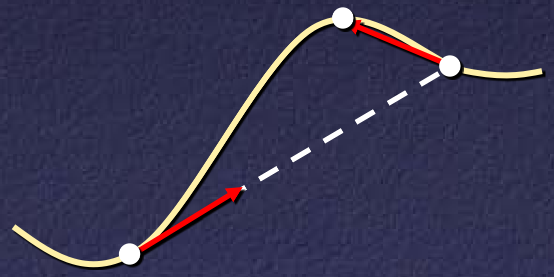
- Solved as before, but 6 unknowns instead of 2

Computing Optical Flow: Improvements

- Larger motion: how to maintain "differential" approximation?
- Solution: iterate
- Even better: adjust window / smoothing
  - Early iterations: use larger Gaussians to allow more motion
  - Late iterations: use less blur to find exact solution, lock on to high-frequency detail

Iteration

- Local refinement of optical flow estimate
- Sort of equivalent to multiple iterations of Newton's method



Computing Optical Flow: Lucas-Kanade

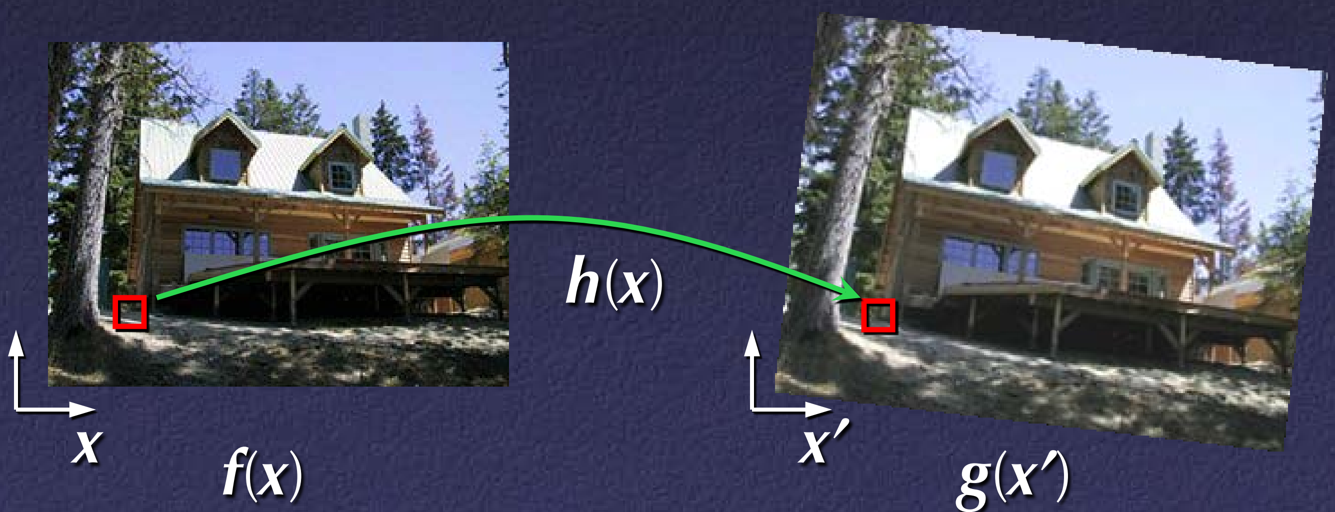
- Iterative algorithm:
  1. Set $\sigma =$ large (e.g. 3 pixels)
  2. Set $I' \leftarrow I_1$
  3. Set $\mathbf{v} \leftarrow 0$
  4. Repeat while $\mathrm{SSD}(I', I_2) > \tau$
     1. $\mathbf{v} \mathrel{+}= \text{Optical flow}(I' \rightarrow I_2)$
     2. $I' \leftarrow \mathrm{Warp}(I_1, \mathbf{v})$
  5. After n iterations, set $\sigma =$ small (e.g. 1.5 pixels)

Computing Optical Flow: Lucas-Kanade

- I' always holds warped version of I_1
  - Best estimate of I_2
- Gradually reduce thresholds
- Stop when difference between I' and I_2 small
  - Simplest difference metric = sum of squared differences (SSD) between pixels

Image Warping

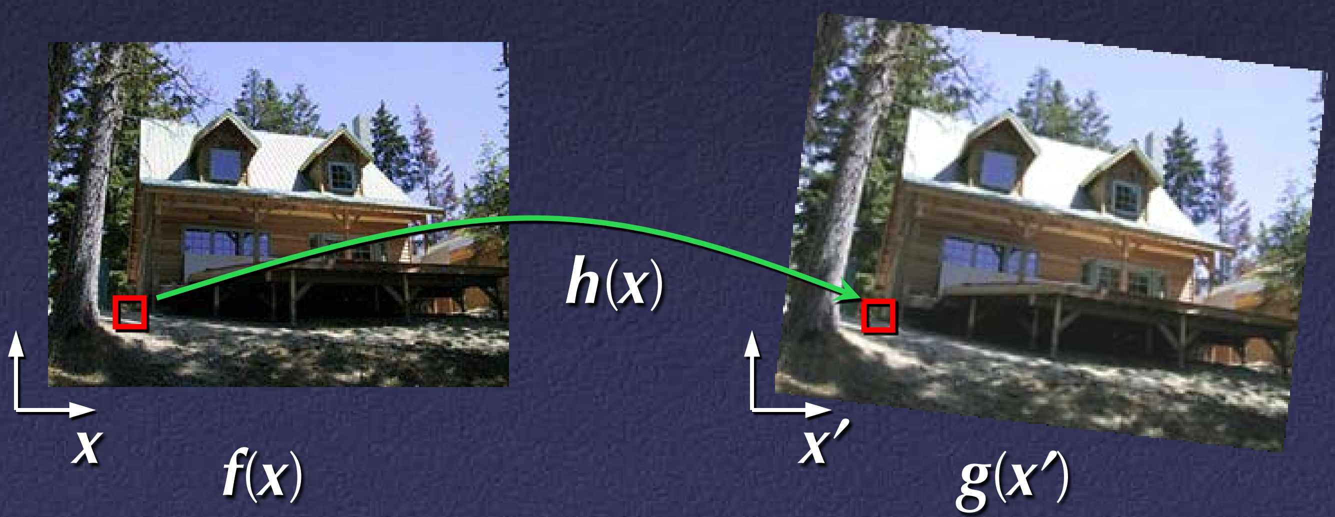
- Given a coordinate transform $\mathbf{x}' = \mathbf{h}(\mathbf{x})$ and a source image $\boldsymbol{f}(\mathbf{x})$, how do we compute a transformed image $\boldsymbol{g}(\mathbf{x}') = \boldsymbol{f}(\mathbf{h}(\mathbf{x}))$?



Szeliski

Forward Warping

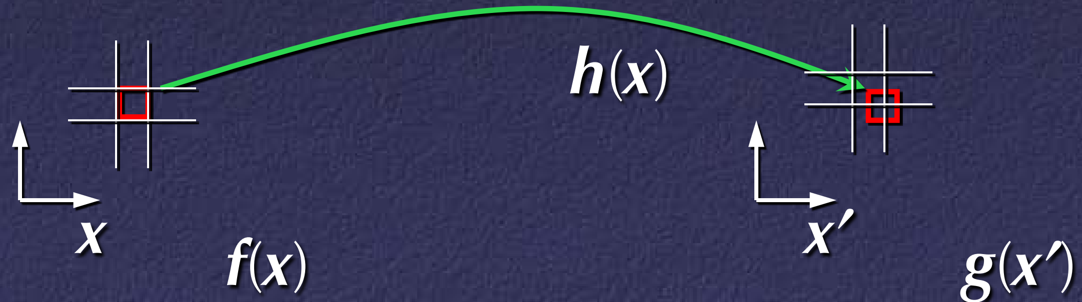
- Send each pixel f(x) to its corresponding location x′ = h(x) in g(x′)
- What if pixel lands "between" two pixels?



Szeliski

Forward Warping

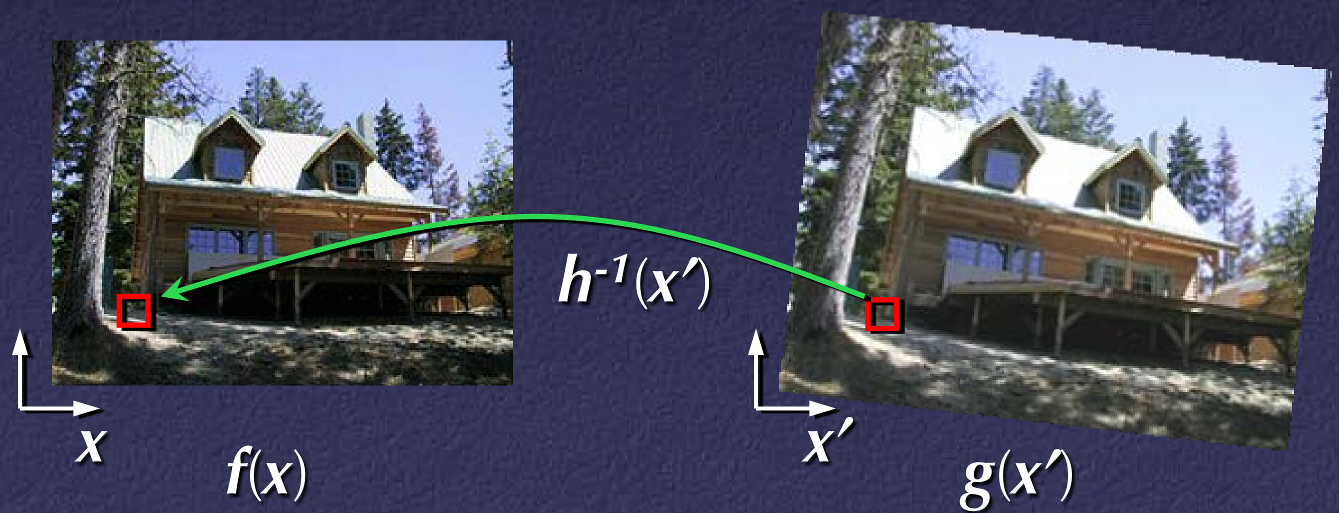
- Send each pixel f(x) to its corresponding location x′ = h(x) in g(x′)
- What if pixel lands "between" two pixels?
- Answer: add "contribution" to several pixels, normalize later (splatting)



Szeliski

Inverse Warping

- Get each pixel g(x′) from its corresponding location x = h-1(x′) in f(x)
- What if pixel comes from “between” two pixels?



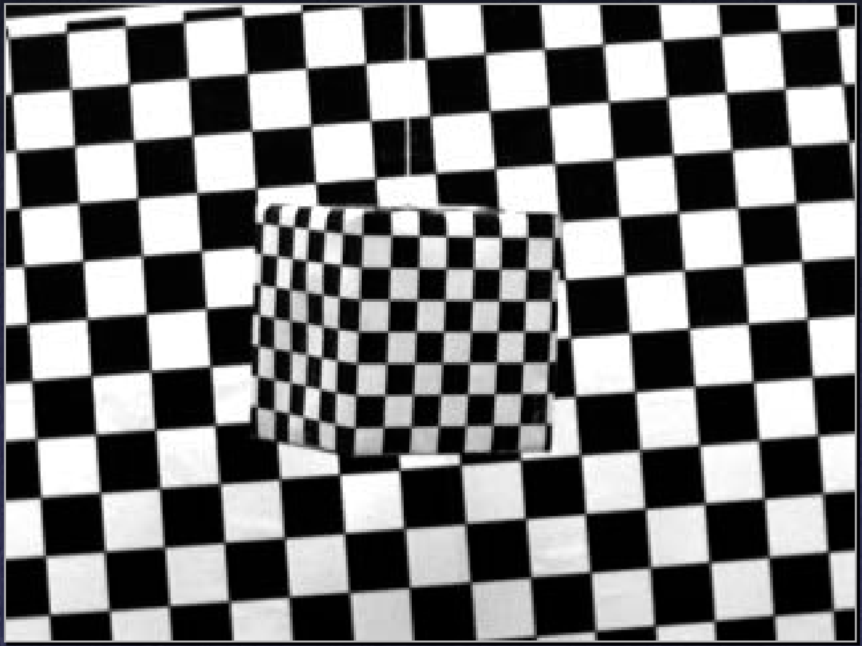
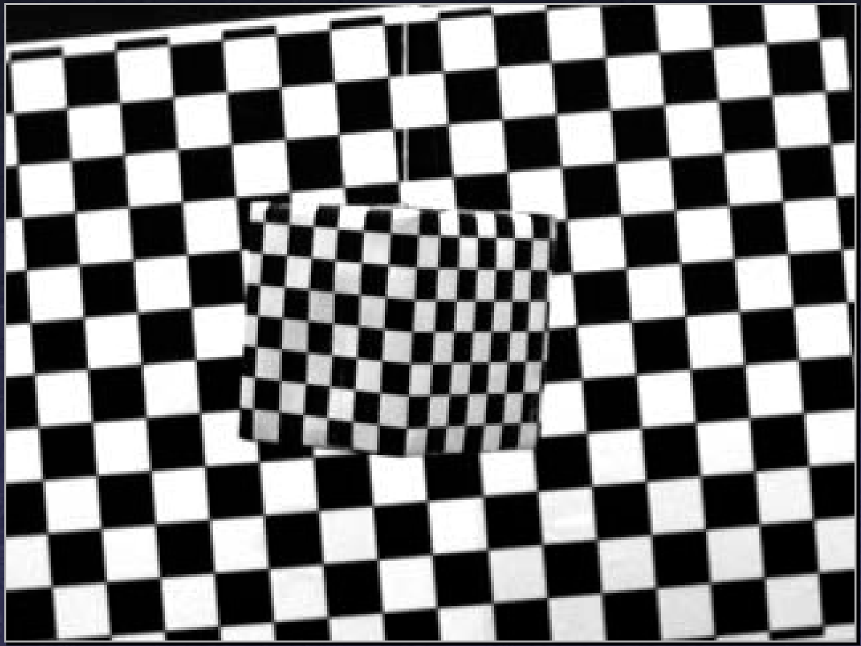
Szeliski

Inverse Warping

- Get each pixel $g(x')$ from its corresponding location $x = h\text{-}1(x')$ in $f(x)$
- What if pixel comes from "between" two pixels?
- Answer: resample color value from interpolated (prefiltered) source image

Szeliski

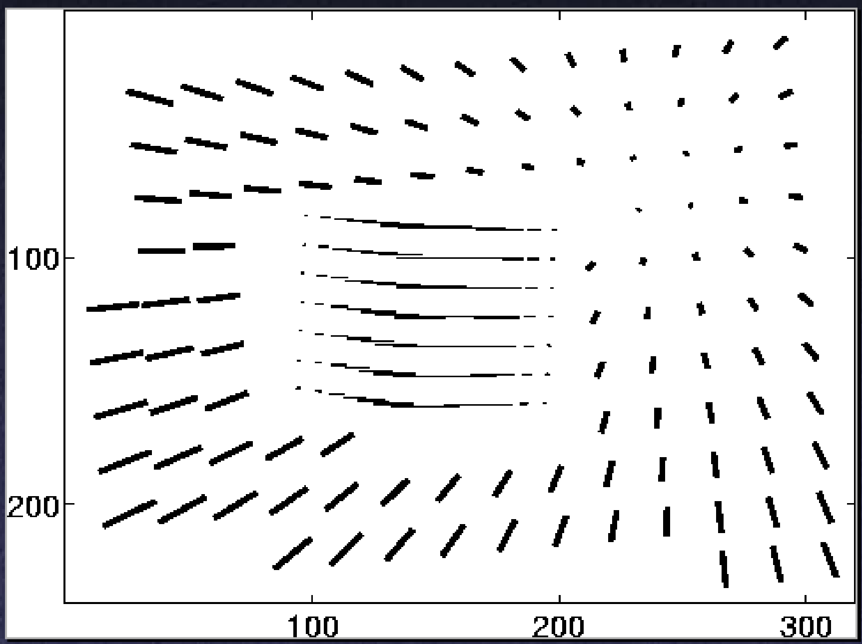
Optical Flow Applications

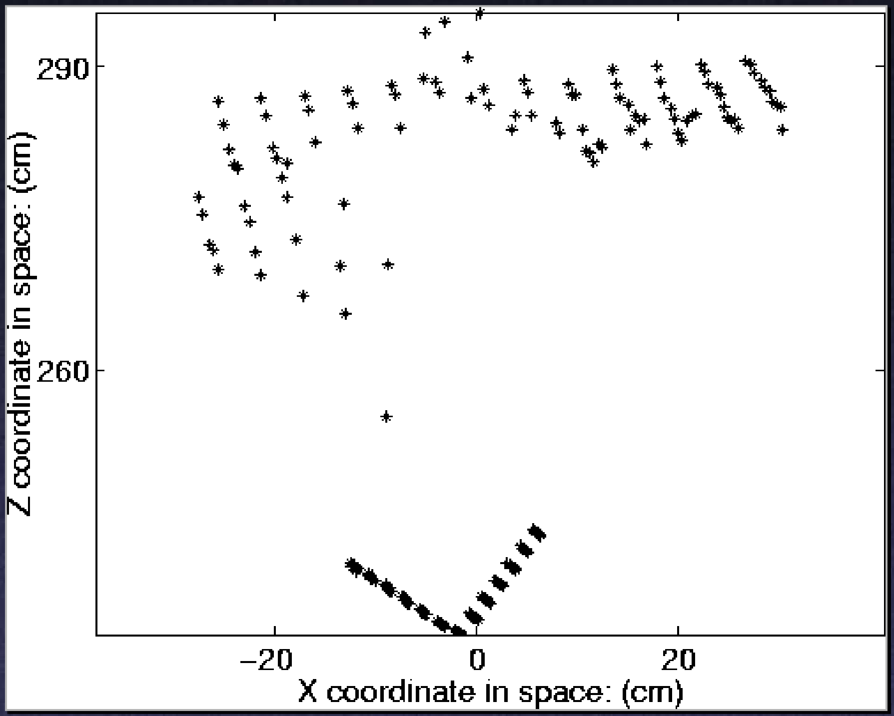
Video Frames

[Feng & Perona]

Optical Flow Applications



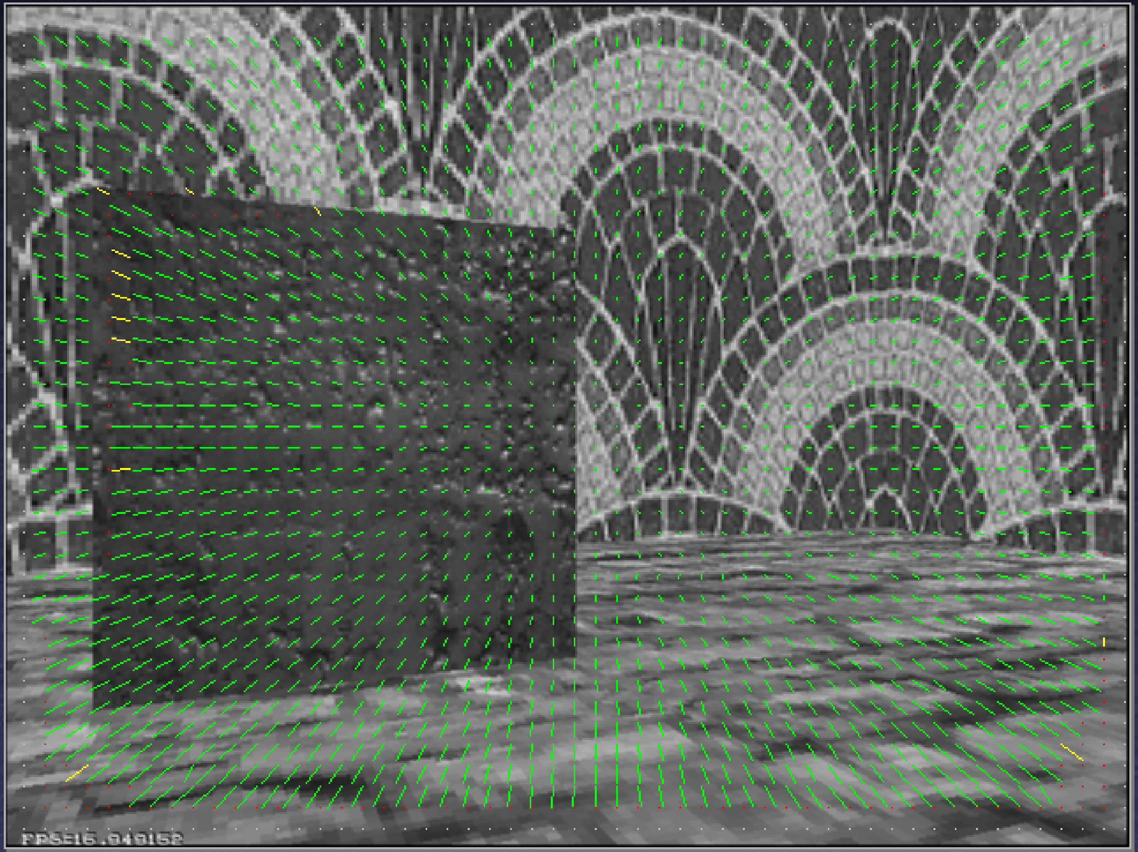
Optical Flow



Depth Reconstruction

[Feng & Perona]

Optical Flow Applications



Obstacle Detection: Unbalanced Optical Flow

[Temizer]

Optical Flow Applications

- Collision avoidance: keep optical flow balanced between sides of image

[Temizer]