Optical Flow

CS 763,

Ajit Rajwade

Contents

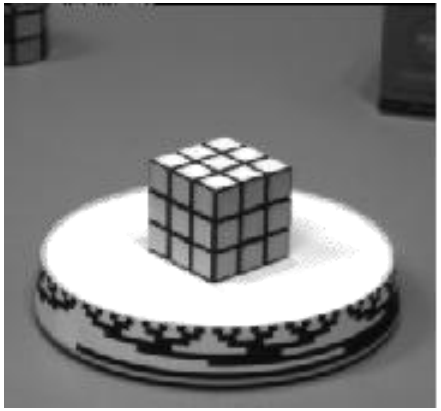
- Definition and example
- Computational methods
- Aperture problem
- Applications

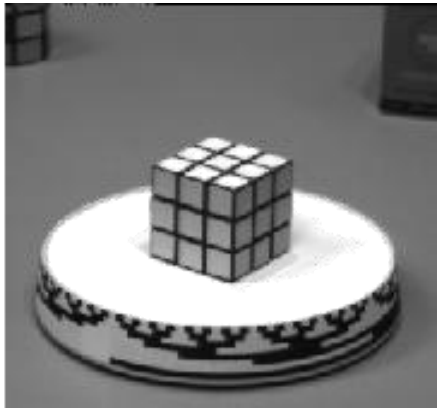
What is optical flow?

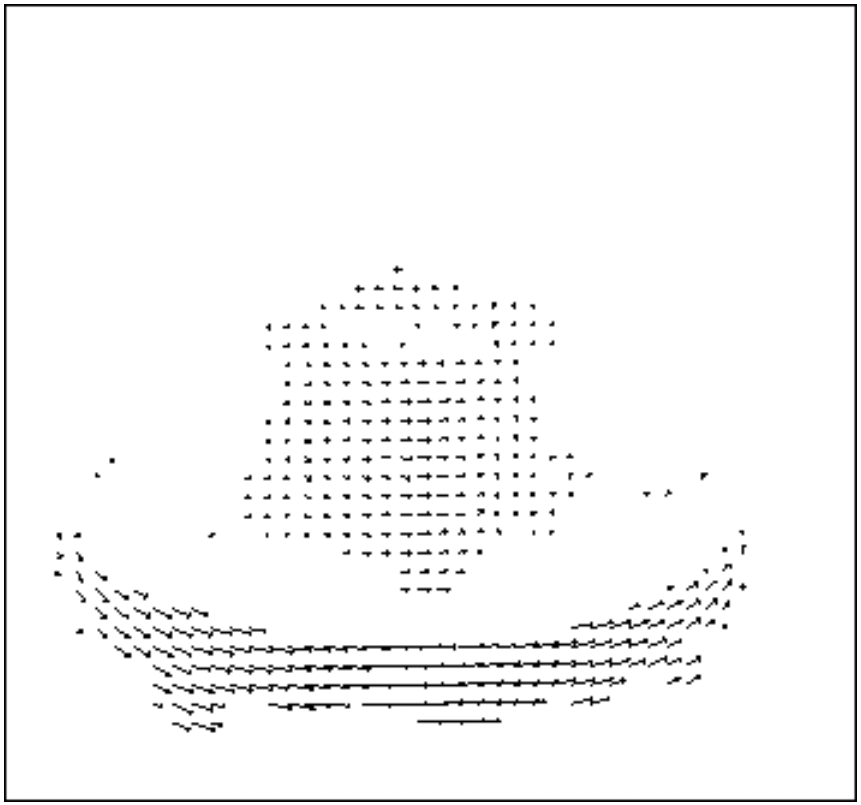
- The brightness pattern in an image will move as the objects in the underlying scene move.

- The *apparent* motion of the brightness patterns is called as **optical flow**.

- The optical flow is a field of 2D vectors and is defined on the image domain, i.e. at each pixel *(x,y)* in the image, there is a vector *(u(x,y),v(x,y))* giving the apparent displacement at *(x,y)* per unit time.







http://www.ri.cmu.edu/research_project_detail.html?project_id=10

Computing optical flow

- Input: *T* >= 2 images

$$\{I(x,y,t)\,|\,1 \le t \le T, (x,y) \in \Omega\}$$

- Output:

$$\{(u(x,y,t), v(x,y,t))\,|\,1 \le t < T, (x,y) \in \Omega\}$$

- **Assumption 1:** The apparent intensity value at a physical point does not change across the image sequence – **brightness constancy assumption.**
- **Assumption 2:** Across consecutive frames, every point undergoes **only a small** displacement.

Computing optical flow

$$I(x,y,t) = I(x+\Delta x, y+\Delta y, t+\Delta t)\,[\text{why?}]$$

$$= I(x,y,t) + \Delta x \frac{\partial I(x,y,t)}{\partial x} + \Delta y \frac{\partial I(x,y,t)}{\partial y} + \Delta t \frac{\partial I(x,y,t)}{\partial t}\,[\text{why?}]$$

$$\therefore \frac{\Delta x}{\Delta t}\frac{\partial I(x,y,t)}{\partial x} + \frac{\Delta y}{\Delta t}\frac{\partial I(x,y,t)}{\partial y} + \frac{\partial I(x,y,t)}{\partial t} = 0$$

$$\therefore u(x,y,t)\frac{\partial I(x,y,t)}{\partial x} + v(x,y,t)\frac{\partial I(x,y,t)}{\partial y} + \frac{\partial I(x,y,t)}{\partial t} = 0$$

Brightness constancy equation

$$\left(u(x,y,t),\quad v(x,y,t)\right)\bullet\left(\frac{\partial I(x,y,t)}{\partial x},\quad \frac{\partial I(x,y,t)}{\partial y}\right) + \frac{\partial I(x,y,t)}{\partial t} = 0$$

Underconstrained – one equation, two unknowns per pixel!

$$(u(x,y,t),\quad v(x,y,t))\bullet\left(\frac{\partial I(x,y,t)}{\partial x},\quad \frac{\partial I(x,y,t)}{\partial y}\right)+\frac{\partial I(x,y,t)}{\partial t}=0$$

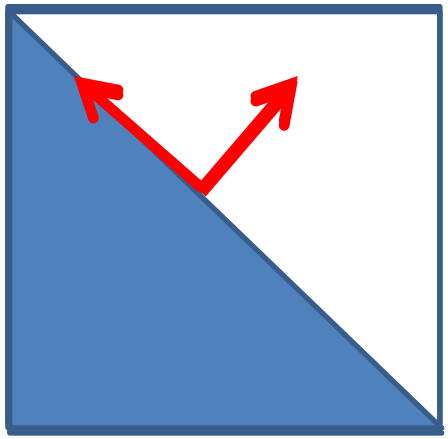
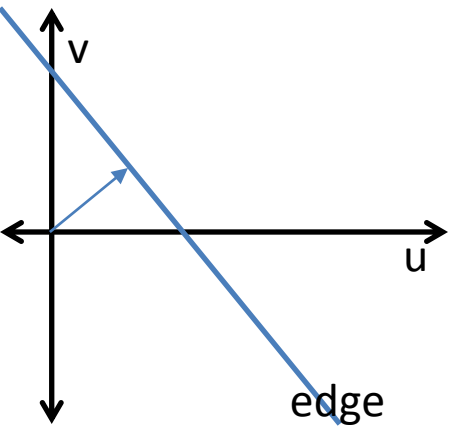


Image gradient – (perpendicular to the edge) – is the direction of the greatest change in the intensity



- ✓ The brightness constancy equation says that only the component of the optical flow along the direction of the **gradient** can be determined!
- ✓ The component along the **edge** direction (i.e. perpendicular to the gradient) **cannot** be determined!

Aperture Problem

- The aforementioned issue is called the **aperture problem** – the motion of an edge as seen through an aperture is essentially ambiguous.

- See demo at:

  http://web.mit.edu/persci/demos/Motion&Form/demos/one-square/one-square.html

Working around the aperture problem

- Two methods:

✓ Horn and Shunck method – global method

Horn and Shunck, "Determining optical flow",
http://people.csail.mit.edu/bkph/papers/Optical_Flow_OPT.pdf

✓ Lucas-Kanade method – local (semi-local) method, uses linear algebra

Horn and Shunck

- The optical flow ($u(x,y),v(x,y)$) at pixel (x,y) is *undetermined* in the direction of the edge at (x,y).
- Image intensities can be *noisy*.
- Mitigate these problems, by assuming that the underlying optical flow is **smooth**, i.e. the optical flow vectors at adjacent pixels are **similar**.
- Leads to following energy functional to be minimized:

$$J(u,v) = \iint_{\Omega} \left((I_x u + I_y v + I_t)^2 + \lambda (u_x^2 + u_y^2 + v_x^2 + v_y^2) \right) dx dy$$

Notice the following!

- The second term is called a **smoothness term** or **regularizer** or **regularization term**.

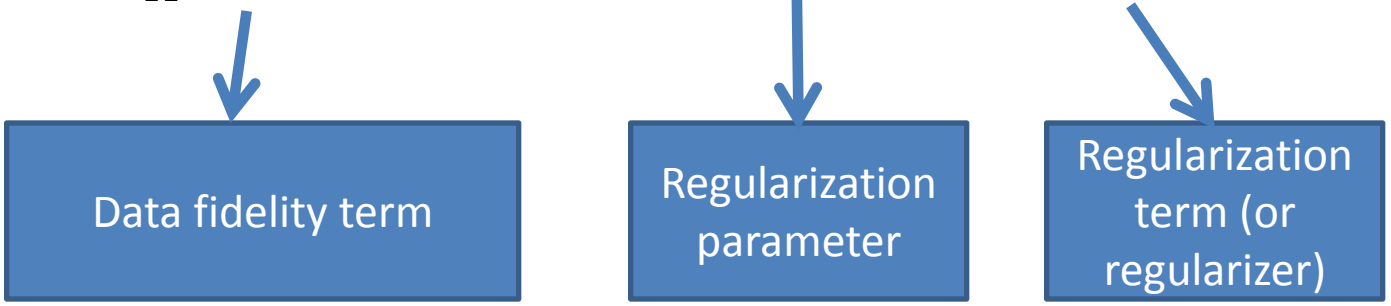
- The addition of the smoothness constraint in the under-constrained problem is called as **regularization**.

- Regularization is a common feature of MANY computer vision problems (and in fact, many problems in machine learning and statistics).

Notice the following!

- Look at the equation again:

$$J(u,v) = \iint_{\Omega} \left((I_x u + I_y v + I_t)^2 + \lambda (u_x^2 + u_y^2 + v_x^2 + v_y^2) \right) dxdy$$



- The parameter $\lambda \geq 0$ performs a weighting between the regularizer and data fidelity term. A larger λ means more weight to the regularizer, and a smaller λ means more weight to the data fidelity term.

- It is usually a user-specified parameter (though there is a large body of literature on automatic choice of λ).

Horn and Shunck: Basic update equations

$$J(u,v) = \iint_{\Omega} \left((I_x u + I_y v + I_t)^2 + \lambda (u_x^2 + u_y^2 + v_x^2 + v_y^2) \right) dx dy$$

$$J(\{u_{i,j}, v_{i,j}\}) = \sum_{i=1}^{N} \sum_{j=1}^{M} \left(\begin{array}{l} (I_{x;i,j} u_{i,j} + I_{y;i,j} v_{i,j} + I_{t;i,j})^2 + \\ \lambda ((u_{i,j+1} - u_{i,j})^2 + (u_{i+1,j} - u_{i,j})^2 + (v_{i,j+1} - v_{i,j})^2 + (v_{i+1,j} - v_{i,j})^2) \end{array} \right)$$

$$\frac{\partial J}{\partial u_{k,l}} = 0 \Rightarrow (I_{x;k,l}^2 + 4\lambda) u_{k,l} + I_{x;k,l} I_{y;k,l} v_{k,l} = 4\lambda \bar{u}_{k,l} - I_{x;k,l} I_{t;k,l}$$

$$\bar{u}_{k,l} = (u_{k,l+1} + u_{k+1,l} + u_{k,l-1} + u_{k-1,l}) / 4$$

$$\frac{\partial J}{\partial v_{k,l}} = 0 \Rightarrow I_{x;k,l} I_{y;k,l} u_{k,l} + (I_{y;k,l}^2 + 4\lambda) v_{k,l} = 4\lambda \bar{v}_{k,l} - I_{y;k,l} I_{t;k,l}$$

$$\bar{v}_{k,l} = (v_{k,l+1} + v_{k+1,l} + v_{k,l-1} + v_{k-1,l}) / 4$$

Replace the integral by a summation. Take derivatives of the energy function *J* w.r.t. the flow vector components (*u* and *v*) at each pixel, and set those derivatives to 0. Note that $I_{x;k,l}$ stands for the derivative of *I* in the x direction at pixel (*k*,*l*)

Horn and Shunck: Basic update equations

$$\frac{\partial J}{\partial u_{k,l}} = 0 \Rightarrow (I_{x;k,l}^2 + 4\lambda)u_{k,l} + I_{x;k,l}I_{y;k,l}v_{k,l} = 4\lambda\bar{u}_{k,l} - I_{x;k,l}I_{t;k,l}$$

$$\frac{\partial J}{\partial v_{k,l}} = 0 \Rightarrow I_{x;k,l}I_{y;k,l}u_{k,l} + (I_{y;k,l}^2 + 4\lambda)v_{k,l} = 4\lambda\bar{v}_{k,l} - I_{y;k,l}I_{t;k,l}$$

Solving the equations simultaneously

$$u_{kl} = \frac{\bar{u}_{kl}(I_{y;k,l}^2 + 4\lambda) - I_{x;k,l}I_{y;k,l}\bar{v}_{kl} - I_{x;k,l}I_{t;k,l}}{I_{x;k,l}^2 + I_{y;k,l}^2 + 4\lambda} = \bar{u}_{kl} - \frac{I_x(I_x\bar{u}_{kl} + I_y\bar{v}_{kl} + I_t)}{I_x^2 + I_y^2 + 4\lambda}$$

$$v_{kl} = \frac{\bar{v}_{kl}(I_{x;k,l}^2 + 4\lambda) - I_{x;k,l}I_{y;k,l}\bar{u}_{kl} - I_{y;k,l}I_{t;k,l}}{I_{x;k,l}^2 + I_{y;k,l}^2 + 4\lambda} = \bar{v}_{kl} - \frac{I_y(I_x\bar{u}_{kl} + I_y\bar{v}_{kl} + I_t)}{I_x^2 + I_y^2 + 4\lambda}$$

Dropping the subscripts (k,l) for more readability

Horn and Shunck: digital derivatives

- The image derivatives are expressed as follows:

$$I_{x;k,l} = I_{x;k,l,t} = (I_{k,l+1,t} - I_{k,l,t} + I_{k+1,l+1,t} - I_{k+1,l,t} + I_{k,l+1,t+1} - I_{k,l,t+1} + I_{k+1,l+1,t+1} - I_{k+1,l,t+1})/4$$

$$I_{y;k,l} = I_{y;k,l,t} = (I_{k+1,l,t} - I_{k,l,t} + I_{k+1,l+1,t} - I_{k,l+1,t} + I_{k+1,l,t+1} - I_{k,l,t+1} + I_{k+1,l+1,t+1} - I_{k,l+1,t+1})/4$$

$$I_{t;k,l} = I_{t;k,l,t} = (I_{k,l,t+1} - I_{k,l,t} + I_{k,l+1,t+1} - I_{k,l+1,t} + I_{k+1,l,t+1} - I_{k+1,l,t} + I_{k+1,l+1,t+1} - I_{k+1,l+1,t})/4$$

- Important: the digital approximations should represent the derivatives at the same point in space and time. They should use the same set of points for the computation as well.

Horn and Shunck: digital derivatives

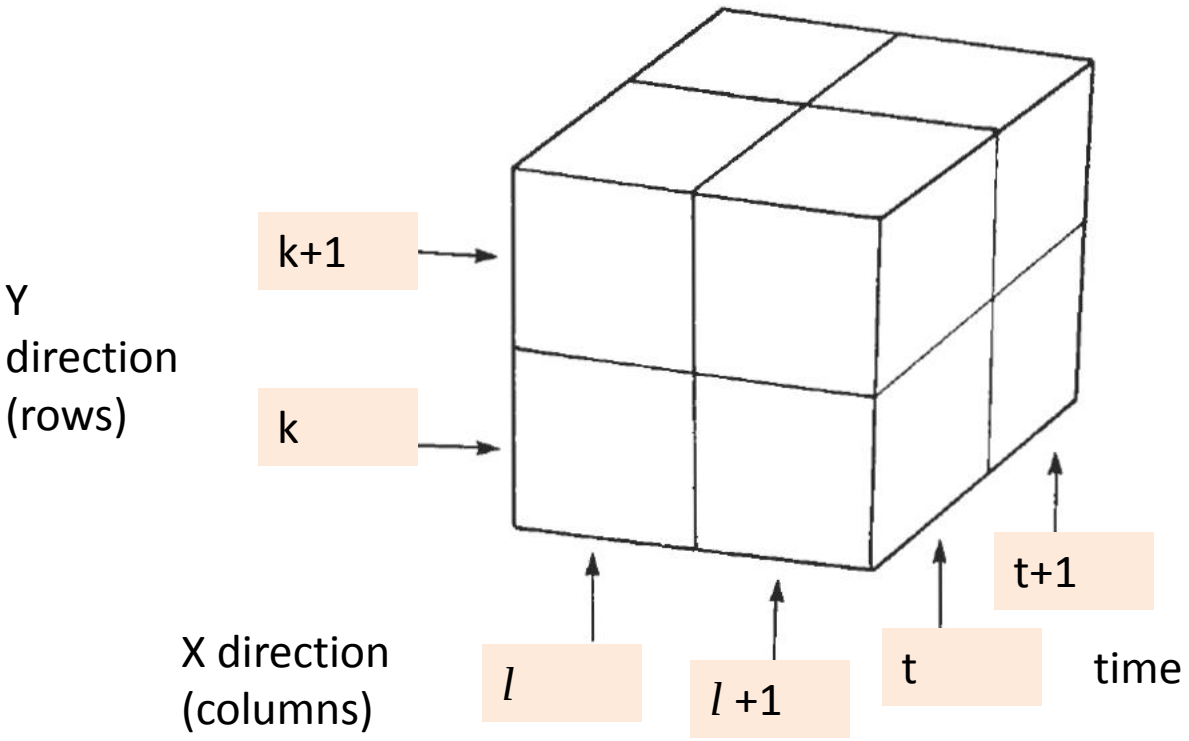


FIG. 2. The three partial derivatives of images brightness at the center of the cube are each estimated from the average of first differences along four parallel edges of the cube. Here the

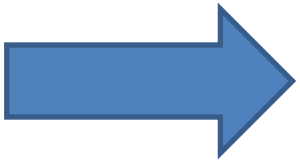
Horn and Shunck: Boundary conditions

- We are taking forward differences: all image derivatives at the right-side or bottom boundary of the image are set to 0.
- Values of *u* and *v* at pixels that fall outside the image domain are set to 0 (and ignored from the computation of the average).

Horn and Shunck: Solution

$$u_{kl} = \bar{u}_{kl} - \frac{I_x(I_x\bar{u}_{kl} + I_y\bar{v}_{kl} + I_t)}{I_x^2 + I_y^2 + 4\lambda}$$

$$v_{kl} = \bar{v}_{kl} - \frac{I_y(I_x\bar{u}_{kl} + I_y\bar{v}_{kl} + I_t)}{I_x^2 + I_y^2 + 4\lambda}$$



Note: If you knew the value of *u* at the each of the four neighbors of pixel *(k,l)*, you can update u_{kl}. Likewise for v_{kl}. But you don't know these values!
There will be 2 such equations per pixel, for a total of 2*MN* equations.

$$u_{kl} - \bar{u}_{kl}\left(1 - \frac{I_x^2}{I_x^2 + I_y^2 + 4\lambda}\right) + \bar{v}_{kl}\left(\frac{I_xI_y}{I_x^2 + I_y^2 + 4\lambda}\right) = \frac{-I_xI_t}{I_x^2 + I_y^2 + 4\lambda}$$

$$v_{kl} - \bar{v}_{kl}\left(1 - \frac{I_y^2\bar{v}_{kl}}{I_x^2 + I_y^2 + 4\lambda}\right) + \bar{u}_{kl}\left(\frac{I_yI_x}{I_x^2 + I_y^2 + 4\lambda}\right) = \frac{-I_yI_t}{I_x^2 + I_y^2 + 4\lambda}$$

We can re-arrange all these equations collecting together terms involving *u* and *v* at any pixel location on the LHS, and writing down terms without *u* and *v* on the RHS.

Hence, these equations in the form **Ax** = **b** where **A** is a 2*MN* x 2*MN* matrix, and **x** and **b** are 2*MN* x 1 vectors. Vector **x** will contain all the unknowns, i.e. *u* and *v* values at a pixel (k,l) and its neighbors. Vector **b** will contain the (known) terms on the RHS of these equations. Matrix **A** is sparse and contains at the most 9 non-zero values in each row (why?).

Horn and Shunck: Solution

- Note that such a system is extremely expensive to invert.
- Initial condition for **u** and **v** – zeros; boundary conditions for derivatives of **I** – zero across boundary of the image
- Iterative algorithm (Jacobi's method): run for *T* iterations or until convergence: Update the optical flow at one pixel keeping the optical flow at all other pixels fixed.

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

Applying the Jacobi method leads to updates of the following form at the *t*-th iteration (see next slide as well):

$$u_{kl}^{(t+1)} = \bar{u}_{kl}^{(t)} - \frac{I_x\left(I_x\bar{u}_{kl}^{(t)} + I_y\bar{v}_{kl}^{(t)} + I_t\right)}{I_x^2 + I_y^2 + 4\lambda}$$

$$v_{kl}^{(t+1)} = \bar{v}_{kl}^{(t)} - \frac{I_y\left(I_x\bar{u}_{kl}^{(t)} + I_y\bar{v}_{kl}^{(t)} + I_t\right)}{I_x^2 + I_y^2 + 4\lambda}$$

Segway: Jacobi's method

- Consider system of equations **Ax** = **b**.
- **A** can be written as **D**+**R** where **D** is a diagonal matrix – as follows:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & . & . & a_{1n} \\ a_{21} & a_{22} & & & . \\ . & & & & . \\ . & & & & . \\ a_{n1} & a_{n2} & . & . & a_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & 0 & . & . & 0 \\ 0 & a_{22} & & & . \\ . & & & & . \\ . & & & & . \\ 0 & a_{n2} & . & . & a_{nn} \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & . & . & a_{1n} \\ a_{21} & 0 & & & . \\ . & & & & . \\ . & & & & . \\ a_{n1} & a_{n2} & . & . & 0 \end{pmatrix}$$

Diagonal matrix **D**

Remainder matrix **R**

- The solution vector is iteratively obtained as

$$\mathbf{x}^{(t+1)} = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{R}\mathbf{x}^{(t)})$$

$$\mathbf{x}_i^{(t+1)} = \frac{\mathbf{b}_i - \sum_{j \neq i} A_{ij}\mathbf{x}_j^{(t)}}{A_{ii}}$$

Comments on Horn and Shunck (+)

- Optical flow is completely indeterminate in regions where intensity is constant.

- But this method allows for **filling in** of the flow vectors in such regions, because of the **regularizer** term.

- The update of u (or v) at pixel (k,l) at iteration t+1 does not depend on u_{kl} or v_{kl} in iteration t, but it depends only the neighboring values of u or v.

Comments on Horn and Shunck (-?)

- Note that the method banks on the brightness constancy assumption.
- It also assumes that the flow at all points is small in magnitude (otherwise the basic constancy equation doesn't hold).
- It also assumes smoothness of the flow (this can get violated if two independently moving objects come close together in a video frame).

Horn and Shunck: Laplacian of u and v

$$\frac{\partial J}{\partial u_{k,l}} = 0 \Rightarrow (I_{x;k,l}^2 + 4\lambda)u_{k,l} + I_{x;k,l}I_{y;k,l}v_{k,l} = 4\lambda\bar{u}_{k,l} - I_{x;k,l}I_{t;k,l}$$

$$\frac{\partial J}{\partial v_{k,l}} = 0 \Rightarrow I_{x;k,l}I_{y;k,l}u_{k,l} + (I_{y;k,l}^2 + 4\lambda)v_{k,l} = 4\lambda\bar{v}_{k,l} - I_{y;k,l}I_{t;k,l}$$

$$(I_{x;k,l}^2)u_{k,l} + I_{x;k,l}I_{y;k,l}v_{k,l} = 4\lambda(\bar{u}_{k,l} - u_{k,l}) - I_{x;k,l}I_{t;k,l}$$

$$\therefore (I_{x;k,l}^2)u_{k,l} + I_{x;k,l}I_{y;k,l}v_{k,l} = 4\lambda(\Delta u_{k,l}) - I_{x;k,l}I_{t;k,l}$$

$$\therefore I_{x;k,l}I_{y;k,l}u_{k,l} + (I_{y;k,l}^2)v_{k,l} = 4\lambda(\Delta v_{k,l}) - I_{y;k,l}I_{t;k,l}$$

Laplacian of **u** and **v** using the well-known mask (see next slide). You may also use other masks for the Laplacian in which case these update equations change slightly.

Laplacian of a 2D signal f

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Rotationally symmetric operator (in the continuous domain)

$$= f(x+1, y) + f(x-1, y) - 2f(x, y)$$
$$+ f(x, y+1) + f(x, y-1) - 2f(x, y)$$
$$= f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Laplacian operators: second operator is obtained by adding second derivatives along both the diagonals, to the first operator

Horn and Shunck: Laplacian

- The original paper by Horn and Shunck suggests the following mask for the Laplacian of **u** and **v**:

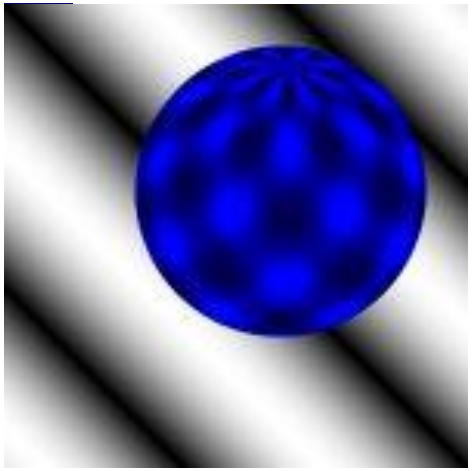
$$12\begin{pmatrix} \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \\ \frac{1}{6} & -1 & \frac{1}{6} \\ \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \end{pmatrix}$$

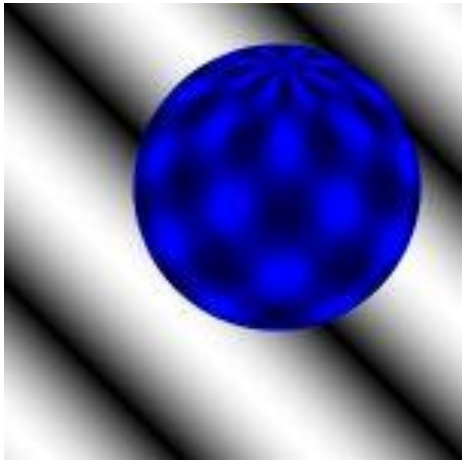
Laplacian of an image

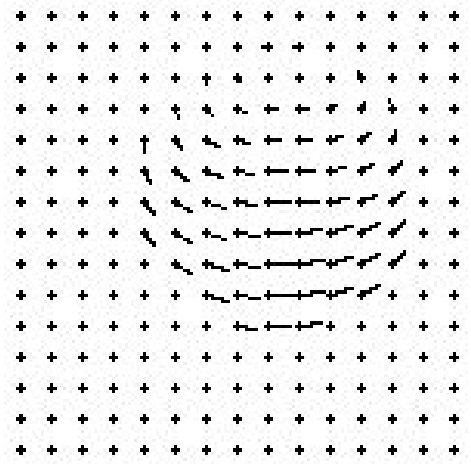
- The two masks on the previous slide are applied point-wise to the entire image.
- For image smoothing as well, we applied point-wise masks.
- But here the mask values sum to 0. In smoothing the weights summed to 1.

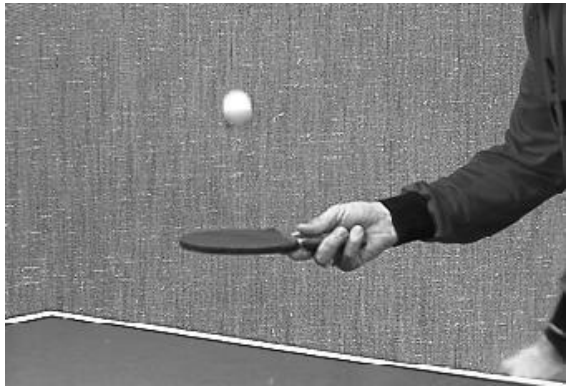
Horn and Shunck: examples

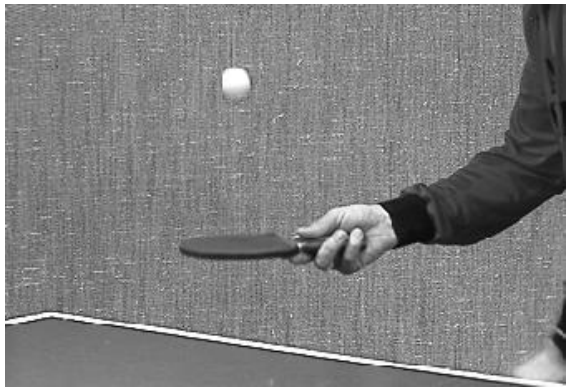
http://of-eval.sourceforge.net/











Two frames from the so-called “table tennis” video sequence

The next slide contains a quiver plot of the optical flow vectors. A clearer plot can be observed in the following pdf:

http://www.cse.iitb.ac.in/~ajitvr/CS763_Spring2016/hs_result.pdf

A quiver plot is a plot of vectors (u(x,y),v(x,y)) in 2D. The vector at a pixel (x,y) is represented by an arrow whose length is proportional to the vector magnitude. The quiver plot is generated by the MATLAB function `quiver’.

Horn and Shunck: one last point

- The original paper by Horn-Shunck uses the **calculus of variations** to minimize the functional J(u,v) w.r.t. u and v.

- Calculus of variations is a branch of mathematics that asks the following question: Which function f minimizes a functional E(f)? (eg: which curve passing through two points has the least length?)

- In our lectures, we have used a discrete form for J(u,v) and directly minimized it using simple derivatives.

- The resulting equations are very similar to the ones derived using calculus of variations (though there are problems where using calculus of variations is of special use!).

Lucas-Kanade

- Is a **local** (semi-local) method – it solves several small problems independently, whereas Horn and Shunck solves one large (global) problem.
- Assume that the optical flow is **constant** within a small window (say, of size N x N, N is around 5 to 8).
- Uses a **least squares** approach to solve for the optical flow by combining together several (N^2 in case of N x N window) brightness constancy equations.

Lucas-Kanade

$$\forall i, 1 \le i \le N^2, u(x_i, y_i, t) I_x(x_i, y_i, t) + v(x_i, y_i, t) I_y(x_i, y_i, t) = -I_t(x_i, y_i, t)$$

'(x_i, y_i)' denotes the i^{th} spatial location (i.e. pixel) inside a small N x N window

Note that u and v are constant over this window.

$$\begin{pmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \cdot & \cdot \\ \cdot & \cdot \\ I_{xN^2} & I_{yN^2} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \approx -\begin{pmatrix} I_{t1} \\ I_{t2} \\ \cdot \\ \cdot \\ I_{tN^2} \end{pmatrix}$$

$$\mathbf{A}\mathbf{w} \approx \mathbf{b}$$

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

This is the **least squares** solution (which we obtain by the **pseudo-inverse**). It is the solution to an **over-constrained** problem, i.e. a problem where the number of knowns is more than the number of unknowns (in this case, 2).

This solution minimizes the following quantity:

$$J(u,v) = \sum_{i=1}^{N^2} (uI_{xi} + vI_{yi} + I_{ti})^2$$

$$\therefore \mathbf{w} = \begin{pmatrix} \sum_{i=1}^{N^2} I_{xi}^2 & \sum_{i=1}^{N^2} I_{xi} I_{yi} \\ \sum_{i=1}^{N^2} I_{xi} I_{yi} & \sum_{i=1}^{N^2} I_{yi}^2 \end{pmatrix}^{-1} \begin{pmatrix} -\sum_{i=1}^{N^2} I_{xi} I_{ti} \\ -\sum_{i=1}^{N^2} I_{yi} I_{ti} \end{pmatrix}$$

This 2 x 2 matrix is called as a **local structure-tensor** or **local second moment matrix**.

Lucas-Kanade

- This solution minimizes the following quantity:

$$J(u,v) = \sum_{i=1}^{N^2} (uI_{xi} + vI_{yi} + I_{ti})^2$$

- Taking partial derivatives and setting them to zero yields the following set of simultaneous equations (the exact same as before):

$$\sum_{i=1}^{N^2} (uI_{xi} + vI_{yi} + I_{ti}) I_{xi} = 0$$

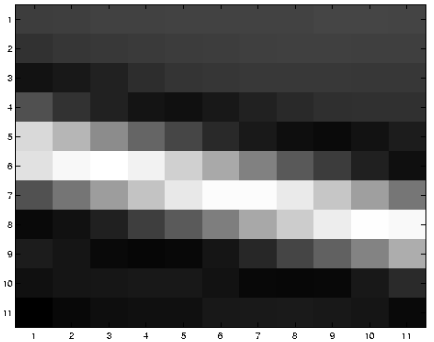
$$\sum_{i=1}^{N^2} (uI_{xi} + vI_{yi} + I_{ti}) I_{yi} = 0$$

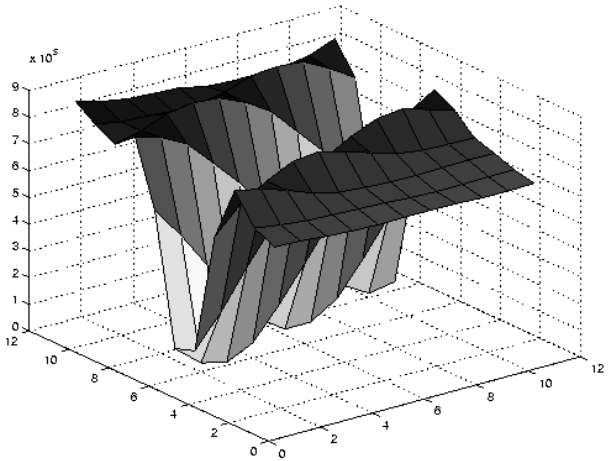
Note that *u* and *v* in these equations are without indices because they are assumed **constant** for the **entire** *N* x *N* window.

Lucas-Kanade

- Note: There is an assumption that the matrix $\mathbf{A}^T\mathbf{A}$ (i.e. structure tensor) is invertible!
- This assumption fails if all gradients in a region are aligned (i.e. a region of constant shading, i.e. of the form $I(x,y)=ax+by+c$) or if all gradients are null (constant intensity, i.e. $I(x,y) = c$).
- Former case: just use the normal flow (i.e. flow component along gradient direction) and interpolate from surrounding regions.
- Latter case: optical flow is determined by interpolation from surrounding regions.

Edges cause problems



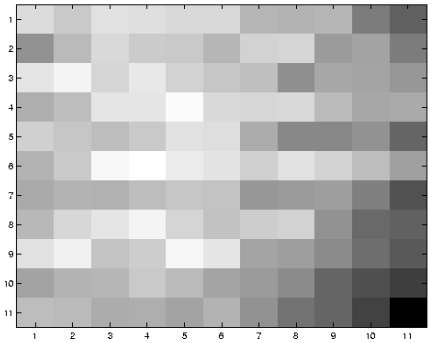


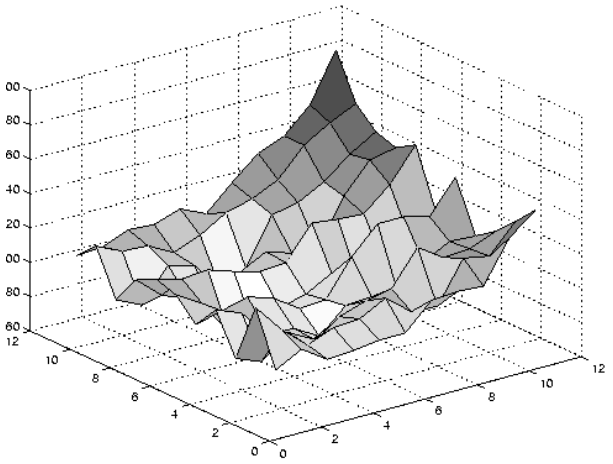
$$\sum \nabla I (\nabla I)^T = V \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} V^T$$

– large gradients, all the same

– large λ_1, small λ_2 (eigenvalues of the structure tensor)

Low texture regions don't work



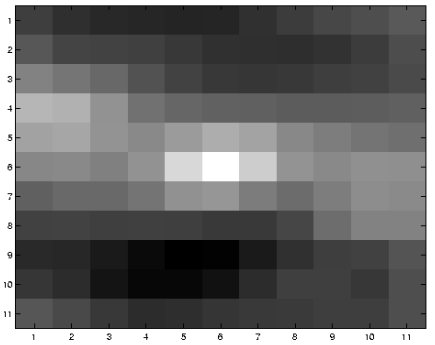


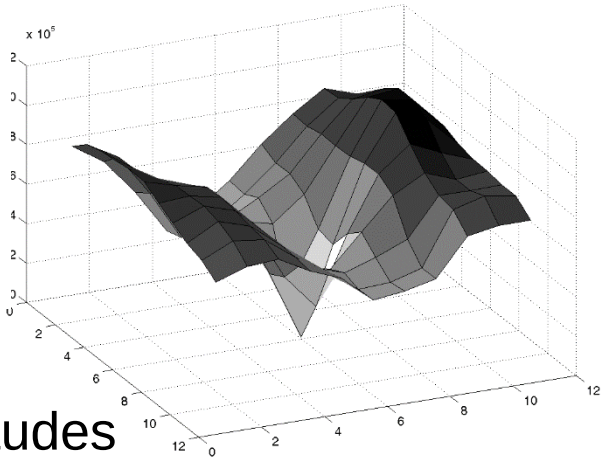
$$\sum \nabla I (\nabla I)^T$$

– gradients have small magnitude

– small λ_1, small λ_2

High textured region work best





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

– gradients are different, large magnitudes

– large λ_1, large λ_2

Lucas-Kanade

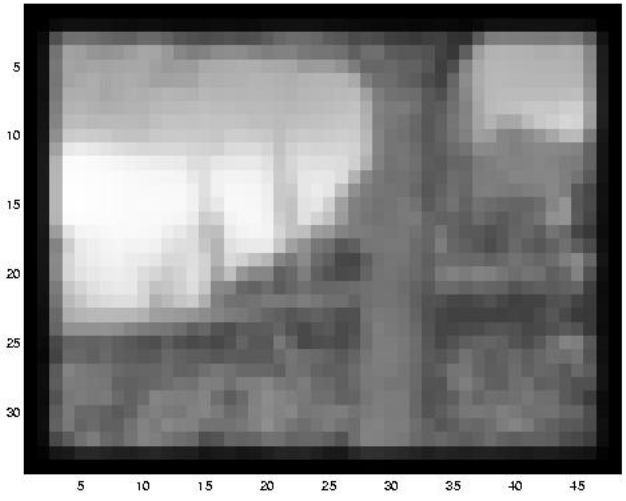
- In practice, better results are obtained by smoothing the images prior to computation of optical flow.
- The method assumes small valued flow between consecutive frames. If this assumption fails, then one option is to run the algorithm on downsampled images, and then upsample the computed flow.
- What will happen if window is too large? Too small?

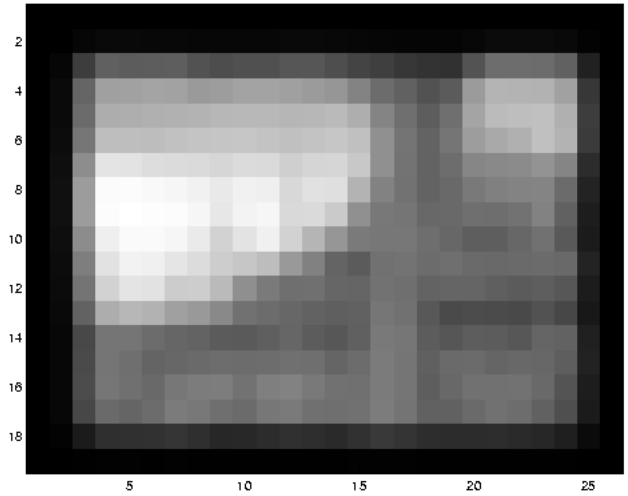
Slides taken from the course by Steve Seitz at University of Washington,

Revisiting the small motion assumption

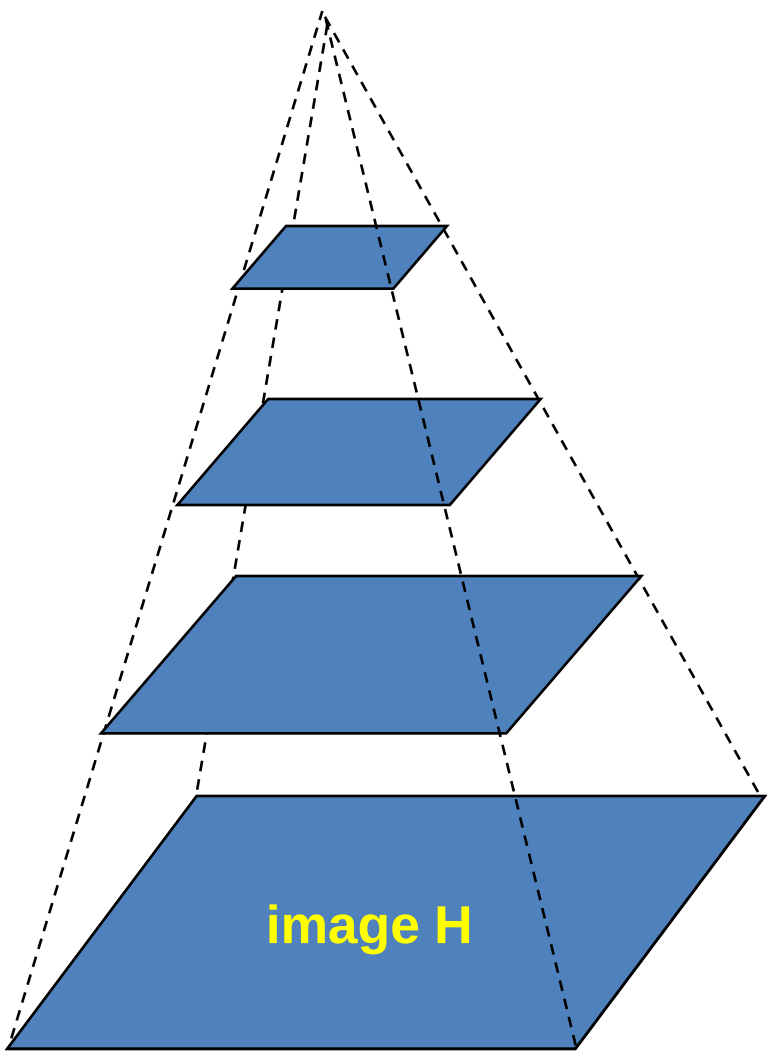
- Is this motion small enough?
  - Probably not—it's much larger than one pixel (2nd order terms dominate)
  - How might we solve this problem?

Reduce the resolution!





Coarse-to-fine optical flow estimation



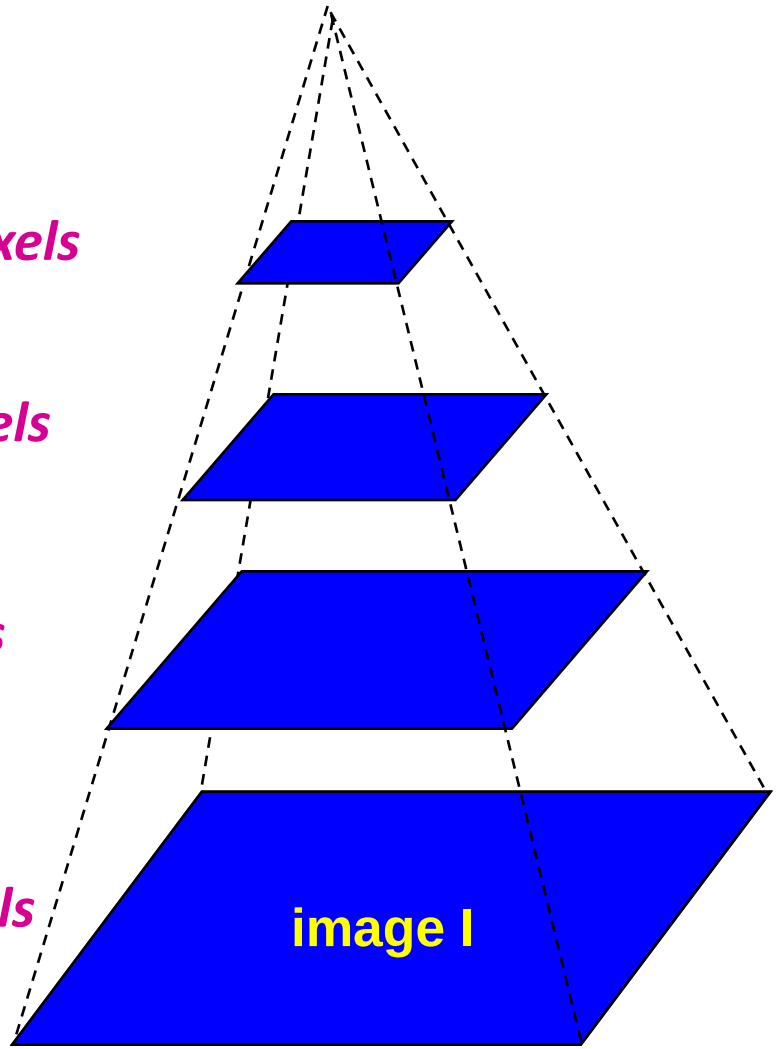
Gaussian pyramid of image H

u=1.25 pixels

u=2.5 pixels

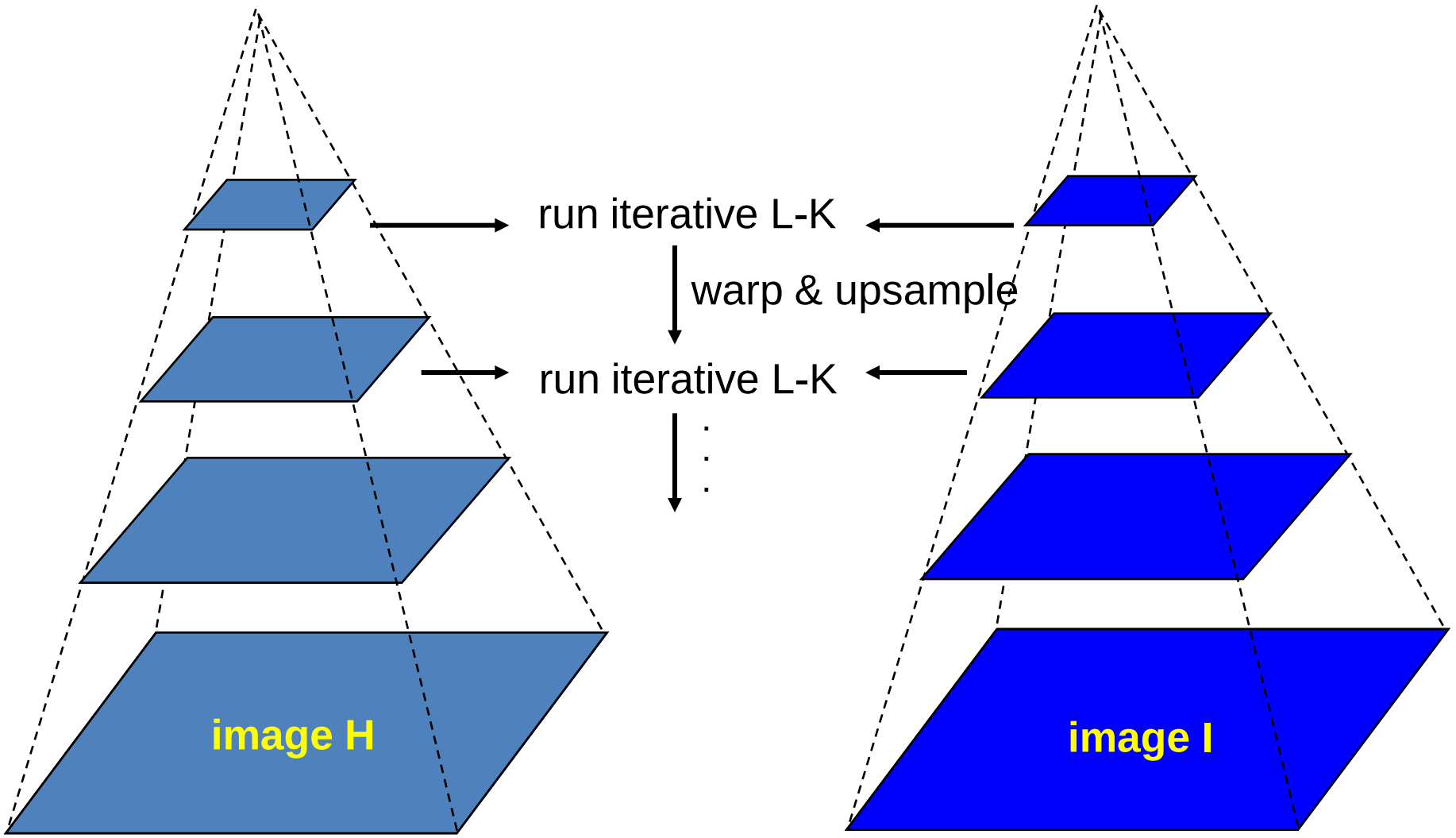
u=5 pixels

u=10 pixels



Gaussian pyramid of image I

Coarse-to-fine optical flow estimation



Gaussian pyramid of image H

Gaussian pyramid of image I

A Few Details

- Top Level
  - Apply L-K to get a flow field representing the flow from the first frame to the second frame.
  - Apply this flow field to warp the first frame toward the second frame.
  - Rerun L-K on the new warped image to get a flow field from it to the second frame.
  - Repeat till convergence.
- Next Level
  - Upsample the **flow field** to the next level as the first guess of the flow at that level.
  - Apply this flow field to warp the first frame toward the second frame.
  - Rerun L-K and warping till convergence as above.
- Etc.

Comments on Lucas-Kanade (+)

- In some windows, the optical flow estimate is more reliable than at other windows. This reliability depends on the rank of the structure tensor (and does not require any information about the next frame, i.e. *I(:,:,t+1)*!).
- Errors in estimates in one part of the image do not affect the estimates in another part.

H-S versus L-K

- H-S is global, L-K is local.
- Both have parallelizable implementations.
- H-S gives an optical flow value at all pixels regardless of gradient. L-K requires a post-processing step to interpolate flow in case of regions with aligned gradients or null gradients.
- H-S required selection of λ, L-K requires selection of window-size.
- H-S is more noise-sensitive, as errors in one part of the image propagate all over. In L-K, the optical flow estimates in different windows are generally independent.
- H-S (without modifications) has no inbuilt reliability estimate unlike L-K.

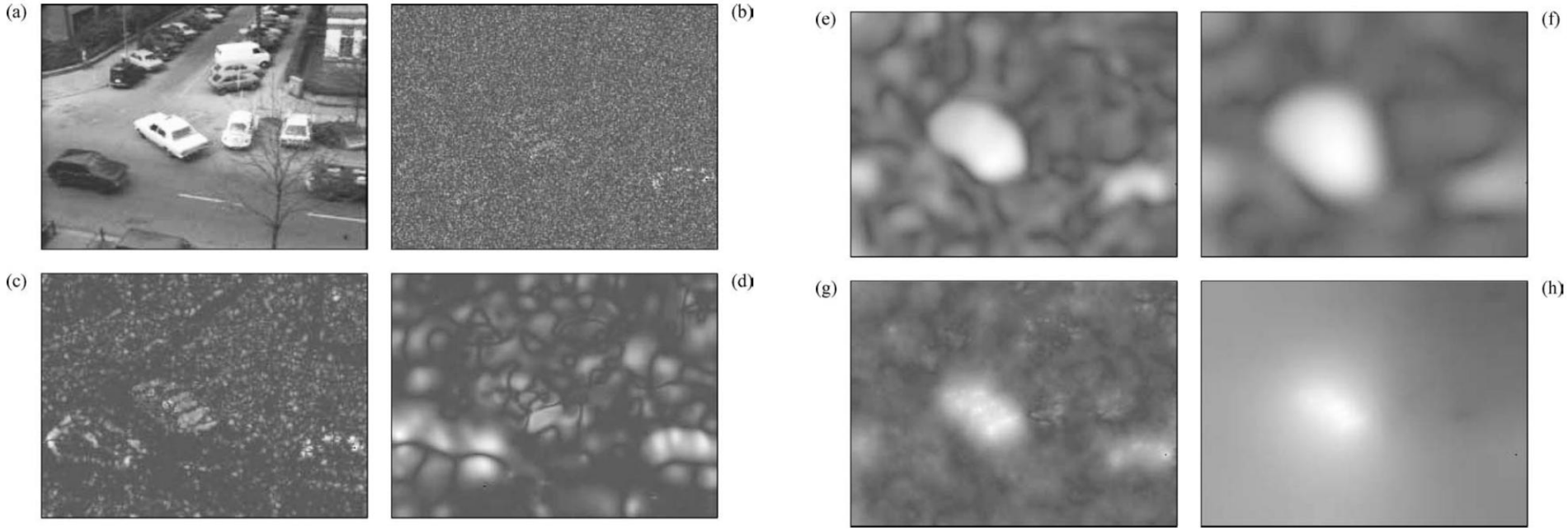


Figure 1. From left to right, and from top to bottom: (a) Frame 10 of the Hamburg taxi sequence, where Gaussian noise with standard deviation $\sigma_n = 10$ has been added. The white taxi turns around the corner, the left car drives to the right, and the right van moves to the left. (b) Normal flow magnitude without presmoothing. (c) Normal flow magnitude, presmoothing with $\sigma = 1$. (d) Ditto, presmoothing with $\sigma = 5$. (e) Lucas-Kanade method with $\sigma = 0$, $\rho = 7.5$. (f) Ditto, $\sigma = 0$, $\rho = 15$. (g) Optic flow magnitude with the Horn-Schunck approach, $\sigma = 0$, $\alpha = 10^5$. (h) Ditto, $\sigma = 0$, $\alpha = 10^6$.

Image source: “Lucas-Kanade meets Horn-Shunck: combining global and local optical flow methods”, Bruhn, Weickert, Schnorr, IJCV 2005.

Image warping using computed optical flow

- Consider images I_1 and I_2.
- Let u(x,y) and v(x,y) be the computed optical flow between them such that $I_1(x+u(x,y),y+v(x,y)) = I_2(x,y)$.
- **Forward warping method:** Copy intensity value $I_1(x,y)$ to $I_2(round(x+u(x,y)),round(y+v(x,y)))$.
- Here 'round' refers to the operation of rounding off to the nearest integer.
- Problem: some pixels may remain unoccupied after warping, or two pixels from I_1 may map onto the same pixel in I_2.

Image warping using computed optical flow

- Reverse warping method: For every pixel (x,y) in the output image (i.e. warped version of I1), determine which pixels in I1 map somewhere in the range [x-1,x+1] x [y-1,y+1].
- Interpolate the flow field values at these pixels to determine which subpixel will map onto (x,y).

Aperture Problem: Visual Perception

- Images are formed on the retina. Image processing and analysis occurs in the brain (visual cortex).
- The basic (low-level) processing occurs in an area of the visual cortex called V1.
- Advanced (high-level) processing occurs in an area of the visual cortex called V5.

Aperture Problem: Visual Perception

- Each neuron in the V1 area is sensitive to visual stimuli in a small part of the visual field, i.e. as if it were “looking” through a small aperture.
- The motion direction will be ambiguous and defined only in the direction of the gradient of the image – **aperture problem**.
- Thus edges moving in many different ways may give rise to identical stimulus to the V1 neurons.
- It is the job of the V5 neurons to integrate the local motion estimates to produce global estimates.

Aperture Problem

- Barber-pole illusion

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

Optical Flow versus True Motion

- Consider a uniform sphere rotating about its axis under constant illumination. Optical flow is zero, true motion is not!
- Consider a still sphere under lighting of varying direction. Optical flow is not zero, true motion is zero!

Applications: Faces

- Tracking salient feature points (see video above) across frames of a video.

  http://www.cs.toronto.edu/~dross/ivt/dudek.avi

- Facial expression recognition – Input: video of a person's face, Output: facial expression (smile, frown, surprise, anger, sorrow, etc.)

  http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5226597&tag=1

- Does the web-cam see a real face? Or a mask/poster of a face?

Applications: Medical imaging

- Analysis of the motion of a deformable object (Example: biological cell, micro-organism, or an organ like the heart).

http://www.youtube.com/watch?v=JA0Wb3gc4mE

http://www.mate.tue.nl/mate/pdfs/10121.pdf

Applications: Structure from Motion!

- Consider a video sequence of an object undergoing constant rigid motion (translation/rotation).
- Assume orthographic projections.
- Suppose we tracked some **N** feature points across all the frames (using optical flow!)
- It turns out you can make some estimates of the **3D coordinates** of those points in every frame as well as the **motion**. This is called **structure from motion** (SfM).

Applications: Collision Avoidance/Prediction of Collision Time

- Given a video sequence containing two independently moving objects.
- Compute optical flow at each frame.
- Extrapolate the trajectory of salient points on each object.
- Will the images of the objects collide? If so, when?
- Will the actual objects collide? If so, when? (requires calibrated camera)

References

- Horn and Shunck, “Determining optical flow”, http://people.csail.mit.edu/bkph/papers/Optical_Flow_OPT.pdf
- Horn and Shunck explained in detail: http://www.ipol.im/pub/art/2013/20/article_lr.pdf
- http://en.wikipedia.org/wiki/Lucas%E2%80%93Kanade_method
- http://en.wikipedia.org/wiki/Horn%E2%80%93Schunck_method
- http://en.wikipedia.org/wiki/Optical_flow