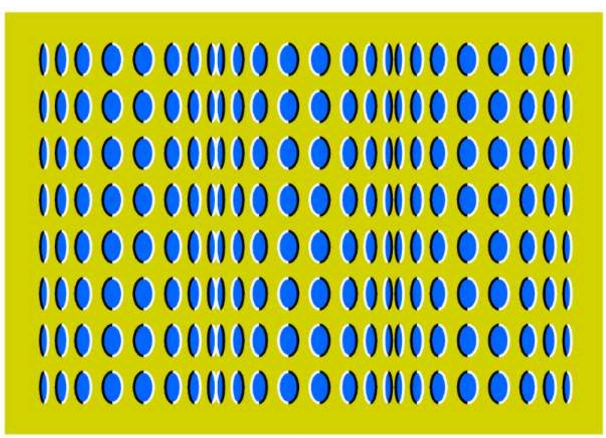
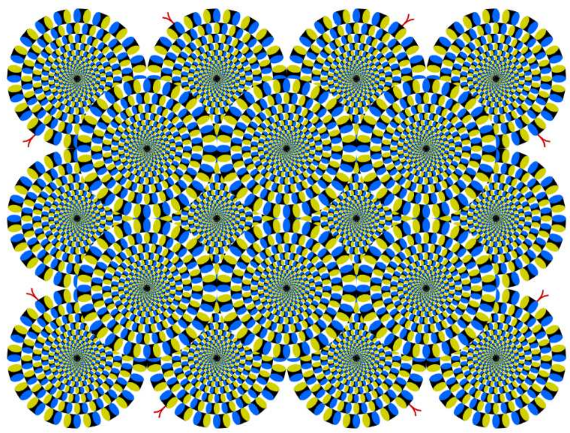
Motion Estimation

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

Motion Illusion created by Akiyoshi Kitaoka



Motion Illusion created by Akiyoshi Kitaoka



ANALYSIS of VISUAL MOTION

- GOAL: RECOVER RELATIVE MOTION BETWEEN OBSERVER AND ENVIRONMENT. I.E., INFER MOTION of 3D OBJECTS FROM SEQUENCE of 2D IMAGES
- GIVEN: $I(x, y, t)$
- MEASUREMENT PROBLEM:
  COMPUTE INSTANTANEOUS VELOCITY, (u, v), OF EVERY POINT AT EVERY TIME. I.E., RECOVER $V(x,y,t) = (u(x,y,t), v(x,y,t))$
  $= (dx/dt, dy/dt)$
- INTERPRETATION PROBLEM:
  USE VELOCITY INFO TO INFER 3D STRUCTURE, HIGH-LEVEL MOTION DESCRIPTION SUCH AS BEHAVIOR RECOGNITION

DIMENSIONS of COMPLEXITY

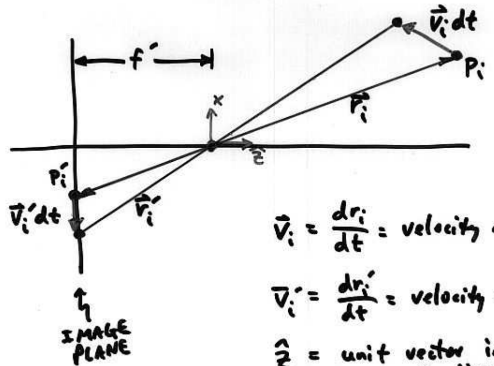
- OBJECT TYPE
  - POINTS
  - RIGID
  - NON-RIGID
- NUMBER of MOVING OBJECTS
- TYPES of OBJECT MOTION
  - 2D or 3D TRANSLATION
  - 2D or 3D ROTATION
  - ARTICULATED
- TYPES of CAMERA MOTION
  - STATIONARY
  - TRANSLATION ⊥ OPTICAL AXIS
  - GENERAL TRANSLATION
  - ROTATION

HUMAN VISUAL SYSTEM

- 2 SEPARATE MOTION PROCESSES LIKELY AT WORK (BRADDICK):
  - SHORT-RANGE
    * APPROX. CONTINUOUS MOTION
    * SPATIAL DISPLACEMENTS < 15' of arc
    * TEMPORAL DISPLACEMENTS < 100 msec
  - LONG-RANGE
    * LARGE SPATIAL AND/OR TEMPORAL DISPLACEMENTS
    * FEATURE DETECTION + CORRESPONDENCE

MOTION FIELD / IMAGE FLOW FIELD

- MOTION FIELD = IMAGE VELOCITY of A POINT THAT IS MOVING IN SCENE. I.E., 2D VECTOR FIELD THAT IS PERSPECTIVE PROJECTION ON TO IMAGE PLANE of 3D VELOCITY FIELD of MOVING SCENE



$\vec{v}_i = \frac{dr_i}{dt}$ = velocity at p_i

$\bar{v}_i' = \frac{dr_i'}{dt}$ = velocity at p_i'

$\hat{z}$ = unit vector in z direction

- BY PERSPECTIVE PROJ. EQ:

$$\frac{\bar{r}_i'}{f'} = -\frac{\bar{r}_i}{\bar{r}_i \cdot \hat{z}}$$

$$\Rightarrow \bar{r}_i' = -\frac{f'\bar{r}_i}{\bar{r}_i \cdot \hat{z}}$$

- DIFFERENTIATING & SIMPLIFYING:

$$\bar{v}_i' = \frac{d\bar{r}_i'}{dt} = -\frac{f'(\bar{r}_i \times \bar{v}_i) \times \hat{z}}{(\bar{r}_i \cdot \hat{z})^2}$$

MOTION FIELD

- HOW TO RECOVER $\bar{v}'$?

OPTICAL FLOW

- OPTICAL FLOW (AKA IMAGE-INTENSITY FIELD) = MOTION OF THE <u>BRIGHTNESS PATTERN</u> IN IMAGE
- ASSUME MOTION FIELD ≈ OPTICAL FLOW
- COUNTER EXAMPLES:

  * ROTATING LAMBERTIAN SPHERE OF CONSTANT ALBEDO, FIXED LIGHT SOURCE AND CAMERA

  ⇒ IMAGE CONSTANT OVER TIME

  ⇒ OPTICAL FLOW = 0

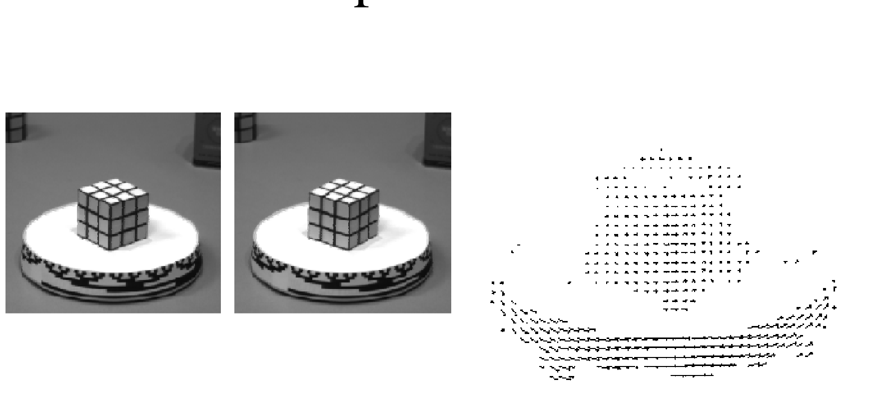
  BUT MOTION FIELD ≠ 0

  * FIXED LAMBERTIAN SPHERE OF CONSTANT ALBEDO, FIXED CAMERA, MOVING LIGHT

  ⇒ OPTICAL FLOW ≠ 0

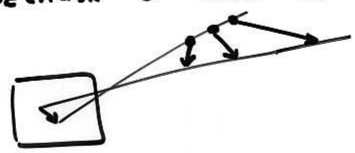
  BUT MOTION FIELD = 0

Optical flow



OPTICAL FLOW PROPERTIES

- ASSUMING GENERAL SCENE MOTION, THERE DOES NOT EXIST A UNIQUE OPTICAL FLOW FIELD THAT IS CONSISTENT WITH IMAGE SEQUENCE.
  (BECAUSE OF BOTH GEOMETRIC

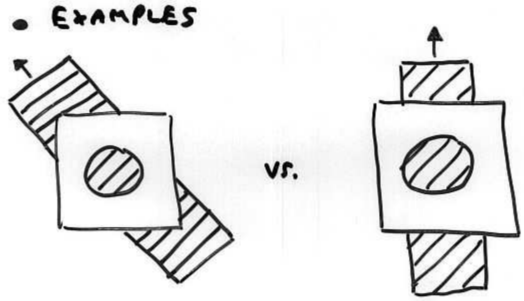


  AND PHOTOMETRIC AMBIGUITY.)
  ⇒ NEED ADDITIONAL CONSTRAINTS

- GENERAL MOTION OF NON-RIGID OBJECTS ⇒ INFINITE NUMBER OF INTERPRETATIONS THAT ARE CONSISTENT W/ DATA

- LOCAL CHANGES IN SPATIOTEMPORAL INTENSITY ARE NOT SUFFICIENT TO COMPLETELY DETERMINE OPTICAL FLOW

- EXAMPLES

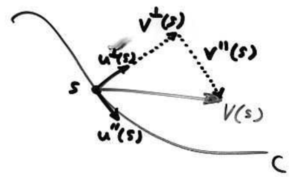


THEY LOOK THE SAME INSIDE THE CIRCULAR WINDOW!
I.E., DIFFERENT PHYSICAL MOTION, BUT SAME MEASURABLE MOTION IN FIXED WINDOW

APERTURE PROBLEM

APERTURE PROBLEM

- GIVEN A STRAIGHT LINE IN MOTION AND A FIXED MEASUREMENT WINDOW, ONLY THE COMPONENT OF MOTION $\perp$ TO LINE ORIENTATION IS RECOVERABLE



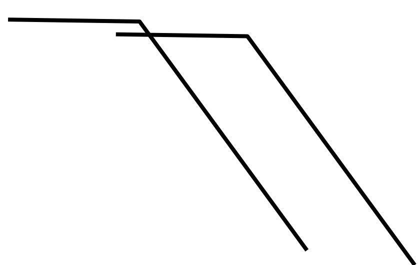
$$V(s) = v^{\parallel}(s)\,u^{\parallel}(s) + v^{\perp}(s)\,u^{\perp}(s)$$

$u^{\perp}(s)$, $u^{\parallel}(s)$ are unit vectors and are measurable

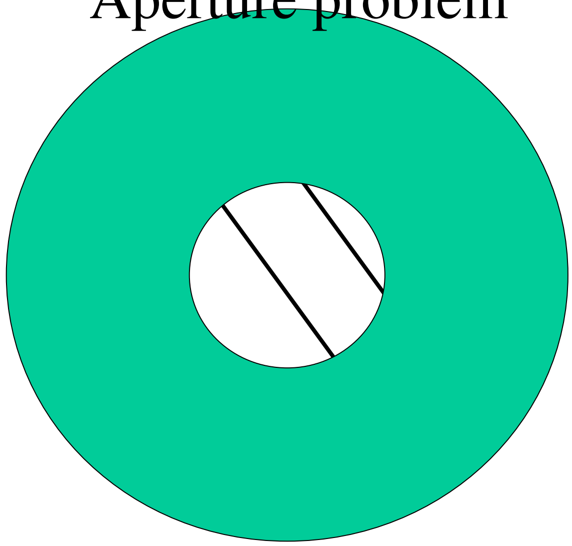
$v^{\perp}(s)$ computable

$v^{\parallel}(s)$ <u>not</u> locally computable in general

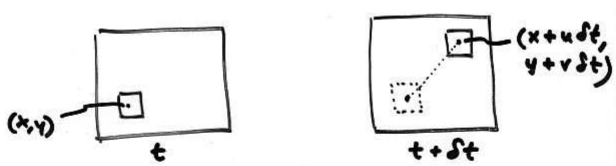
Aperture problem



Aperture problem



Optical Flow Constraint Equation



Optical flow = (u,v) (velocities)

Displacement = $\begin{cases} \delta x = u\,\delta t \\ \delta y = v\,\delta t \end{cases}$

Brightness Constancy Assumption:
Brightness pattern of patch is same in both images

$\Rightarrow$

$$I(x+u\delta t,\ y+v\delta t,\ t+\delta t) = I(x,y,t)$$

- Expand LHS using Taylor series about pt (x,y,t) and keep only linear (1st order) terms:

$$I(x,y,t) + \delta x \frac{\partial I}{\partial x} + \delta y \frac{\partial I}{\partial y} + \delta t \frac{\partial I}{\partial t} \cong I(x,y,t)$$

- Divide by δt and take limit $\delta t \to 0$:

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

- Let $u = \frac{dx}{dt}$, $v = \frac{dy}{dt}$

$$\Rightarrow \boxed{\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} = 0}$$

Optical Flow Constraint Eq.

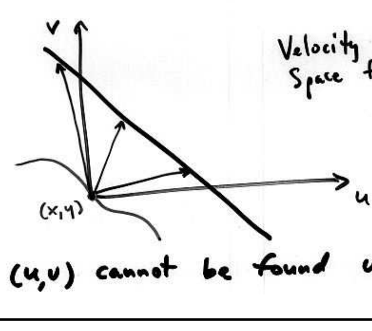
or $\boxed{I_x u + I_y v + I_t = 0}$

Given: 2 images, $I_1(x,y) = I(x,y,t_0)$
$I_2(x,y) = I(x,y,t_0+\delta t)$

Compute: $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, $\frac{\partial I}{\partial t}$ using gradient operators

Solve for: 2 unknowns: u, v

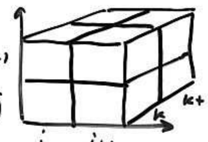
Under constrained: (u,v) must lie on a line in (u,v) space:



Velocity (Displacement) Space for a single pt.

(u,v) cannot be found uniquely!

Computing $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, $\frac{\partial I}{\partial t}$



$$\frac{\partial I}{\partial x} = I_x = \frac{1}{4}\left(I_{i+1,j,k} + I_{i+1,j,k+1} + I_{i+1,j+1,k} + I_{i+1,j+1,k+1}\right) - \frac{1}{4}\left(I_{i,j,k} + I_{i,j,k+1} + I_{i,j+1,k} + I_{i,j+1,k+1}\right)$$

Similarly for $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$

Computing Optical Flow

- Formulate error in optical flow constraint

$$e_c = \iint_{\text{Image}} \left(I_x u + I_y v + I_t\right)^2 \, dx\, dy$$

where $I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$, $I_t = \frac{\partial I}{\partial t}$

- Add additional constraint(s)
- Smoothness Constraint
  (Horn and Schunck, 1981)
  - Nearby points in image undergo similar motion

  $\Rightarrow$ Given n nearby points in domain D in image, assume $V = (u, v)$ is constant for all n points.

  $\Rightarrow$ n linear equations in 2 unknowns

- Formulate error in image smoothness:

$$e_s = \iint_{\substack{\text{pts} \\ \text{in } D}} \underbrace{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2}_{\|\nabla u\|^2} + \underbrace{\left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}_{\|\nabla v\|^2} dx\,dy$$

- Find (u,v) at each image pixel that minimizes:

$$e = e_s + \lambda e_c$$

Optical Flow Algorithm (Discrete)

Consider image point (i,j)

- Departure of smoothness of optical flow at (i,j)

$$s_{ij} = \frac{1}{4}\left((u_{i+1,j} - u_{i,j})^2 + (u_{i,j+1} - u_{i,j})^2 + (v_{i+1,j} - v_{i,j})^2 + (v_{i,j+1} - v_{i,j})^2\right)$$

- Error in optical flow constraint equation

$$c_{ij} = \left(I_x u_{i,j} + I_y v_{i,j} + I_t\right)^2$$

- **Goal**: Find the set $\{u_{i,j}\}$ and $\{v_{i,j}\}$ that minimize

$$e = \sum_i \sum_j (s_{ij} + \lambda c_{ij})$$

- Differentiating e wrt $u_{k,l}$ and $v_{k,l}$

$$\frac{\partial e}{\partial u_{kl}} = 2(u_{kl} - \bar{u}_{kl}) + 2\lambda(I_x u_{kl} + I_y v_{kl} + I_t) I_x$$

$$\frac{\partial e}{\partial v_{kl}} = 2(v_{kl} - \bar{v}_{kl}) + 2\lambda(I_x u_{kl} + I_y v_{kl} + I_t) I_y$$

where $\bar{u}_{kl}$ and $\bar{v}_{kl}$ are averages of u and v around (k,l)

- Setting $\frac{\partial e}{\partial u_{kl}} = 0$ and $\frac{\partial e}{\partial v_{kl}} = 0$ we get:

$$\boxed{\begin{aligned} u_{kl}^{n+1} &= \bar{u}_{kl}^{n} - \frac{I_x \bar{u}_{kl}^{n} + I_y \bar{v}_{kl}^{n} + I_t}{1 + \lambda(I_x^2 + I_y^2)} I_x \\ v_{kl}^{n+1} &= \bar{v}_{kl}^{n} - \frac{I_x \bar{u}_{kl}^{n} + I_y \bar{v}_{kl}^{n} + I_t}{1 + \lambda(I_x^2 + I_y^2)} I_y \end{aligned}}$$

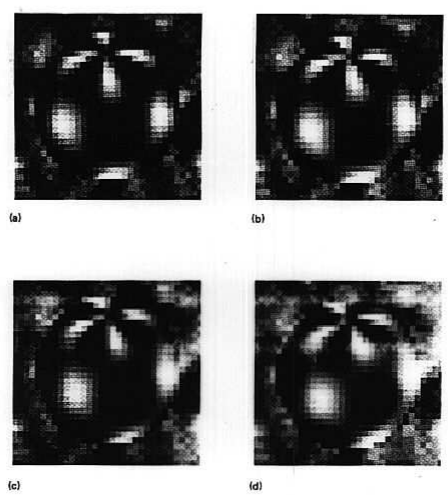
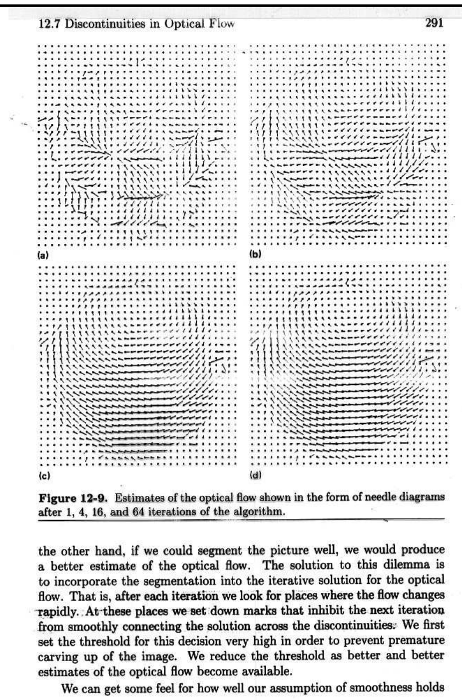


Figure 12-8. Four frames of a synthetic image sequence showing a sphere slowly rotating in front of a randomly patterned background.

12.7 Discontinuities in Optical Flow

There will be discontinuities in the optical flow on the silhouettes, where one object occludes another. We must detect these places if we are to prevent the method presented above from trying to continue the solution smoothly from one region to the other. This seems like a chicken-and-egg problem: If we have a good estimate of the optical flow, we can look for places where it changes very rapidly in order to segment the picture. On



12.7 Discontinuities in Optical Flow 291

(a) (b)

(c) (d)

Figure 12-9. Estimates of the optical flow shown in the form of needle diagrams after 1, 4, 16, and 64 iterations of the algorithm.

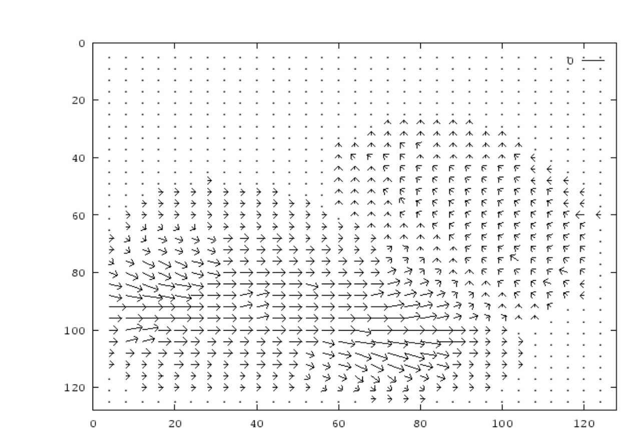
the other hand, if we could segment the picture well, we would produce a better estimate of the optical flow. The solution to this dilemma is to incorporate the segmentation into the iterative solution for the optical flow. That is, after each iteration we look for places where the flow changes rapidly. At these places we set down marks that inhibit the next iteration from smoothly connecting the solution across the discontinuities. We first set the threshold for this decision very high in order to prevent premature carving up of the image. We reduce the threshold as better and better estimates of the optical flow become available.

We can get some feel for how well our assumption of smoothness holds

Hamburg Taxi Video

Hamburg Taxi Video

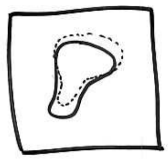
Horn & Schunck Optical Flow



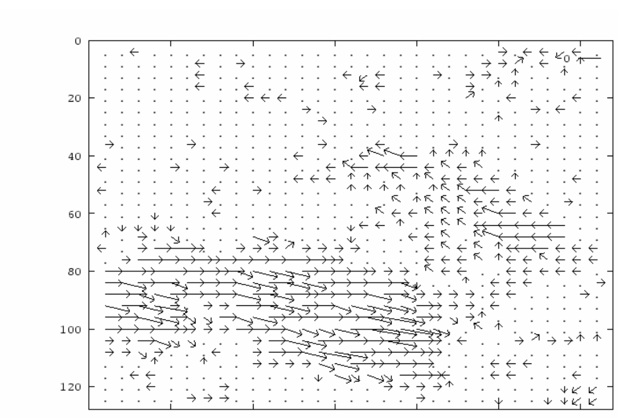
Results

* Works well for <u>textured objects</u>
* Does <u>not</u> work well for <u>homogeneous regions</u>
* Does <u>not</u> work well at points where optical flow is <u>discontinuous</u>. E.g. <u>occluding contours</u>

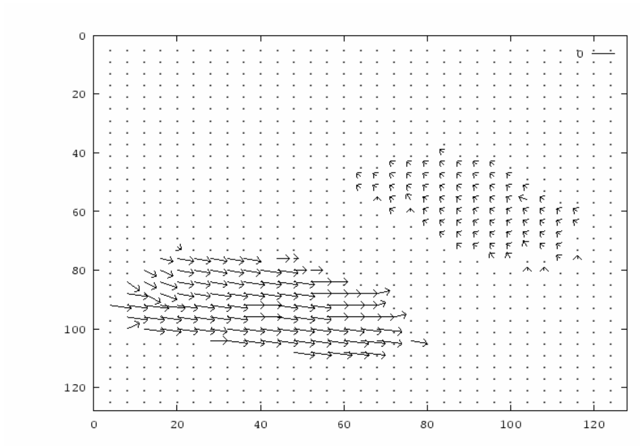
  ⇒ Mark these points and exclude them from the relaxation algorithm



Fleet & Jepson Optical Flow



Tian & Shah Optical Flow



Solving the Aperture Problem

- Basic idea: assume motion field is smooth
- Horn and Schunk: add smoothness term

$$\int\int (I_t + \nabla I \cdot [u\ v])^2 + \lambda^2 (\|\nabla u\|^2 + \|\nabla v\|^2)\ dx\ dy$$

- Lucas and Kanade: assume locally constant motion
  - pretend the pixel's neighbors have the same (u,v)
    - If we use a 5x5 window, that gives us 25 equations per pixel!

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

  - works better in practice than Horn and Schunk

Lucas-Kanade Flow

- 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!

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

$$\underset{25\times 2}{\underbrace{\begin{bmatrix} I_x(\mathbf{p_1}) & I_y(\mathbf{p_1}) \\ I_x(\mathbf{p_2}) & I_y(\mathbf{p_2}) \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}}) & I_y(\mathbf{p_{25}}) \end{bmatrix}}_{A}} \underset{2\times 1}{\underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d}} = - \underset{25\times 1}{\underbrace{\begin{bmatrix} I_t(\mathbf{p_1}) \\ I_t(\mathbf{p_2}) \\ \vdots \\ I_t(\mathbf{p_{25}}) \end{bmatrix}}_{b}}$$

Lucas-Kanade Flow

- Problem: more equations than unknowns

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

- Solution: solve least squares problem
  - minimum least squares solution given by solution of:

$$\overset{2\times 2}{(A^T A)} \ \overset{2\times 1}{d} = \overset{2\times 1}{A^T b}$$

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

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

Eigenvectors of A^TA

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

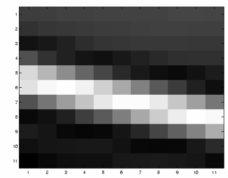
- Suppose (x,y) is on an edge. What is A^TA?
  - gradients along edge all point the same direction
  - gradients away from edge have small magnitude

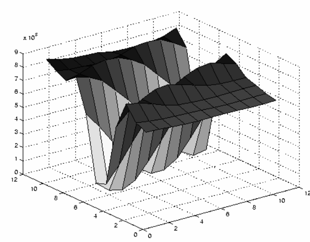
    $\left(\sum \nabla I (\nabla I)^T\right) \approx k \nabla I \nabla I^T$

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

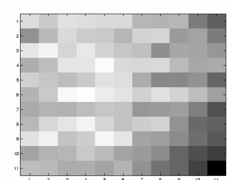


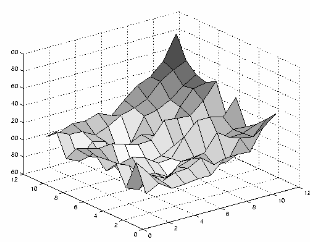


$\sum \nabla I (\nabla I)^T$

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

Low Texture Region

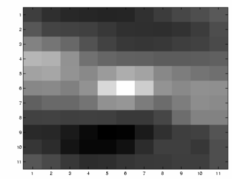


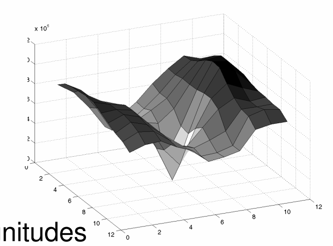


$\sum \nabla I (\nabla I)^T$

– gradients have small magnitude
– small λ_1, small λ_2

High Texture 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

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

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

- 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 with more iterations

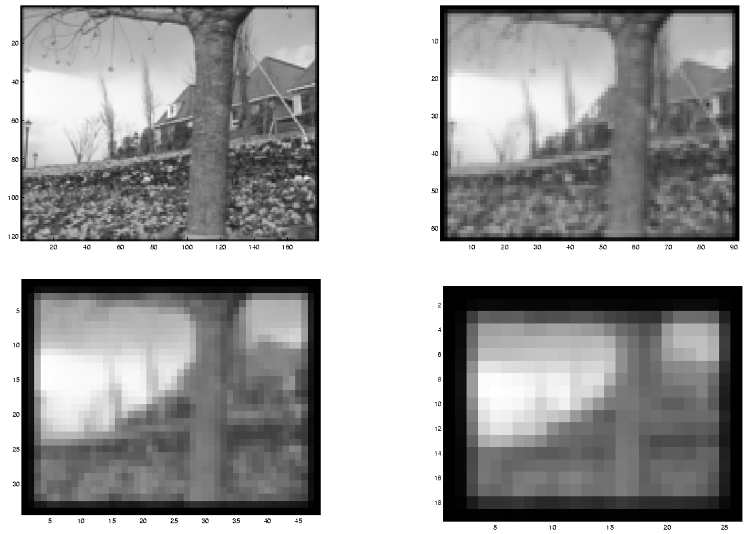
Iterative Refinement

- 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

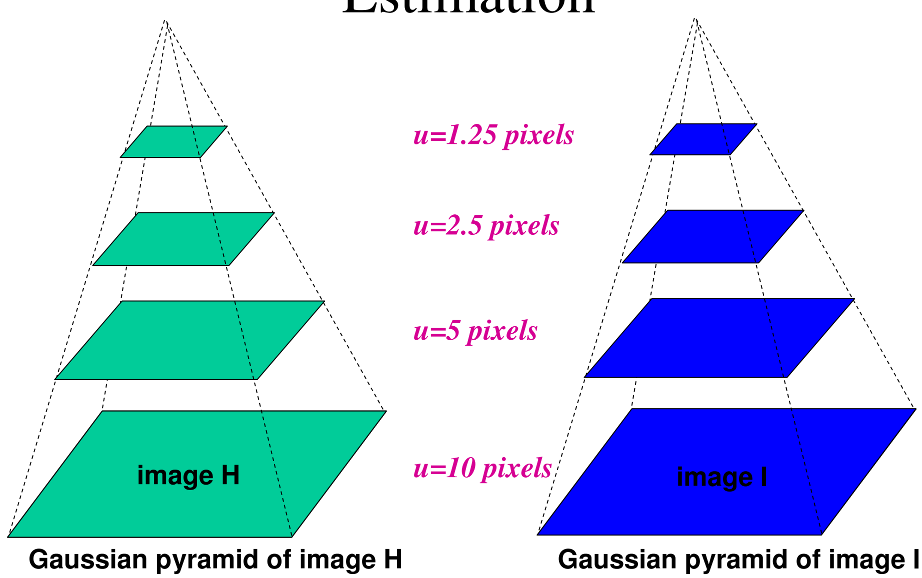
Revisiting the Small Motion Assumption

- When is the motion small enough?
  – Not if it's much larger than one pixel (2^{nd} order terms dominate)
  – How might we solve this problem?

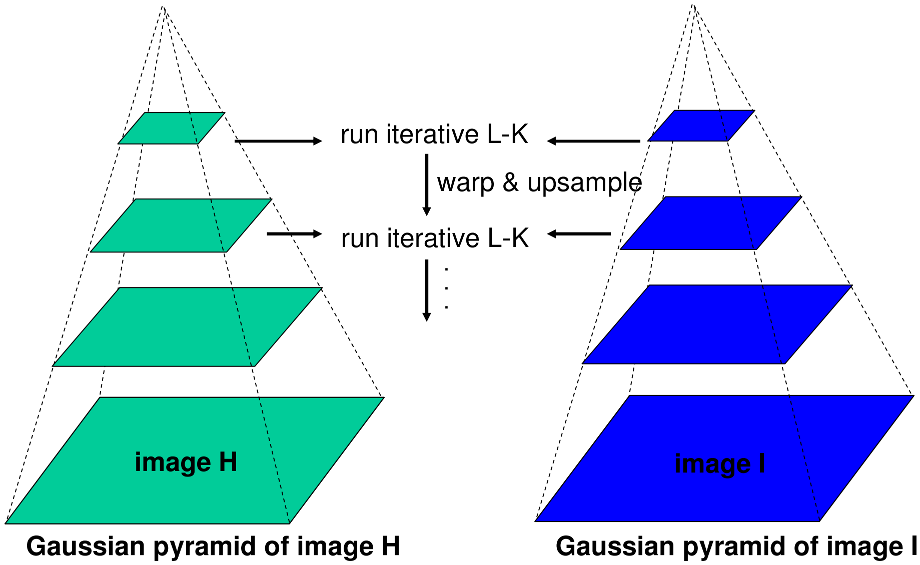
Reduce the Resolution



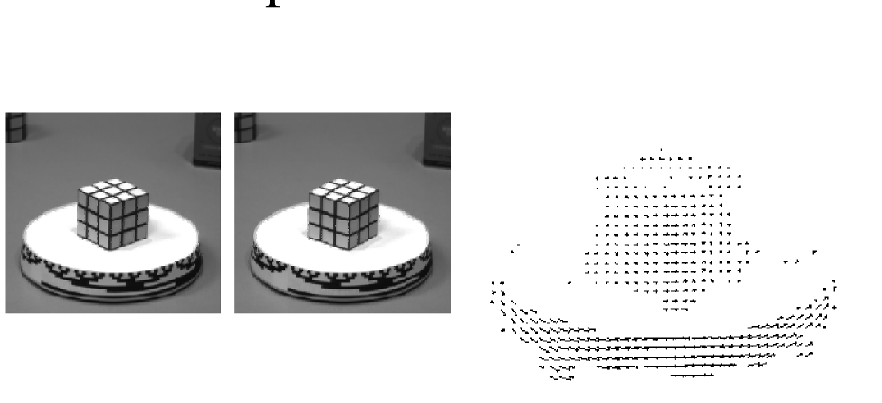
Coarse-to-Fine Optical Flow Estimation



Coarse-to-Fine Optical Flow Estimation



Optical Flow Result



Spatiotemporal (x-y-t) Volumes

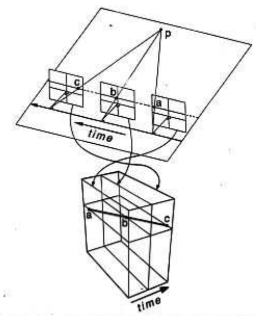


Figure 8.9 Geometry of image acquisition by a camera that is capturing images at equidistant intervals while the camera is translating horizontally in a direction parallel to the camera's image plane. Three positions of the camera's center of projection and its image plane are illustrated, and in each instance, the image position of the scene point *p* is indicated. The figure illustrates that a fixed scene point sweeps out a straight horizontal trajectory in the spatiotemporal stack of images that are captured by a camera at equidistant intervals while the camera is translating horizontally along a straight line parallel to its image plane. (From [Bolles, Baker, and Marimont 1987] with permission.)

8.1.2 Estimation Based on Spatiotemporal Coherence

Approaches to motion-field estimation that we are describing as based on spatiotemporal coherence apparently originated with Yamamoto [Yamamoto

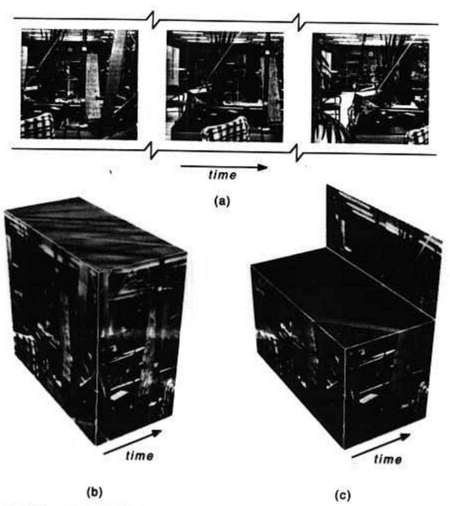
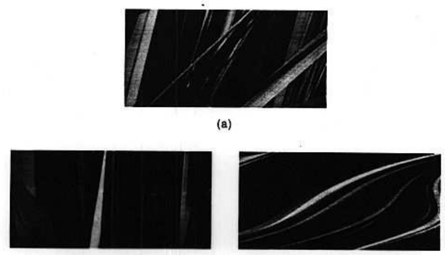


Figure 8.10 Spatiotemporal image data acquired by a camera that is capturing images at equidistant intervals while the camera is translating horizontally (from right to left) in a direction parallel to the camera's image plane. **(a)** Three frames of a 125-image sequence. **(b)** Oblique view of the image sequence in (a) after the sequence has been stacked into a three-dimensional spatiotemporal image data block. **(c)** Same as (b), except now the image data block is horizontally sliced; the top of the last image is shown in conjunction with the bottom of the first image to illustrate the genesis of horizontal spatiotemporal streaks in the image data block. (Adapted from [Bolles, Baker, and Marimont 1987] with permission.)



(b) (c)

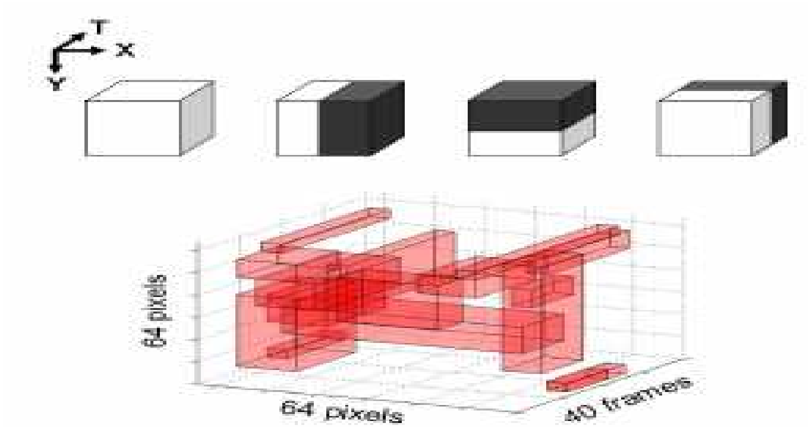
Figure 8.11 Slices of spatiotemporal image data blocks illustrating the image motions of scene points under various camera motions. In each instance, the individual images in the data block from which the slice is taken are captured at equidistant displacements of the camera. **(a)** This horizontal slice from Figure 8.10(c) illustrates the linear (horizontal) spatiotemporal image trajectories of scene points under linear (horizontal) translation of the camera parallel to its image plane. **(b)** This planar slice illustrates the planar, but not necessarily linear, spatiotemporal image trajectories of scene points under linear translation of the camera in a direction not parallel to its image plane—the planar slice here is not horizontal, but rather is oriented to contain the focus of expansion, which sweeps out a straight horizontal trajectory orthogonal to the individual image frames. **(c)** This slice, which is nonplanar in three dimensions, illustrates the nonplanar (and, therefore, nonlinear) spatiotemporal image trajectories of scene points under motion of the camera that includes both linear translation and simultaneous rotation. (From [Bolles, Baker, and Marimont 1987] with permission.)

Now, the spatiotemporal image of a scene point is a coherent structure of one higher dimension: It is a curve. The spatiotemporal image of a scene curve is a surface, and the spatiotemporal image of a scene surface is a volume. Whereas we can easily determine the image velocity of a scene point that maps onto a locally discriminable spatiotemporal image curve by projecting the tangent to this curve (i.e., its time derivative) onto the image plane, the image velocity of a scene point whose spatiotemporal image can be determined only to lie on some discriminable spatiotemporal surface has a

Visual Event Detection using Volumetric Features

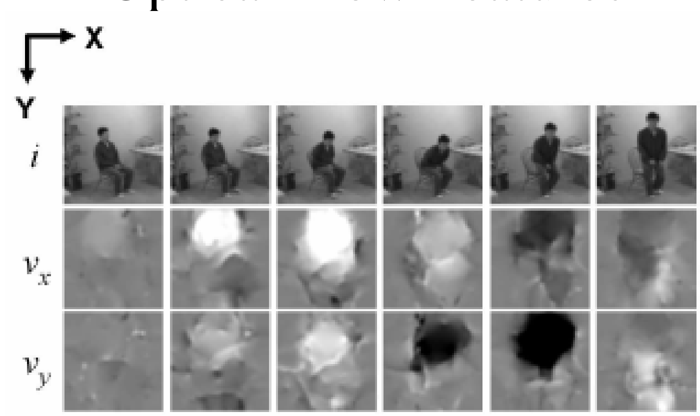
- Y. Ke, R. Sukthankar, and M. Hebert, CMU, CVPR 2005
- Goal: Detect motion events and classify actions such as *stand-up*, *sit-down*, *close-laptop*, and *grab-cup*
- Use *x-y-t* features of optical flow
  - Sum of *u* values in a cube
  - Difference of sum of *v* values in one cube and *v* values in an adjacent cube

3D Volumetric Features



Approximately 1 million features computed

Optical Flow Features



Optical flow of *stand-up* action (light means positive direction)

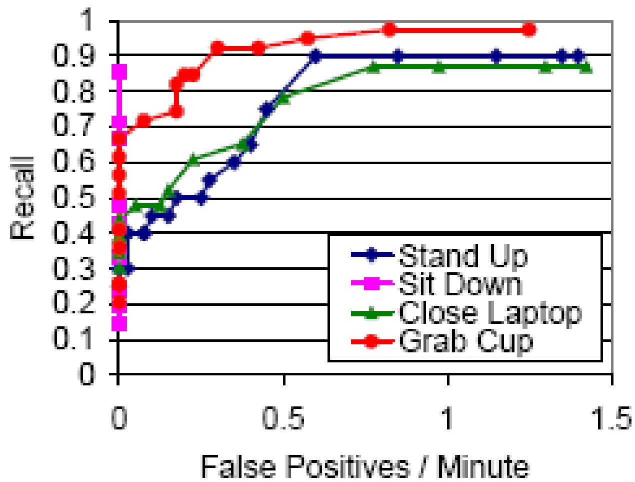
Classifier

- Cascade of binary classifiers that vote on the classification of the volume
- Given a set of positive and negative examples at a node, each feature and its optimal threshold is computed. Iteratively add filters at each node until a target detection rate (e.g., 100%) or false positive rate (e.g., 20%) is achieved
- Output of the node is the majority vote of the individual filters

Action Detection

- 78% - 92% detection rate on 4 action types: *sit-down*, *stand-up*, *close-laptop*, *grab-cup*
- 0 – 0.6 false positives per minute
- Note: while lengths of actions vary, the first frames are all aligned to a standard starting position for each action
- Classifier learns that beginning of video is more discriminative than end because of variable length
- Relatively robust to viewpoint (< 45 degrees) and scale (< 3x)

Results



Structure-from-Motion

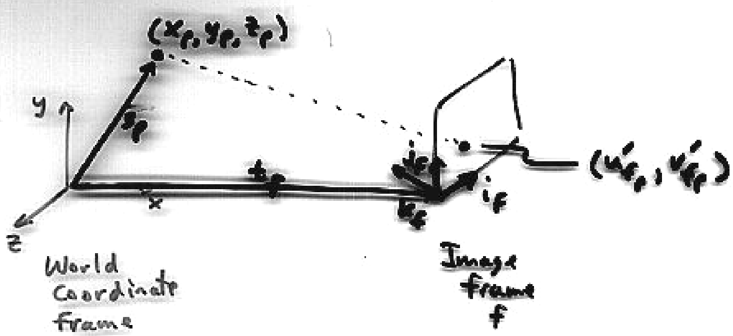
- Determining the 3-D structure of the world, and the motion of a camera (i.e., its extrinsic parameters) using a sequence of images taken by a moving camera
  - Equivalently, we can think of the world as moving and the camera as fixed
- Like stereo, but the position of the camera isn't known (and it's more natural to use many images with little motion between them, not just two with a lot of motion) and we have a long sequence of images, not just 2 images
  - We may or may not assume we know the intrinsic parameters of the camera, e.g., its focal length

Factorization Method

- Tomasi and Kanade, 1990
- GOAL: Compute both shape and motion. Shape computed in object-centered coordinate frame to avoid noise problems when small objects are far from camera
- ASSUMPTIONS:
  - Fixed scene
  - Moving Camera
  - Orthographic projection
  - Batch processing
  - No motion model
  - No shape model

- Input: From a sequence of F images, track P feature points over <u>all</u> frames ($\Rightarrow$ must be visible entire time)

$$\{(u'_{fp}, v'_{fp}) \mid f = 1, \dots, F;\ p = 1, \dots, P\}$$



- Scene pt $s_p = (x_p, y_p, z_p)^T$ orthographically projects to (u'_{fp}, v'_{fp}) where

$$u'_{fp} = i_f^T(s_p - t_f)$$
$$v'_{fp} = j_f^T(s_p - t_f)$$

where i_f^T = unit vector in camera frame x-direction

j_f^T = unit vector in camera frame y-direction

$t_f = (a_f, b_f, c_f)^T$ translation vector from world coord origin to camera origin

- Let world coord. origin be at centroid of scene pts.

$$\Rightarrow \frac{1}{P}\sum_{p=1}^{P} s_p = 0$$

- Define image coords relative to centroid of image pts:

$$\begin{cases} u_{fp} = u'_{fp} - \bar{u}_f \\ v_{fp} = v'_{fp} - \bar{v}_f \end{cases}$$

where $\bar{u}_f = \frac{1}{P}\sum_{p=1}^{P} u'_{fp}$, $\bar{v}_f = \frac{1}{P}\sum_{p=1}^{P} v'_{fp}$

Hence,

$$\begin{aligned} u_{fp} &= u'_{fp} - \bar{u}_f \\ &= i_f^T(s_p - t_f) - \frac{1}{P}\sum_i i_f^T(s_p - t_f) \\ &= i_f^T\left((s_p - t_f) - \frac{1}{P}\sum_i (s_p - t_f)\right) \\ &= i_f^T\Big(s_p - \underbrace{\frac{1}{P}\sum s_p}_{=0}\Big) \\ &= i_f^T s_p \end{aligned}$$

Uses the fact that the centroid is preserved under orthographic projection. I.e., projection of centroid = centroid of projections

- Similarly, $v_{fp} = j_f^T s_p$

- Interpretation:

  $[i_f, j_f]$ represents "motion" of f^{th} frame
  I.e., rotation of camera
  (6 unknowns)

  $s_p = (x_p, y_p, z_p)$ represents "shape"
  I.e., position of p^{th} point in world coord. frame
  (3 unknowns)

  ⇒ Each point in each frame produces 2 equations in 9 unknowns

  ⇒ P points in F frames produces 2FP equations

- Let $U = \begin{bmatrix} u_{11} & \cdots & u_{1P} \\ \vdots & \ddots & \vdots \\ u_{F1} & \cdots & u_{FP} \end{bmatrix}$ ← frame 1 pts … ← frame F pts

  ↑pt 1 tracked over F frames

  Similarly define V.

- Define "Measurement Matrix"

  $W = \begin{bmatrix} U \\ \hline V \end{bmatrix}$ 2F × P matrix of "registered" image pts

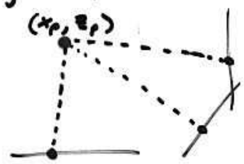
- Let $M = \begin{bmatrix} i_1^T \\ \vdots \\ i_F^T \\ j_1^T \\ \vdots \\ j_F^T \end{bmatrix}$ ← orientation of frame 1's x-axis wrt world coords

  2F × 3 "motion" matrix

  $S = [s_1 \cdots s_P]$ 3 × P "shape" matrix

- So, $W = MS$
  given W, solve for M and S

- Problem is constrained because each pt projects in each frame along a "pencil" of projector lines:



$$|i_f| = |j_f| = 1$$
$$i_f^T j_f = 0$$

$\Rightarrow$ $2FP$ measurements are <u>not</u> independent

- <u>RANK Theorem</u>:
  Without noise, W is at most rank 3.
  (since M is $2F \times 3$ and S is $3 \times P$)

- Noise corrupts W, so $\mathrm{rank}(W) \neq 3$

By SVD decompose W:

$$W = L \Sigma R$$

$$= \begin{array}{c} \\ 2F \end{array}\!\begin{array}{c} P \\ \begin{bmatrix} & \\ & \\ & \end{bmatrix} \end{array} \begin{array}{c} P \\ \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_P \end{bmatrix} \end{array} \begin{array}{c} P \\ \begin{bmatrix} & \\ & \end{bmatrix} \end{array}\!\begin{array}{c} \\ P \end{array}$$

Partitioning we get

$$L \Sigma R = L' \Sigma' R' + L'' \Sigma'' R''$$

where
$$L = \left[\underset{3}{L'} \;\; \underset{P-3}{L''} \right] \; 2F$$

$$\Sigma = \begin{array}{c} \begin{bmatrix} \Sigma' & 0 \\ 0 & \Sigma'' \end{bmatrix} \\ \begin{array}{cc} 3 & P-3 \end{array} \end{array} \begin{array}{l} 3 \\ P-3 \end{array}$$

$$R = \begin{array}{c} \begin{bmatrix} R' \\ R'' \end{bmatrix} \\ P \end{array} \begin{array}{l} 3 \\ P-3 \end{array}$$

8

By rank theorem, ideal matrix W^* has rank ≤ 3 $\Rightarrow$

$\sigma_4 = \ldots = \sigma_p = 0$ for W^*

W = noisy version of W^*

$\Rightarrow L'' \Sigma'' R''$ due to noise only

and $L' \Sigma' R'$ represents best approximation of W^*

Factorization Algorithm

1. Compute W
2. Compute $SVD(W) = L \Sigma R$
3. $\hat{M} = L' \Sigma'^{\frac{1}{2}}$ after partitioning

   $\hat{S} = \Sigma'^{\frac{1}{2}} R'$ (Hence set to 0 all but 3 largest singular vals)

   Now $\hat{M}, \hat{S}$ of right size but $W^* = \hat{M}\hat{S}$ not unique

4. Find 3×1 matrix A such that

   $M = \hat{M} A^{-1}$

   and $S = A \hat{S}$

   Nonlinear optimization problem

   (add metric constraints to solve for A. E.g., rows in M are unit vectors)

5. Solve for M and S

$$\begin{cases} M = \hat{M} A^{-1} \\ S = A \hat{S} \end{cases}$$

Results

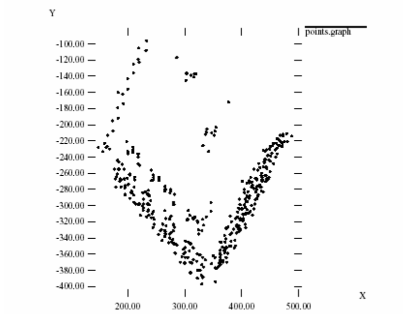


Figure 4.5: A view of the computed shape from approximately above the building (compare with figure 4.6).

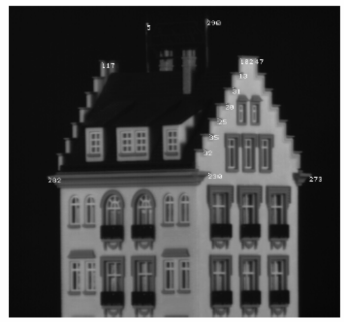


Figure 4.7: For a quantitative evaluation, distances between the features shown in the picture were measured on the actual model, and compared with the computed results. The comparison is shown in figure 4.8.

Extensions

- Paraperspective
  - [Poelman & Kanade, PAMI 97]
- Sequential Factorization
  - [Morita & Kanade, PAMI 97]
- Factorization under perspective
  - [Christy & Horaud, PAMI 96]
  - [Sturm & Triggs, ECCV 96]
- Factorization with Uncertainty
  - [Anandan & Irani, IJCV 2002]

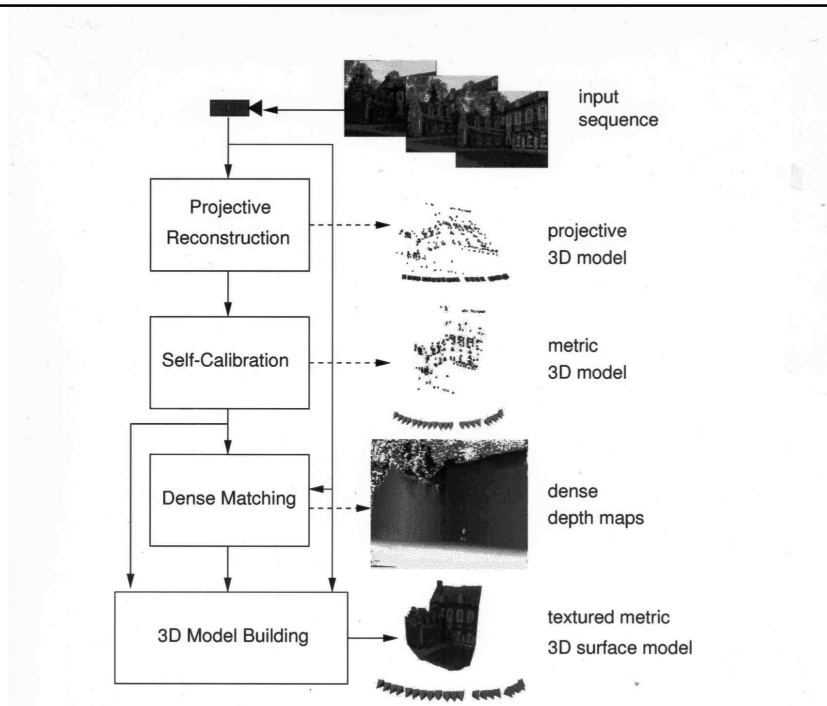
Stratified Approach to Structure from Motion

Given: Sequence of images from an uncalibrated camera moving arbitrarily around a static 3D scene

Goal: 3D model of scene

Approach:

1. Compute Projective reconstruction
2. Compute Affine reconstruction
3. Compute Metric reconstruction



Projective Reconstruction

Goal: Compute for each image its 3×4 projection matrix P that maps

$$\begin{bmatrix} x \\ y \\ s \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Step 1: Detect a sparse set of feature points using a corner detector (e.g., Harris or Tomasi + Kanade) in each image

Step 2: Use RANSAC to estimate the fundamental Matrix, F, between first 2 images

Since $F = [e']_x M'$ homography from I_1 to I_2

$\Rightarrow P = [I \mid 0]$, $P' = [M' \mid e']$ $= [[e']_x F \mid e']$

Step 3: Using the projection matrices, P and P', reconstruct the initial structure by triangulation

Step 4: Incrementally add images $3, \ldots, n$ by detecting corner points, estimate epipolar geom. using RANSAC w/ previous image, using already reconstructed points to compute projection matrix for i^{th} view

$\Rightarrow$ At end of this step we have

$P_1 = [I \mid 0]$

$P_i = [H_{1i} \mid e_{1i}]$, $i = 2, \ldots, n$

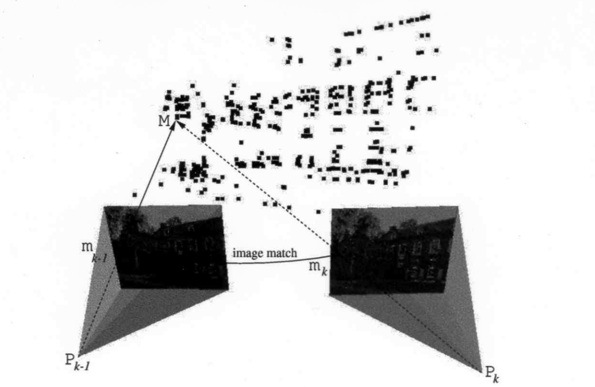


Fig. 3. Image matches (m_{k-1}, m_k) are found as described before. Since the image points, m_{k-1}, relate to object points, M_k, the pose for view k can be computed from the inferred matches (M, m_k). $\mathbf{P_i}$ represent the camera stations.

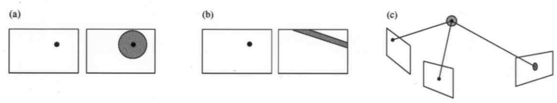


Fig. 4. (a) a priori search range, (b) search range along the epipolar line and (c) search range around the predicted position of the point.

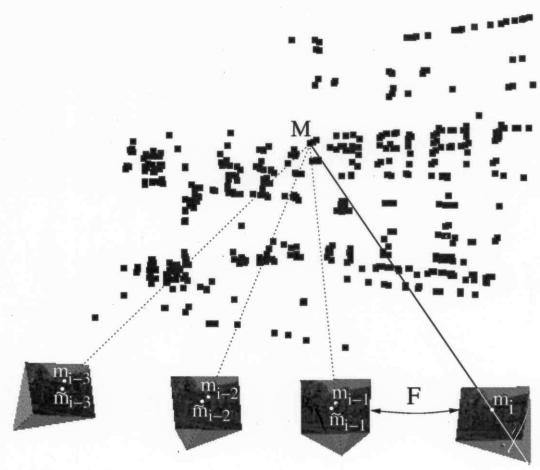
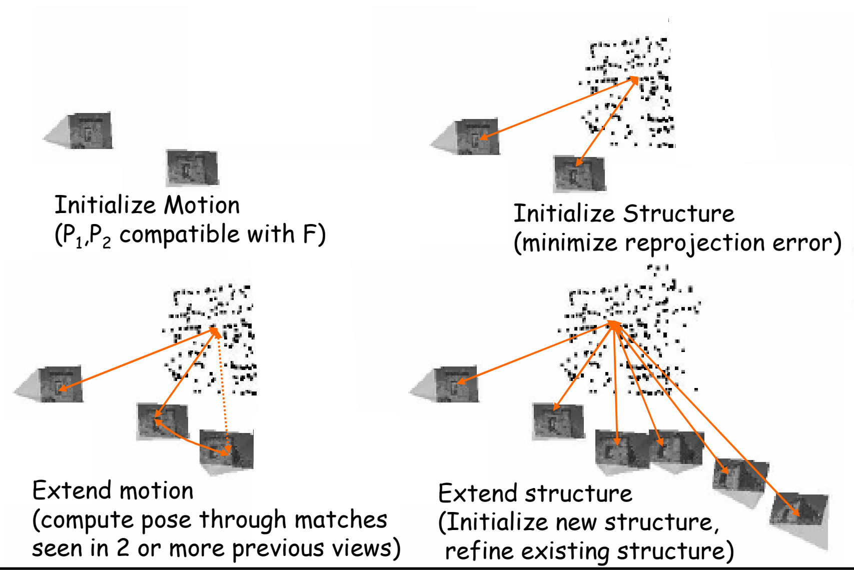


Figure 5.1: Image matches (m_{i-1}, m_i) are found as described before. Since the image points, m_{i-1}, relate to object points, M_i, the pose for view i can be computed from the inferred matches (M, m_i). A point is accepted as an inlier if its line of sight projects sufficiently close to all corresponding points.

Sequential Structure and Motion Computation



Sequential structure and motion recovery

- Initialize structure and motion from two views
- For each additional view
  - Determine pose
  - Refine and extend structure

- Determine correspondences robustly by jointly estimating matches and epipolar geometry

Metric Reconstruction

Goal: Euclidean reconstruction up to a scale factor (called Metric reconstruction)

Step 5: Self-Calibration

→ Compute the intrinsic camera parameters

E.g. add constraints on intrinsic parameters such as

1) no "skew"
2) square pixels

Note: these constraints do not require that other intrinsic parameters not vary over time

e.g. zoom can change

Result is a new set of projection matrices of form:

$$P_i = K_i \left[R_i \mid -R_i t_i \right]$$

where $K_i = \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$

$\alpha_u = f k_u$

$\alpha_v = f k_v$

α_u / α_v = aspect ratio of pixel

(u_0, v_0) = principal point

s = image skew

One common method for self-calib:

Compute the "absolute conic", a virtual conic corresponding to the unique degenerate quadric of planes that is invariant under all rigid transformations

Step 6: Compute dense correspondence

First, rectify images

Then detect correspondences

using 1) epipolar constraint

2) uniqueness constraint

3) ordering constraint (monotonicity)

Step 7: Depth map fusion and uncertainty reduction

→ Find chains of correspondences across subsequences of images

→ Intersect depth estimates

Pollefeys' Result

Object Tracking

- 2D or 3D motion of known object(s)
- Recent survey: “Monocular model-based 3D tracking of rigid objects: A survey” available at http://www.nowpublishers.com/

XVision Tracker

- G. Hager and K. Toyama, CVIU, 1998
- GOAL: REAL-TIME TRACKING USING PC
- MAIN IDEAS:

  1) HIERARCHICAL
     - SMALL SET OF PRIMITIVE FEATURE TYPES
     - COMPLEX OBJECTS BUILT FROM SIMPLER FEATURES

  2) STATE VECTOR DESCRIPTION
     - EACH PRIMITIVE FEATURE DESCRIBED AS A STATE VECTOR
       ⇒ SMALL # PARAMETERS THAT SPECIFY POSITION AND APPEARANCE

  3) WINDOW-BASED, REGIONS-OF-INTEREST

4) TRACKING = INVERSE ANIMATION

- IMAGE WARPING TRANSFORMS RAW IMAGE WINDOW INTO CANONICAL CONFIG.

  $\Rightarrow$ "RECTIFY" FEATURES, THEN DETECT

5) MOTION MODEL = TRANSFORMATION FOR WARPING WINDOWS

- 2D RIGID (3 DoF)
- 2D AFFINE (6 DoF)

PRIMITIVE FEATURE : EDGE (LINE SEGMENT)

- STATE VECTOR AT TIME t:

$$L_t = (x_t, y_t, \theta_t, r_t)^T$$

"appearance" i.e., ideal filter response

- STATE VECTOR UPDATE:

$$L_{t+\delta t} = L_t + \text{Edge}(\text{Warp}_{\text{rotation}}(I(t+\delta t); x_t, y_t, \theta_t); L_t)$$

$\Rightarrow$ 1) Estimate orientation at $t+\delta t$

2) Apply rotational warp to obtain vertical edge

3) Detect position of vertical edge

More specifically,

1) Warp window at several (3) orientations, $+\alpha$, 0, $-\alpha$ wrt ~~orientation~~ at time t.

Equiv. to 1st deriv, then sum

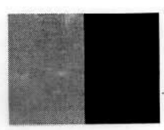
2) Sum columns
3) Compute difference b/w adjacent column sums
4) Max diff = edge strength

5) Quadratic interpolation to estimate orientation

Result = Estimate of translation change, $\delta trans$, in direction $\perp$ to edge, and orientation change, $\delta\theta$

- Update State Vector:

$$\begin{bmatrix} x_{t+\delta t} \\ y_{t+\delta t} \\ \theta_{t+\delta t} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} -\delta trans \ \sin(\theta_t + \delta\theta) \\ \delta trans \ \cos(\theta_t + \delta\theta) \\ \delta\theta \end{bmatrix}$$



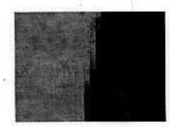


Figure 2. Close-up of tracking windows at two time points. Left, time t_i, where the edge tracking algorithm has computed the correct warp parameters to make an edge appear vertical (the "setpoint"). Right, the edge acquired at time t_{i+1}. The warp parameters computed for t_i were used to acquire the image, but the underlying edge has changed orientation. Figure 3 shows how the new orientation is computed.

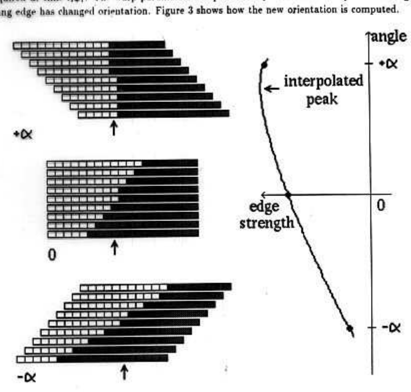


Figure 3. Schematic for computing edge orientation. The diagrams on the left show a window of pixels at three different "orientations." The middle figure displays the edge after a warped acquisition (Figure 2, right). The top and bottom figures show the effect of shifting rows to simulate orientational offset. Summing the columns for each figure and taking differences between adjacent sets of columns gives estimates for edge strength. The arrows at the bottom show where each image experiences the strongest vertical edge within the window. At the right, these values are plotted with angle offset on the vertical axis and edge strengths on the horizontal axis. The three data points are fit to a quadratic, whose peak offers an estimate for the best angular offset of the actual edge. (Both the orientation of the edge in the middle figure and the extent of the shearing in the top and bottom figures have been exaggerated for illustrative purposes.)

PRIMITIVE FEATURE: REGION

- Given 2D Template Window corresponding to approx. <u>planar</u> scene surface, and motion model = affine

- <u>State Vector Description</u>:

$$S_t = (\underbrace{x_t, y_t}_{\text{pos.}}, \underbrace{\theta_t}_{\text{orient}}, \underbrace{s_{x_t}, s_{y_t}}_{\text{scale}}, \underbrace{\gamma_t}_{\text{shear}}, \underbrace{r_t}_{\text{degree of match w/ template}})$$

- <u>State Vector Update</u>

$$S_{t+\delta t} = S_t + SSD(Warp_{affine}(I(t+\delta t); \underbrace{A_t}_{\text{rotate, scale, shear}}; \underbrace{d_t}_{\text{translate}}); S_t)$$

⇒ 1) Apply estimated affine warp

2) Compute degree of match using SSD

3) Brightness constancy assumption

- To estimate best affine transformation, solve optimization problem:

$$\min \sum_{\bar{x} \in W} \left(I(A\bar{x} + \bar{d}, t) - I(\bar{x}, t_0) \right)^2 w(\bar{x})$$

where

$\bar{x} = (x, y)^T$ = pixel in ~~center of~~ window W

$\bar{d} = (u, v)^T$ = translation vector

$$A = \begin{bmatrix} s_x & \gamma \\ 0 & s_y \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$ rotation, scale, shear

$w(\bar{x})$ positive weighting function

- Given solution at t, $(A_t, \bar{d}_t)$, and assuming δt small, solve approx. optimization problem:

$$\min \sum_{\bar{x} \in W} \left(\nabla I(\bar{x},t) \cdot (A'\bar{x} + \bar{d}') + (J(\bar{x},t) - I(\bar{x},t_0)) \right)^2 w(\bar{x})$$

where
A' = change in A during δt
$\bar{d}'$ = change in $\bar{d}$ during δt
$J(\bar{x},t) = I(A_t \bar{x} + \bar{d}_t, t + \delta t)$ ← warped window

Solve for 6 unknowns in A' and $\bar{d}'$

- To improve efficiency of solving,
  1) first solve for translation, rotation
  2) compute scale and shear change

- To cope with illumination change
  normalize windows to have
  1^{st} moment = 0
  2^{nd} moment = 1

- To cope with motion > fraction of pixel
  select resolution adaptively:
  <u>if</u> $|\bar{d}'_t|$ > threshold$_1$
  <u>then</u> halve resolution at $t+\delta t$
  (implement using warping + interpolation)
  <u>else</u> <u>if</u> $|\bar{d}'_t|$ < threshold$_2$
  <u>then</u> double resolution at $t+\delta t$

Size	20×20	40×40	60×60	80×80	100×100
Rotational Warping	0.11	0.40	0.94	1.77	2.83
Scale and Shear[1]	0.39	1.52	3.45	6.11	9.69
Resolution by 2	0.06	0.23	0.54	1.02	1.67
Resolution by 4	0.04	0.13	0.30	0.59	0.97
Affine	0.50	1.92	4.39	7.88	12.52
Affine by 2	0.56	2.15	4.93	8.90	14.19
Affine by 4	0.54	2.05	4.69	8.47	13.49

Figure 1. The time in milliseconds consumed image warping for various size regions. The first two lines show the time for each of the warping stages. The third and fourth lines show the time taken for reducing image resolution by a factor of 2 and 4. The final lines show the time needed for affine warping at various scales based on the component times.

Edge Tracking

20 * 20	.39 msec	on 175 MHz R4400 processor
40 * 40	1.17	
60 * 40	2.09	

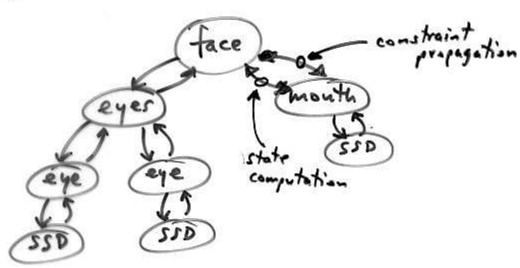
Region (SSD) Tracking

	Rigid	Affine
40 * 40	5.6/1.5 *	8.4/3.7
60 * 60	9.5/3.2	15.6/8.1
80 * 80	17.7/6.1	28.5/14.4

* 1/2 resolution / 1/4 resolution

Composite features

- Feature Network:



- At each cycle,
  1) propagate constraints "down" graph
  2) perform detection at primitive (leaf) features
  3) traverse "up" graph and compute new state vectors for higher level nodes