Image Alignment

- Find the transformation between two images
  - Translation, Rotation, zoom
  - Affine, Homography
  - Assumption: Static Scene, No 3D effects, e.g. motion parallax
- Good for:
  - Video Stabilization, Video Mosaicing, Noise Cleaning…



Parametric (Global) Transformations

Warps by changing camera parameters (except affine):

1 point correspon

ts

2 points

Translation (2 params – t_x, t_y)
Preserves <u>distances</u>, <u>angles</u>

Scaling/zoom (4 params – t_x, t_y, s_x, s_y) Preserves <u>angles</u>

Rotation (3 params – t_x, t_y, θ)
Preserves <u>distances</u>

Most **Affine** cannot be generated by changing camera parameters

3 points

4 points

Affine (6 parameters)
Keeps <u>parallel</u> lines

Projective / Homography (8 parameters) Keeps <u>straight</u> lines

Parametric (Global) Transformations

Warps by changing camera parameters (except affine):

1 point correspondence

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation (2 params – t_x, t_y)
Preserves distances, angles

2 points

$$\begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling/zoom (4 params – t_x, t_y, s_x, s_y) Preserves angles

2 points

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation (3 params – t_x, t_y, θ)
Preserves distances

3 points

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

Affine (6 parameters)
Keeps parallel lines

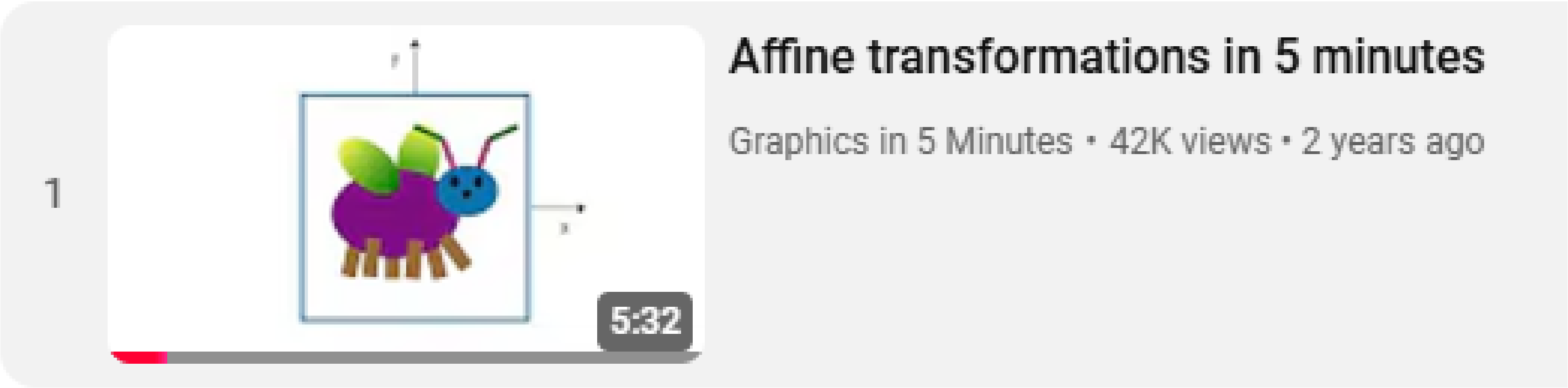
4 points

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

Projective / Homography (8 parameters) Keeps straight lines

Videos to Watch (by Steve Seitz)

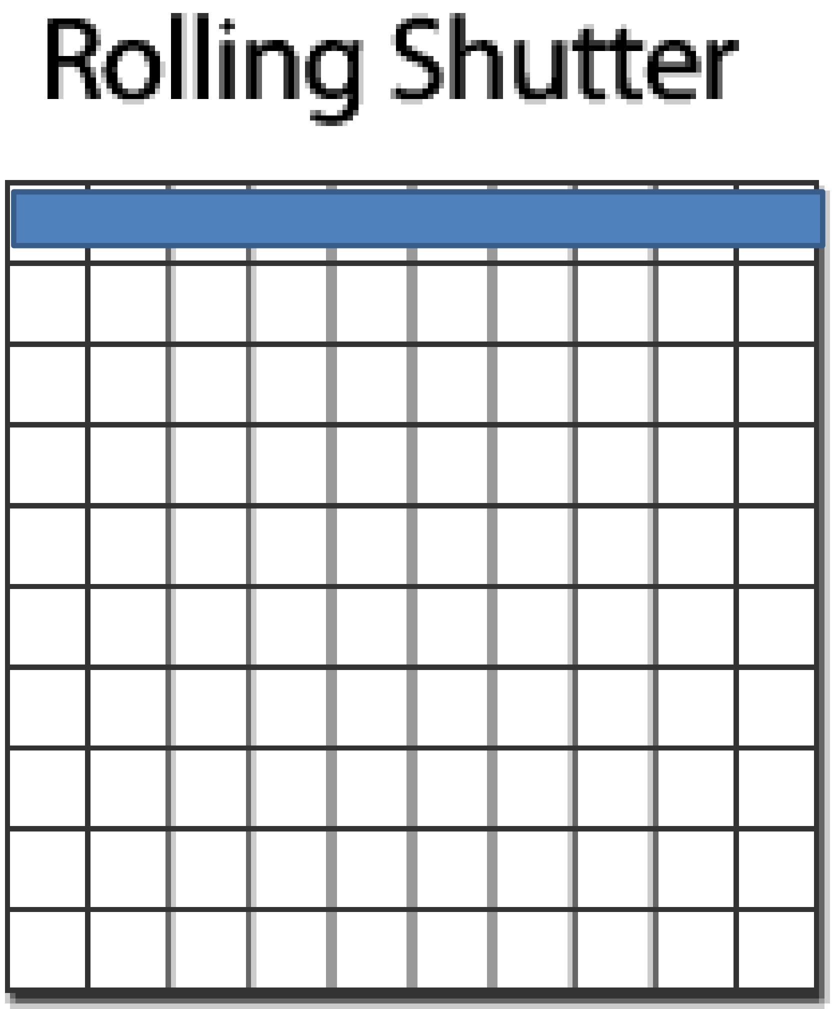
https://www.youtube.com/playlist?list=PLWfDJ5nla8UpwShx-lzLJqcp575fKpsSO



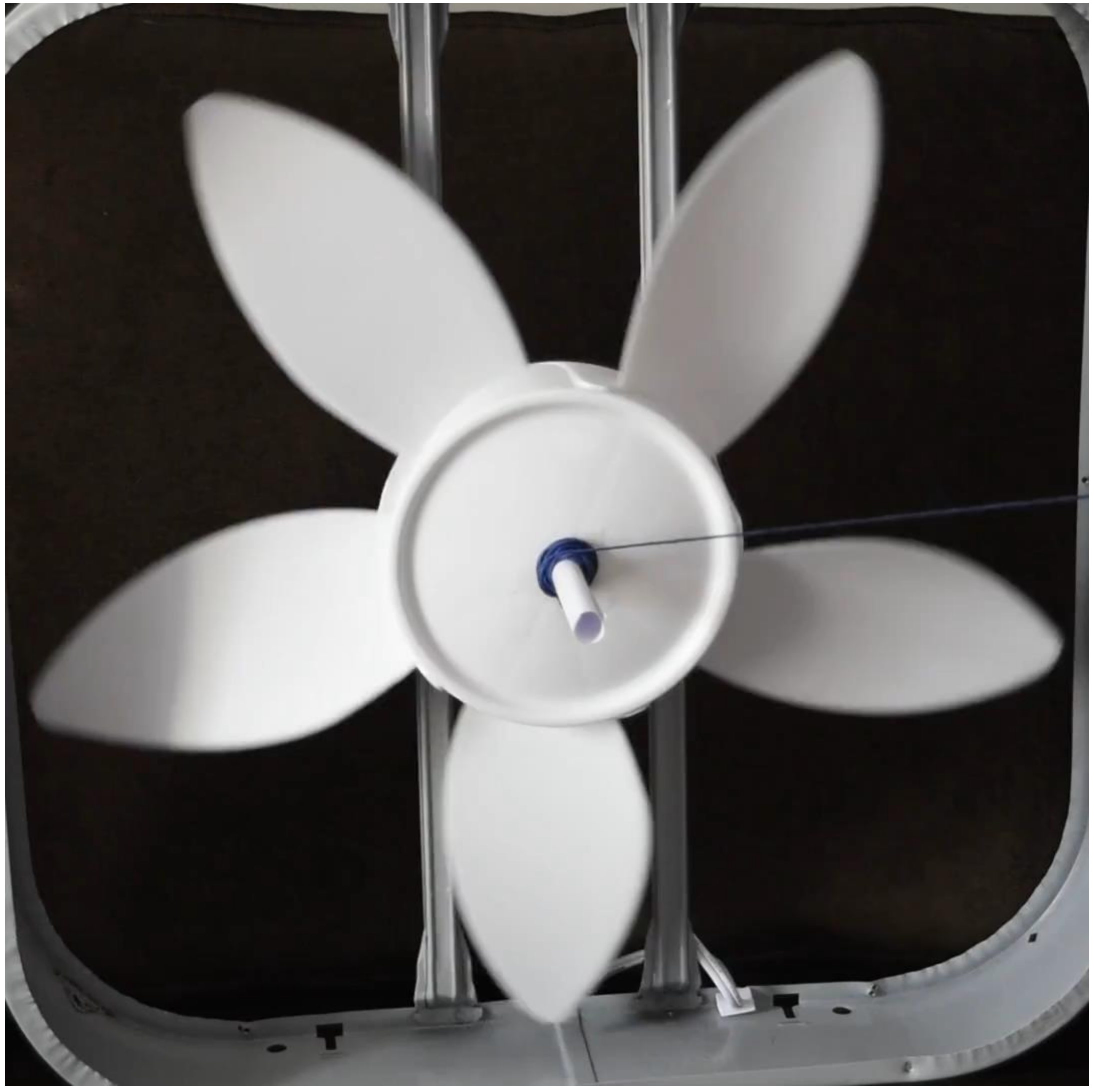
1) Affine Transformations

We Ignore the Rolling Shutter Effect

- Film camera: The entire frame is recorded at the same time.
- First digital cameras (**CCD**): Entire frame recorded at same time
- Newer digital cameras (**CMOS**, all phones): Each line is recorded at a different time (Rolling Shutter)
- 24 fps, 1000 lines, gives 24,000 lines per second for 1K×1K image!



Rolling Shutter Simulation



- 24 fps, 1000 lines, gives 24,000 lines per second!
- In order to get the effect at left, exposure time must be very small.
- Long exposure time, e.g. 1/24 sec, will give blur without the desired effect.

Moving Camera & Objects

Driving Car (What direction? Can we compute distance of buildings)

Rotating propeller

Distances from a single image of a moving camera

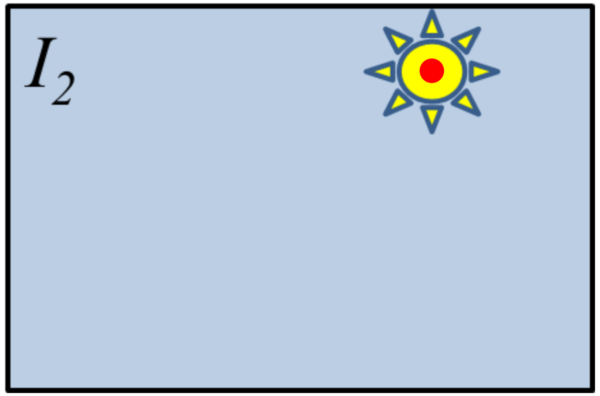
Closer objects move faster in the image, appear more slanted

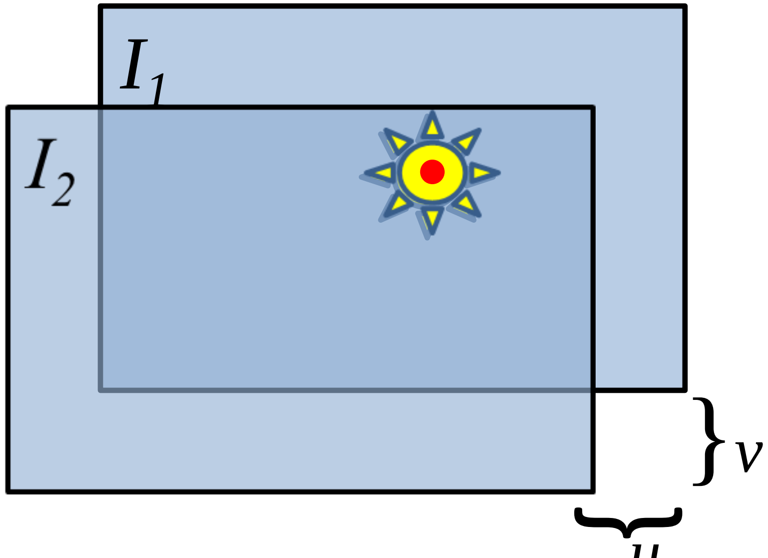
High-Speed Photography

24,000 lines per second… Short Exposure Needed

Computing Global Translation, <u>Point Correspondences</u>

- Assume we can find corresponding point pairs between two images. Compute translation from these point correspondences





Computing Global Translation, <u>Direct Methods</u>

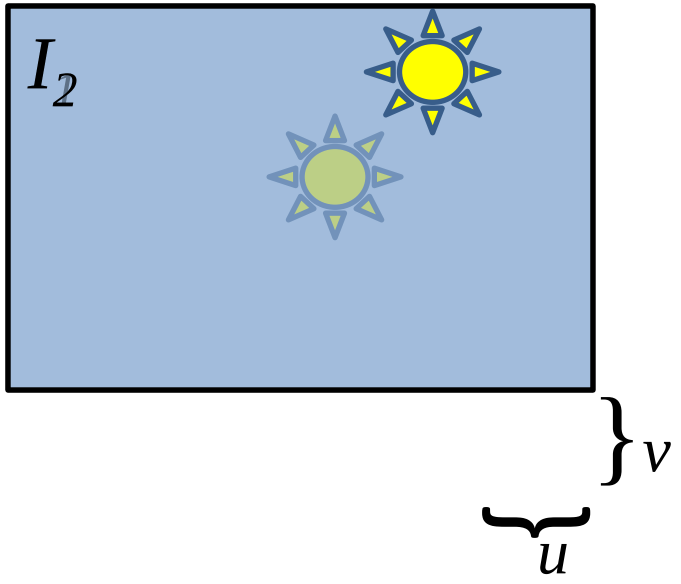
Assumption: Constant Brightness, No Rolling Shutter

- Given images I_1 and I_2, we can find the translation (u,v) that will minimize the squared error

$$E(u,v) = \sum_x \sum_y \left(I_1(x,y) - I_2(x+u, y+v)\right)^2$$

- Implementation: Use average <u>per-pixel</u> error only over <u>area of overlap</u>

- Can also search for rotations: (u,v,α)



Problem with Point Correspondences

- Wrong Correspondences
- No correspondences

Cross Correlation as Match Measure

- Starting from the *SSD* (Sum of Squared Differences)

$$E(u,v) = \sum_x \sum_y (I_1(x,y) - I_2(x+u, y+v))^2$$

- Since $\quad (a-b)^2 = a^2 - 2ab + b^2$
- We can write

$$E(u,v) = \sum_x \sum_y {I_1}^2 - 2 \sum_x \sum_y I_1(x,y) \cdot I_2(x+u, y+v) + \sum_x \sum_y {I_2}^2$$

- Since $\Sigma\, {I_1}^2$ and $\Sigma\, {I_2}^2$ are almost constant, minimizing the *SSD* maximizes the cross-correlation $\Sigma\, I_1 I_2$

$$CC(u,v) = \sum_x \sum_y I_1(x,y) \cdot I_2(x+u, y+v)$$

Familiar?

Normalized Cross Correlation (NCC)

- Given two images I_1 and I_2, search for the translation (x, y) maximizing the cross-correlation

$$CC(u,v) = \sum_x \sum_y I_1(x,y) \cdot I_2(x+u, y+v)$$

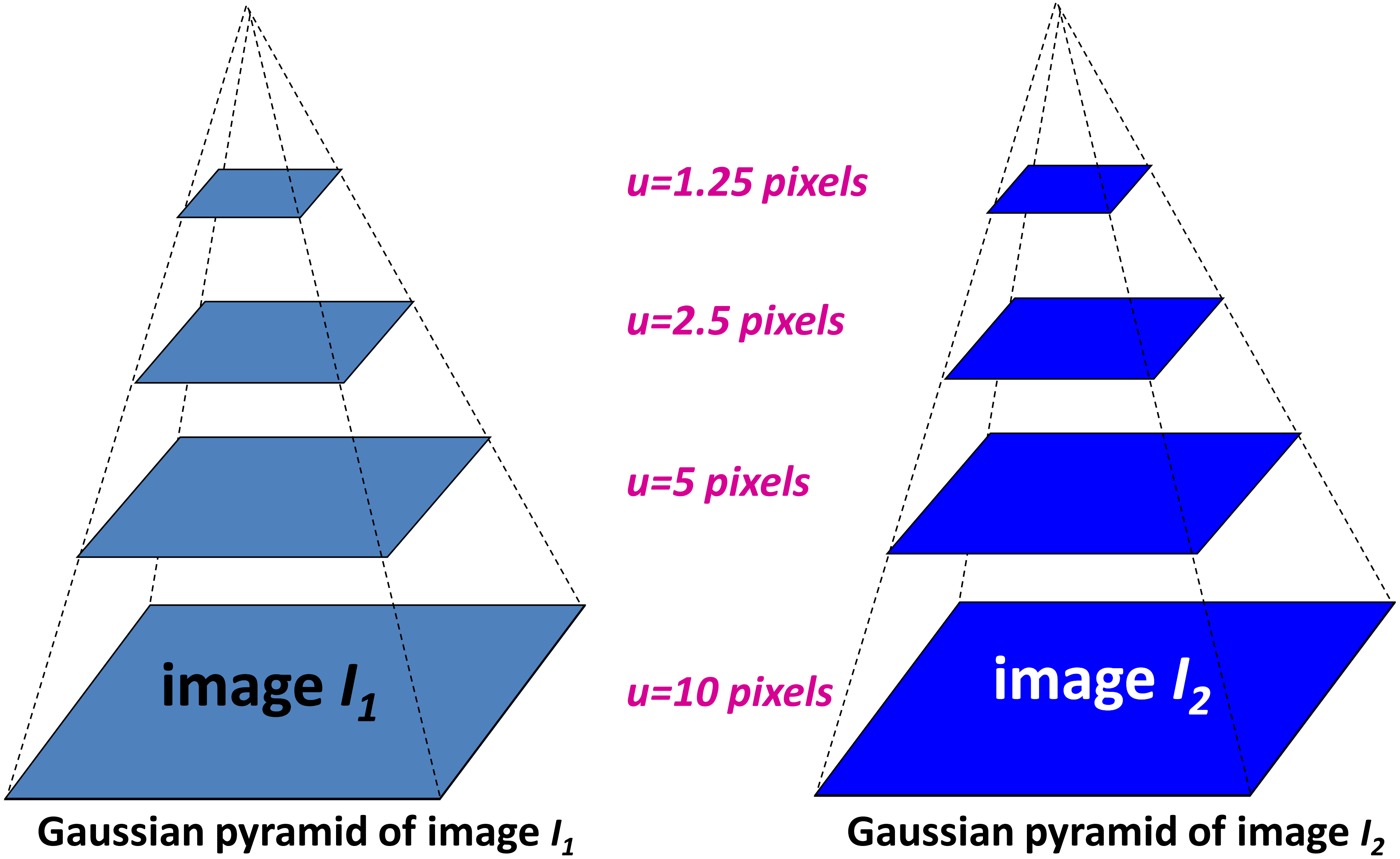
- NCC <u>invariant to global addition and multiplication</u> of intensity ($I_2 = a \cdot I_1 + b$)

$$NCC(u,v) = \frac{\sum (I_1(x,y) - \hat{I}_1) \cdot (I_2(x+u, y+v) - \hat{I}_2)}{\sqrt{\sum (I_1(x,y) - \hat{I}_1)^2} \sqrt{\sum (I_2(x,y) - \hat{I}_2)^2}}$$

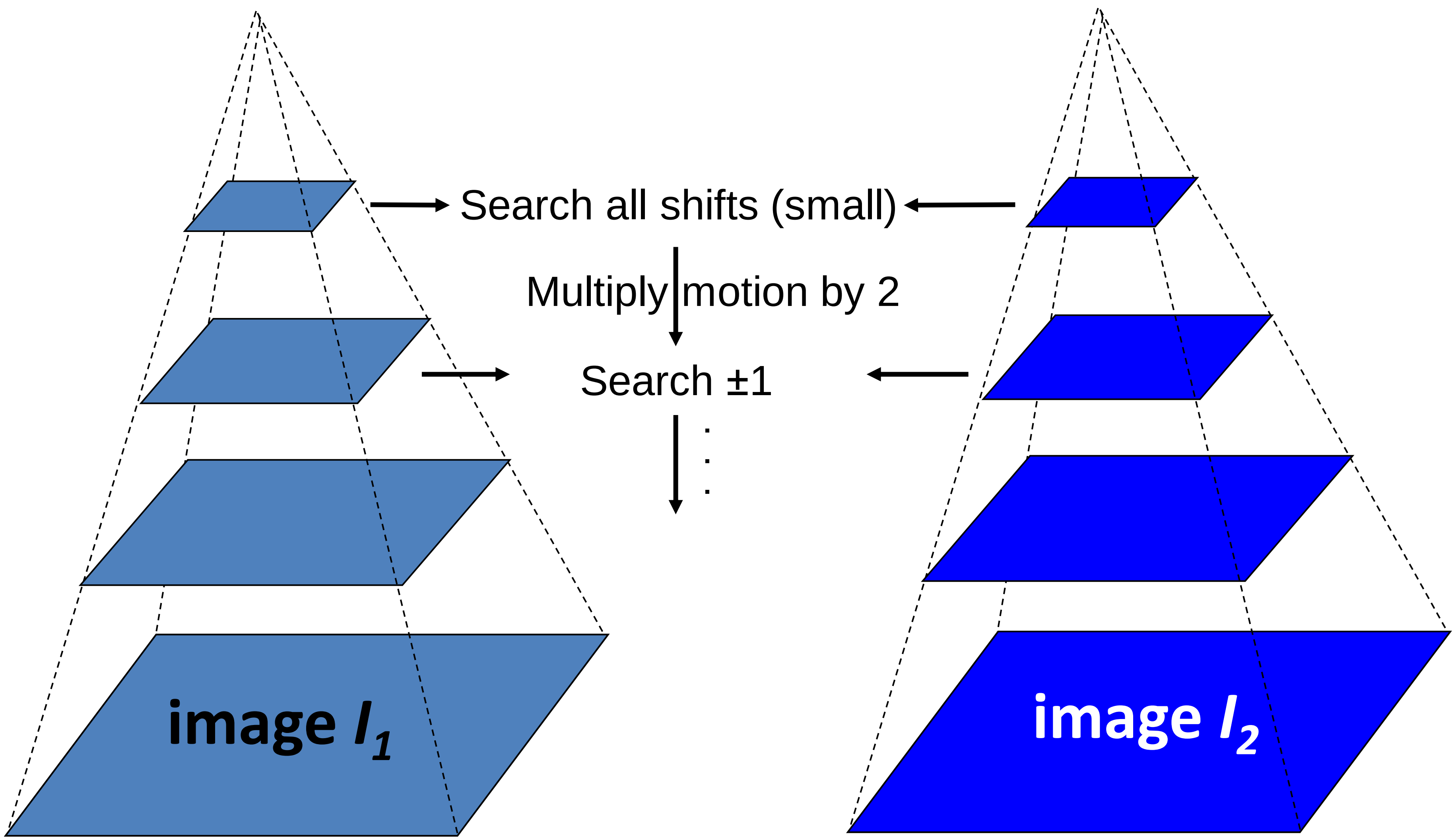
$$\frac{\text{Subtract Average Grey Level}}{\text{Divide by Variance}}$$

- Multiresolution search (Pyramids) increases search efficiency.

Coarse-to-fine motion estimation



Coarse-to-fine Image Alignment



Gaussian pyramid of image I_1 **Gaussian pyramid of image I_2**

Pattern Matching / Tracking: Normalized Cross Correlation on Windows

$$NCC(u,v) = \frac{\sum(I_1(x,y) - \hat{I}_1) \cdot (I_2(x+u,y+v) - \hat{I}_2)}{\sqrt{\sum(I_1(x,y) - \hat{I}_1)^2}\sqrt{\sum(I_2(x,y) - \hat{I}_2)^2}}$$

- Normalized Cross Correlation is an excellent method to find objects in pictures, and to track objects in video.
- Multiresolution search (Pyramids) is used in object search. Not needed when tracking from one frame to another.

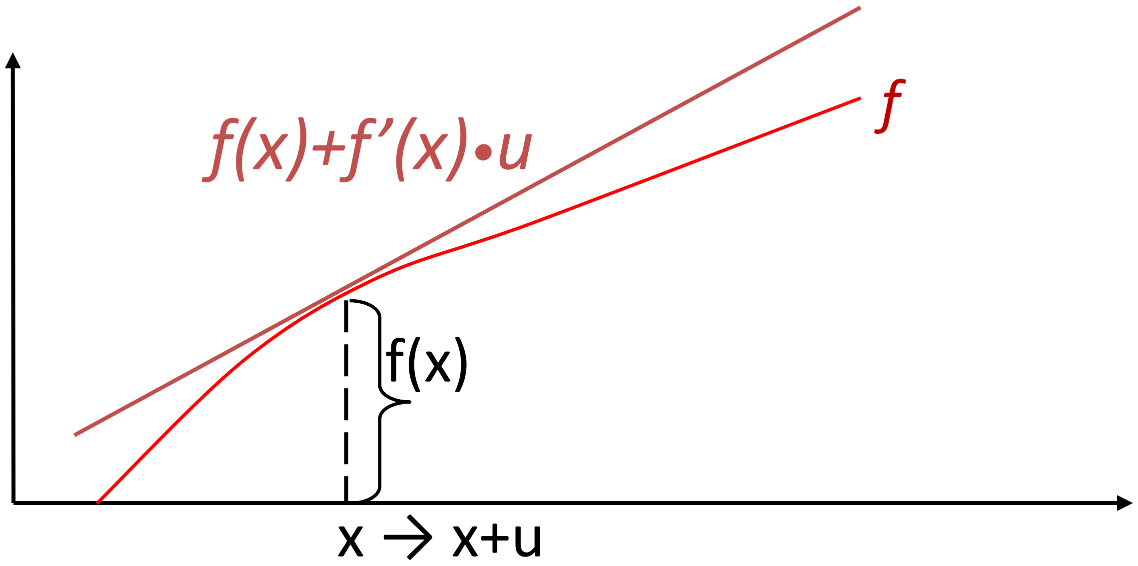
Limitations of Correlation Search

- Discrete accuracy: checking every possible translation in integer pixel values. No Sub-Pixel accuracy.
- Complexity increases exponentially with numbers of parameters
  - Translation: (u,v) Complexity is N^2
  - Rotations: (u,v,α) Complexity is N^3
  - Zoom: (u,v,α,s) Complexity is N^4
  - Affine: N^6
  - Multiresolution can help
  - Rotation does not scale..

Continuous Approximation *(Lucas – Kanade, "LK")*

- Local Taylor approximation in 1D:

$$f(x+u) \approx f(x) + f'(x) \cdot u + \ \ldots$$



- Local Taylor approximation in 2D for images:

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

Alignment by Error Minimization

- Accurate only for very small *(u,v), approximately 1 pixel*
- When I_2 is shifted relative to I_1 , we want to find the translation *(u,v)* by minimizing *SSD:* $E(u,v) = \sum_x \sum_y [I_2(x+u, y+v) - I_1(x,y)]^2$
- To simplify, we look at a single pixel (No $\sum\sum$) and use *Taylor approximation*

$$E(u,v) = [I_2(x+u, y+v) - I_1(x,y)]^2 \approx$$

$$[I_2(x,y) + \frac{\partial I_2}{\partial x} \cdot u + \frac{\partial I_2}{\partial y} \cdot v - I_1(x,y)]^2 = (I_x \cdot u + I_y \cdot v + I_t)^2$$

$$where \quad I_x = \frac{\partial I_2}{\partial x}; \quad I_y = \frac{\partial I_2}{\partial y}; \quad I_t = I_2 - I_1;$$

Error Minimization

- Writing it in simple form

$$E(u,v) = [I_2(x,y) + \frac{\partial I_2}{\partial x} \cdot u + \frac{\partial I_2}{\partial y} \cdot v - I_1(x,y)]^2 =$$

$$(I_x \cdot u + I_y \cdot v + I_t)^2$$

  - I_x : The x derivative of image I_2
  - I_y : The y derivative of image I_2
  - I_t : The image difference I_2 - I_1

- Find *(u,v)* that minimize the error function

$$E(u,v) = \sum_{x,y} (I_x(x,y) \cdot \underset{u(x,y)}{u} + I_y(x,y) \cdot \underset{v(x,y)}{v} + I_t(x,y))^2$$

Summary – Direct Methods for Global Translation

Lucas-Kanade

- When I_2 is shifted relative to I_1 , we want to find the translation (u,v) by minimizing the *SSD* (Sum of Squared Differences):

$$E(u,v) = \sum_x \sum_y [I_2(x+u, y+v) - I_1(x,y)]^2$$

- Same (u,v) will approximately minimize (Taylor approximation)

$$E(u,v) = \sum_{x,y} (I_x \cdot u + I_y \cdot v + I_t)^2$$

- Approximation accurate only for very small (u,v), ~1 pixel, as Taylor approximation is only first order

Minimization: Setting Derivatives to Zero

$$E(u,v) = \sum_{x,y} (I_x \cdot u + I_y \cdot v + I_t)^2$$

- Finding (u,v) that minimizes E by setting derivatives to zero:

$$\begin{cases} \dfrac{\partial E}{\partial u} = \sum_{x,y} I_x \cdot (I_x \cdot u + I_y \cdot v + I_t) \quad = 0 \\[2ex] \dfrac{\partial E}{\partial v} = \sum_{x,y} I_y \cdot (I_x \cdot u + I_y \cdot v + I_t) \quad = 0 \end{cases}$$

$$\begin{cases} \begin{bmatrix} \sum_{x,y} I_x \cdot I_x & \sum_{x,y} I_x \cdot I_y \\ \sum_{x,y} I_y \cdot I_x & \sum_{x,y} I_y \cdot I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_{x,y} I_x \cdot I_t \\ \sum_{x,y} I_y \cdot I_t \end{bmatrix} \end{cases}$$

Computing Motion by Solving Equations

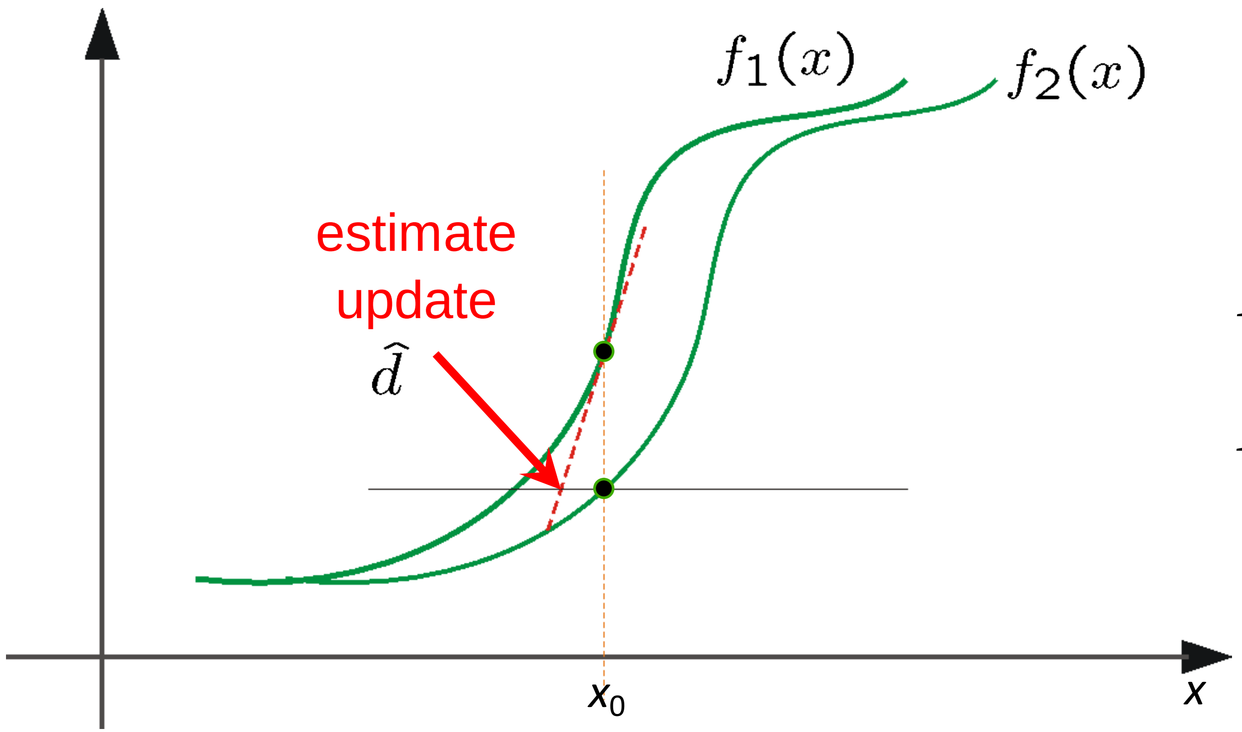
$$\begin{bmatrix} \sum_{x,y} I_x \cdot I_x & \sum_{x,y} I_x \cdot I_y \\ \sum_{x,y} I_y \cdot I_x & \sum_{x,y} I_y \cdot I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_{x,y} I_x \cdot I_t \\ \sum_{x,y} I_y \cdot I_t \end{bmatrix}$$

- These are 2 equations with two unknowns (u and v).
- System has a unique solution when the 2 eignevalues of the 2×2 matrix are high [When do we have eigenvalues of zero?]
- Same matrix is used for Harris Corner, but at a small window
- When the $\sum$ is over all pixels, both eigenvalues are almost always high

Iterative Approach (For Larger (u,v))

- Compute image derivatives I_x, I_y. Set u,v to 0.
- Compute once $$A = \begin{bmatrix} \sum I_x \cdot I_x & \sum I_x \cdot I_y \\ \sum I_y \cdot I_x & \sum I_y \cdot I_y \end{bmatrix}$$
- Iterate until convergence $(I_t \approx 0)$:
  – compute $$b = \begin{bmatrix} \sum I_x \cdot I_t \\ \sum I_y \cdot I_t \end{bmatrix}, I_t(x,y) = I_2(x,y) - I_1(x+u, y+v)$$
  – Solve equations to compute residual motion $$A \cdot \begin{bmatrix} du \\ dv \end{bmatrix} = -b$$
  – Update motion u,v with residual motion: $u{+}{=}du,\ v{+}{=}dv$
  – **Warp** I_2 towards I_1 with total motion (u,v).

Iterative Motion Estimation



Initial: $u_0=0$

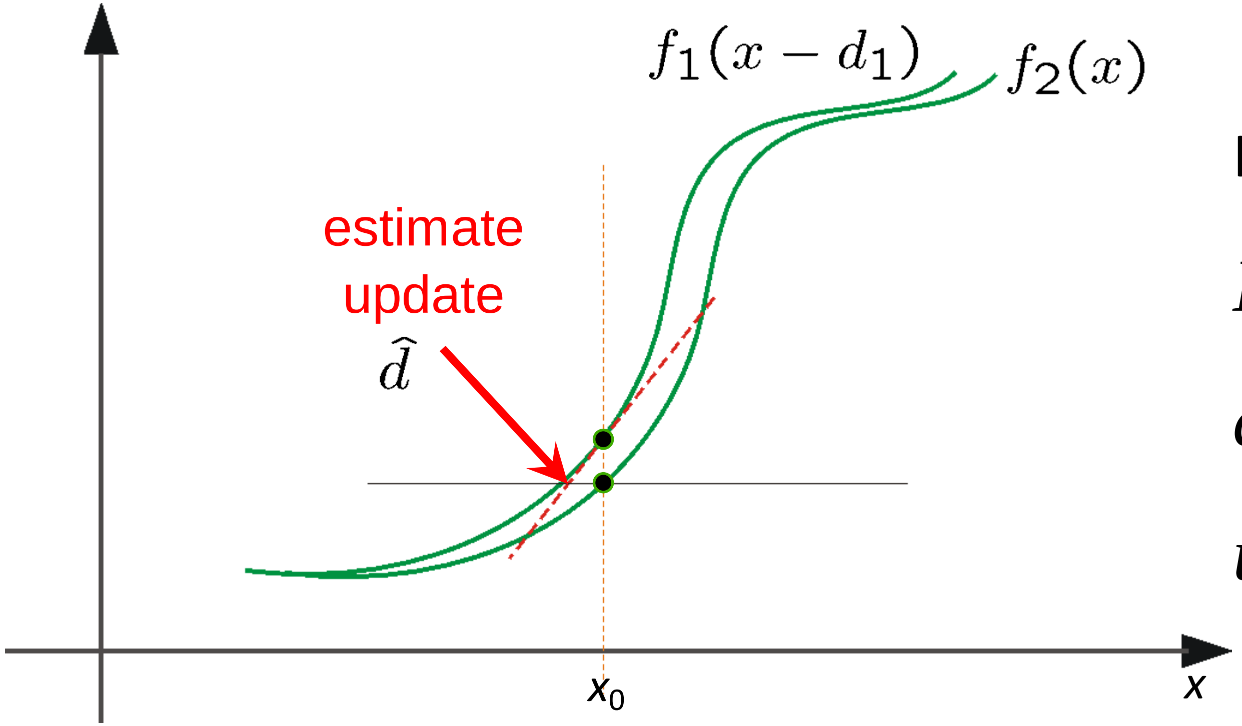
$I_t = f_1(x_0) - f_2(x_0)$

$Du = I_t \,/\, f'_1(x_0)$

$U_1 = u_0+du$

(using d for *displacement* here instead of u)

Iterative Motion Estimation



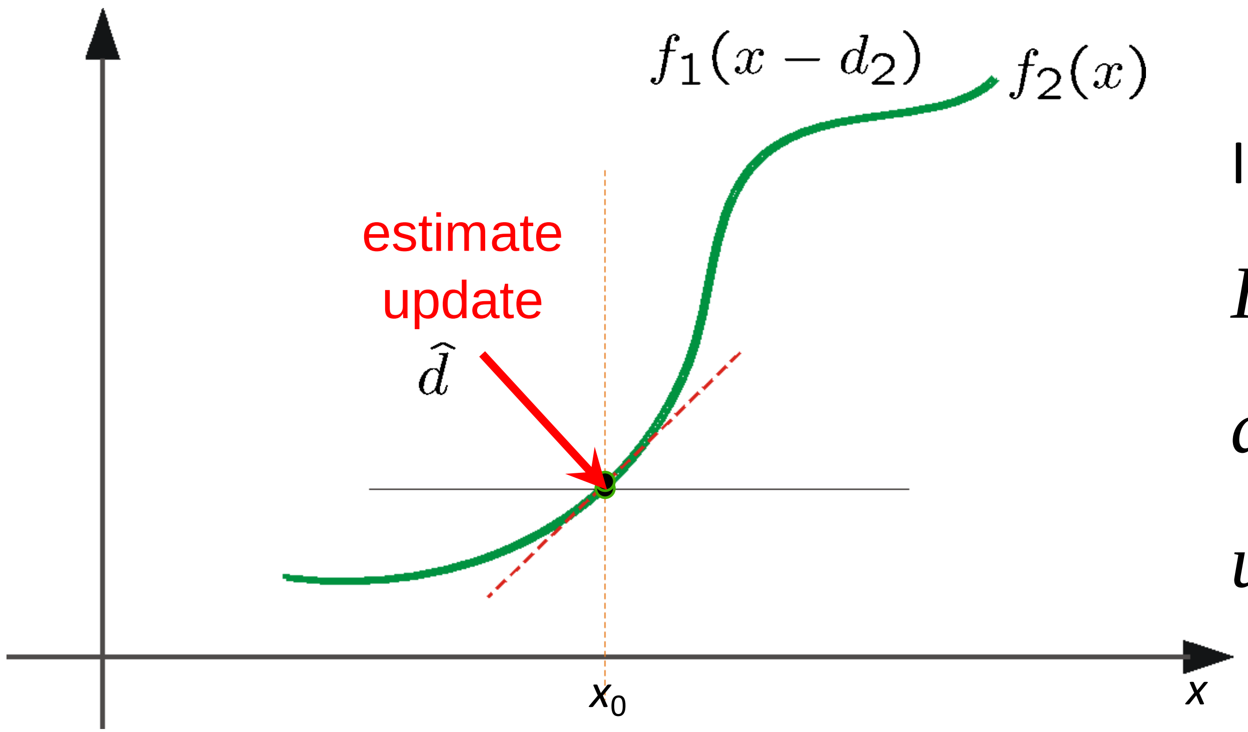
Initial: u_1

$I_t=f_1(x_0-u_1)-f_2(x_0)$

$du=I_t/f'_1(x_0-u_1)$

$u_2=u_1+du$

Iterative Motion Estimation



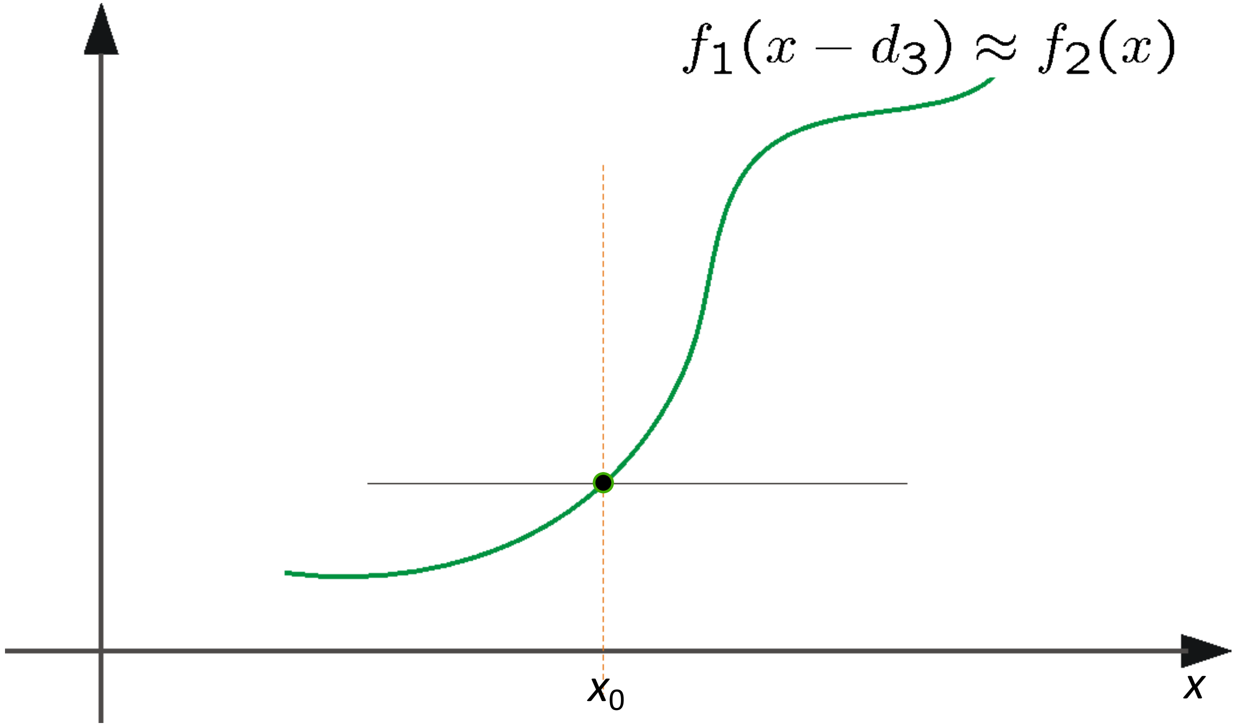
Initial: u_2

$I_t = f_1(x_0 - u_2) - f_2(x_0)$

$du = I_t / f'_1(x_0 - u_2)$

$u_3 = u_2 + du$

Iterative Motion Estimation

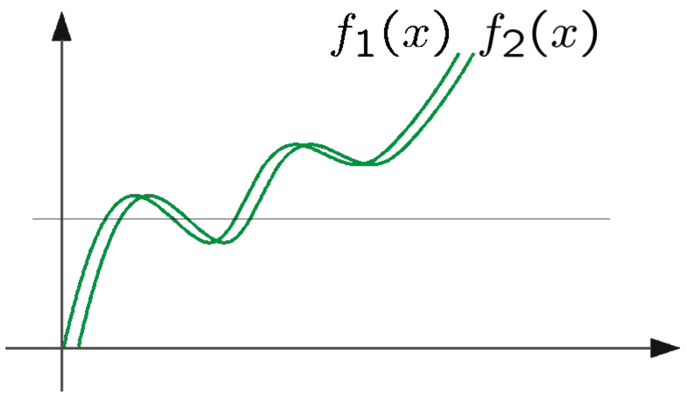


Power of Iterations

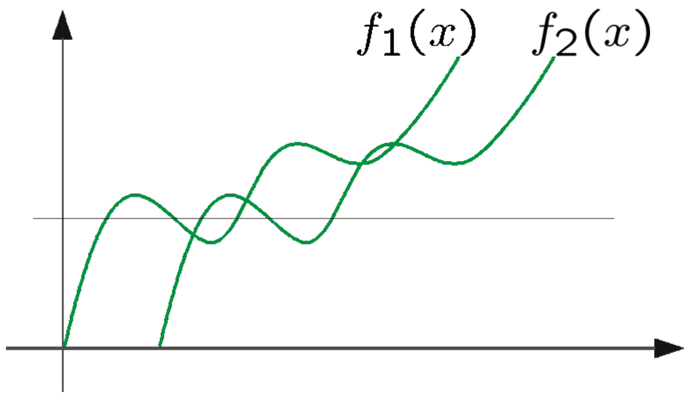
- Compute the image derivatives I_x, I_y only once on I_1
- Has two stages in each iteration:
  - Motion Estimation (Solving equations)
  - Warping I_2 (Usually backward warping)
- Works even with poor motion estimation, as long as it reduces the residual error
- Warping of one image towards the other is done from original image using total motion, and not from previous image using residual motion. (Repetitive warping blurs!)

Multiresolution

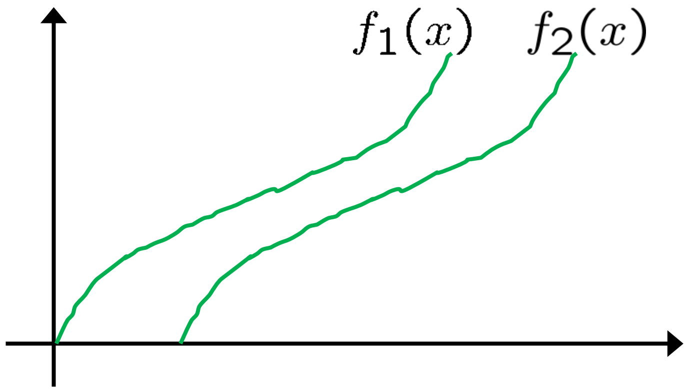
Lucas-Kanade assumes that corresponding pixels in the two images have same derivative. It works OK even if derivatives are similar. But this fails for very large motions.



Small motions: f_1 and f_2 have similar derivatives for most points



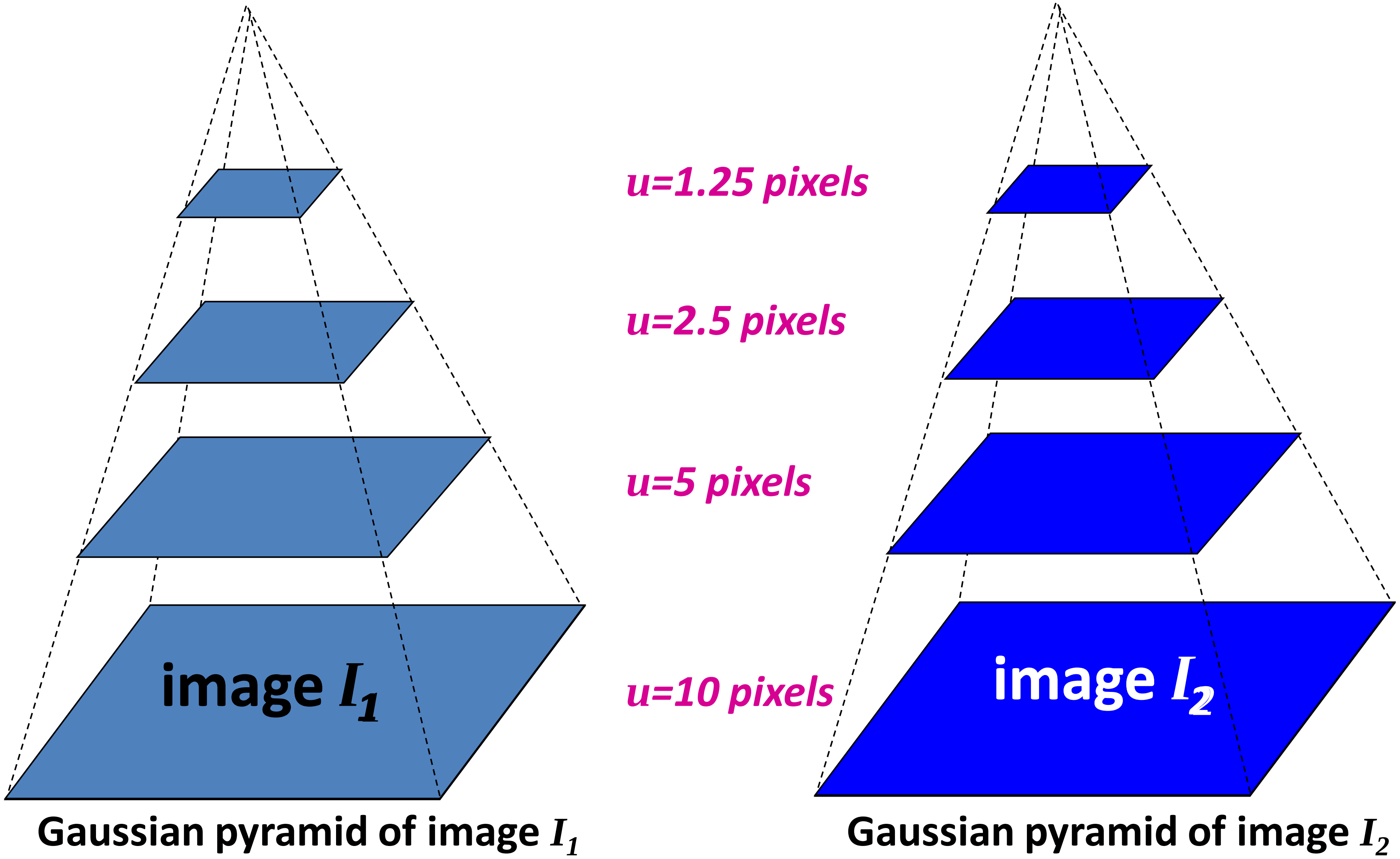
Larger motions: different derivatives for most points (opposite signs)



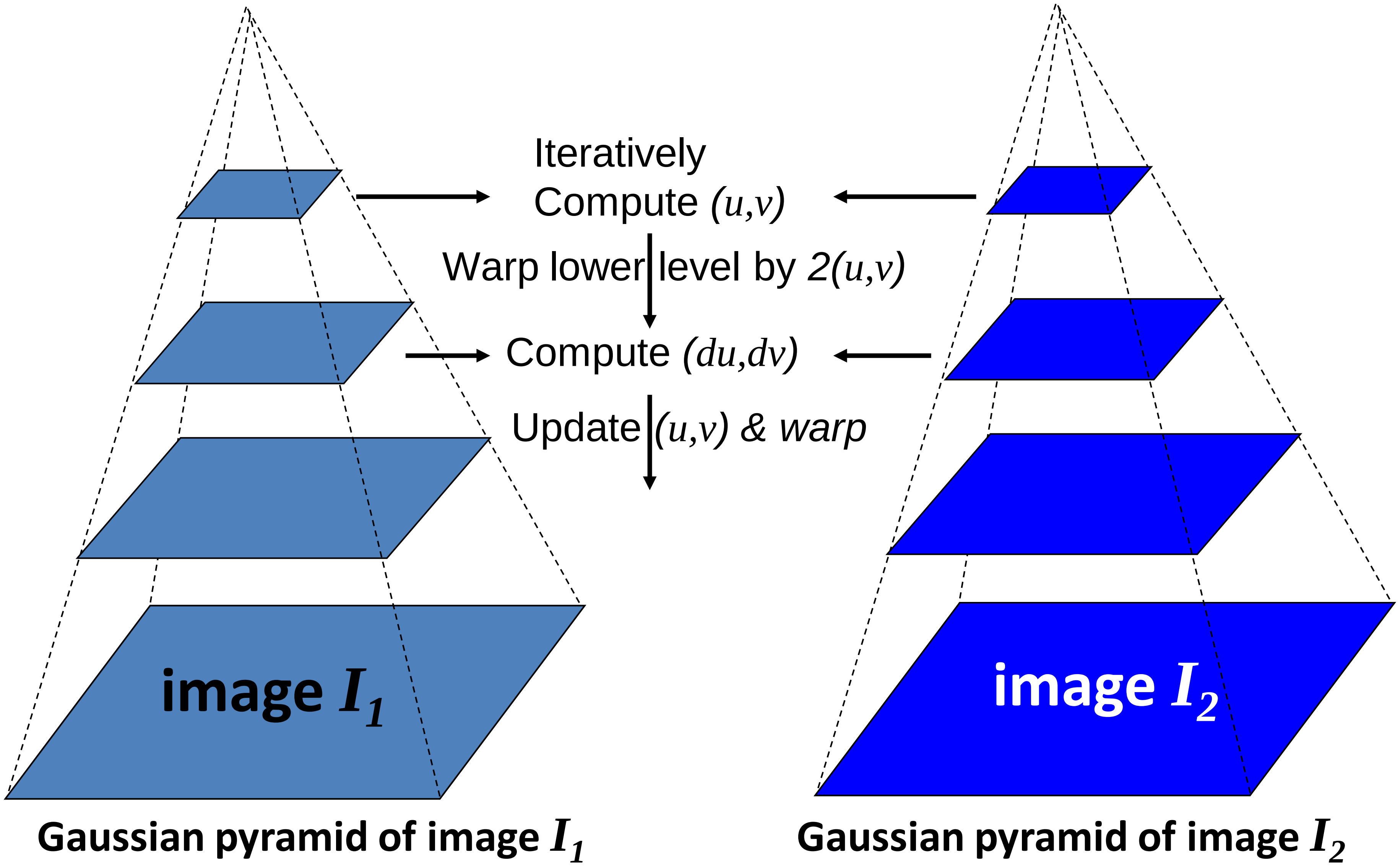
Reducing resolution blurs images. Similar derivatives even for large motions

Coarse-to-fine motion estimation

Needs very small (u,v)



Coarse-to-fine Image Alignment



Feature Points vs. Lucas Kanade (LK)

- Computation: In both cases we go over the image to compute partial derivatives. Similar complexity.
- When unique identifiable feature points can be found, they are better as they can be used to compute homographies.
- In blurry images feature points may be difficult to find. LK may be preferred.
- LK is more accurate in translation.

LK for Global Translation + Scale

$$E(u,v) = \sum_{x,y} (I_x \cdot u(x,y) + I_y \cdot v(x,y) + I_t)^2$$

- Write $u(x,y)$ and $v(x,y)$ for global translation dx, dy and **scale** s

$$\begin{aligned} x_2 &= s \cdot x_1 + dx \quad \Rightarrow \quad u(x,y) = x_2 - x_1 = (s-1)\cdot x + dx \\ y_2 &= s \cdot y_1 + dy \quad \Rightarrow \quad v(x,y) = y_2 - y_1 = (s-1)\cdot y + dy \end{aligned}$$

- Insert into the Error Equation

$$E(dx,dy,s) = \sum_{x,y} (I_x \cdot [(s-1)\cdot x + dx] + I_y \cdot [(s-1)\cdot y + dy] + I_t)^2$$

- Compute optimal dx, dy, and s by using derivatives

$$\frac{\partial E}{\partial dx} = 0; \quad \frac{\partial E}{\partial dy} = 0; \quad \frac{\partial E}{\partial s} = 0$$

LK for Global Translation + Scale

$$E(dx, dy, s) = \sum_{x,y} (I_x \cdot [(s-1)\cdot x + dx] + I_y \cdot [(s-1)\cdot y + dy] + I_t)^2$$

- Compute dx, dy, and s by using derivatives

$$\frac{\partial E}{\partial dx} = 0; \quad \frac{\partial E}{\partial dy} = 0; \quad \frac{\partial E}{\partial s} = 0$$

$$0 = \sum_{x,y} (I_x \cdot [(s-1)\cdot x + dx] + I_y \cdot [(s-1)\cdot y + dy] + I_t) \cdot I_x$$

$$0 = \sum_{x,y} (I_x \cdot [(s-1)\cdot x + dx] + I_y \cdot [(s-1)\cdot y + dy] + I_t) \cdot I_y$$

$$0 = \sum_{x,y} (I_x \cdot [(s-1)\cdot x + dx] + I_y \cdot [(s-1)\cdot y + dy] + I_t) \cdot (xI_x + yI_y)$$

- Solve 3 linear equations with 3 unknowns

LK for Global Translation + Rotation (Small α)

- Needs approximation of <u>small α</u> to remain linear

$$x_2 = \cos(\alpha) \cdot x_1 - \sin(\alpha) \cdot y_1 + dx \approx x_1 - \alpha \cdot y_1 + dx$$
$$y_2 = \sin(\alpha) \cdot x_1 + \cos(\alpha) \cdot y_1 + dy \approx \alpha \cdot x_1 + y_1 + dy$$

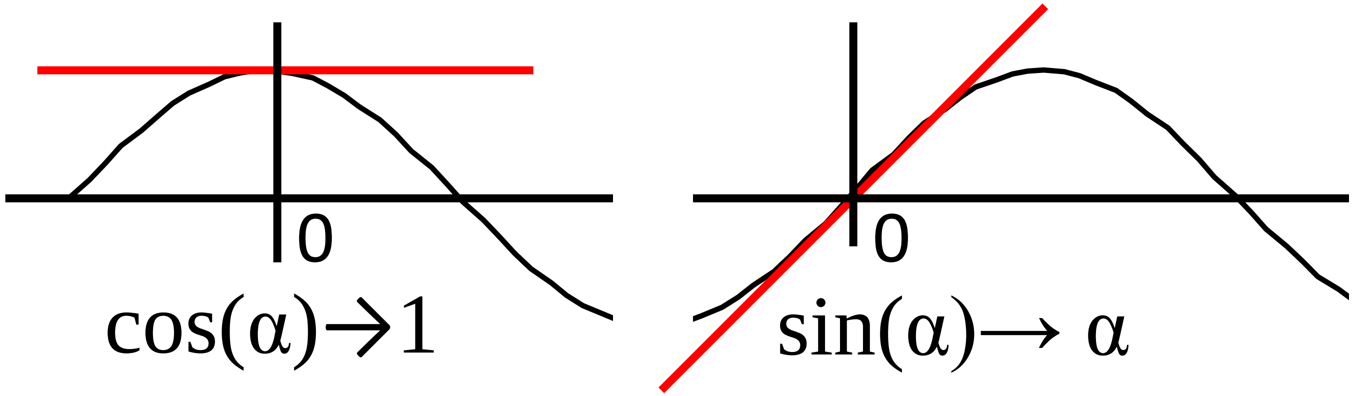
$\sin(\alpha) \rightarrow \alpha$ (Assuming small α)

$\cos(\alpha) \rightarrow 1$ (Assuming small α)

$$u(x,y) = x_2 - x_1 = -\alpha \cdot y_1 + dx$$
$$v(x,y) = y_2 - y_1 = \alpha \cdot x_1 + dy$$

$$E(dx, dy, \alpha) = \sum_{x,y} (I_x \cdot [-\alpha \cdot y + dx] + I_y \cdot [\alpha \cdot x + dy] + I_t)^2$$

Small α Assumption



- The "small α assumption" is used only for motion estimation (solving the equations for angle difference)
- Warping is done with full accuracy of *sin* and *cos*
- Iterations converge to an accurate solution, with α=0

LK for Global Translation + Rotation *(unverified)*

$$E(dx,dy,\alpha)=\sum_{x,y}(I_x\cdot[-\alpha\cdot y+dx]+I_y\cdot[\alpha\cdot x+dy]+I_t)^2$$

$$\frac{\partial E}{\partial dx}=0=\sum_{x,y}(I_x\cdot[-\alpha\cdot y+dx]+I_y\cdot[\alpha\cdot x+dy]+I_t)\cdot I_x$$

$$\frac{\partial E}{\partial dy}=0=\sum_{x,y}(I_x\cdot[-\alpha\cdot y+dx]+I_y\cdot[\alpha\cdot x+dy]+I_t)\cdot I_y$$

$$\frac{\partial E}{\partial \alpha}=0=\sum_{x,y}(I_x\cdot[-\alpha\cdot y+dx]+I_y\cdot[\alpha\cdot x+dy]+I_t)\cdot\left(I_y x-I_x y\right)$$

- Iterations: Solve with "small α assumption"
- <u>Warp with full accuracy of *sin* and *cos*.</u>
- Pyramids: *u*, *v* get smaller, but Angle α remains the same…

Translation + Rotation

(unverified Matrix Representation)

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y & \sum \left(I_y I_x x - I_x I_x y\right) \\ \sum I_x I_y & \sum I_y I_y & \sum \left(I_y I_y x - I_x I_y y\right) \\ \sum I_x \left(I_y x - I_x y\right) & \sum I_y \left(I_y x - I_x y\right) & \sum \left(I_y x - I_x y\right)^2 \end{bmatrix} \begin{bmatrix} dx \\ dy \\ \alpha \end{bmatrix} = \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \\ \sum \left(I_y x - I_x y\right) I_t \end{bmatrix}$$

Representation of Transformation by Homogenous Coordinates

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & d_x \\ \sin(\alpha) & \cos(\alpha) & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

- Transformations can be chained by matrix multiplication. Important for iterations.

Perspective Transformation (Homography) From Corresponding Points

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$x_2 = \frac{ax_1 + by_1 + c}{gx_1 + hy_1 + 1}$$

$$y_2 = \frac{dx_1 + ey_1 + f}{gx_1 + hy_1 + 1}$$

Alignment Error corresponding points

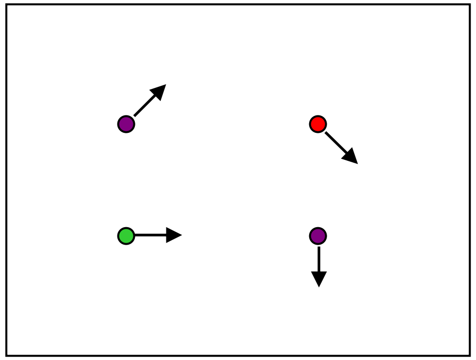
$$E^2 = (\hat{x}_2 - x_2)^2 + (\hat{y}_2 - y_2)^2$$

$$E^2 = \left(\hat{x}_2 - \frac{ax_1 + by_1 + c}{gx_1 + hy_1 + 1}\right)^2 + \left(\hat{y}_2 - \frac{dx_1 + ey_1 + f}{gx_1 + hy_1 + 1}\right)^2$$

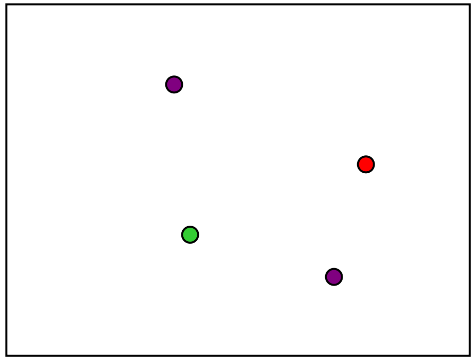
Optical Flow: Different Motion for Each Pixel

- Optical Flow: Individual motion for each pixel
  - Independently moving objects
  - Motion parallax (Different depths)
- What will global LK translation give on the above?

Optical Flow



$H(x, y)$



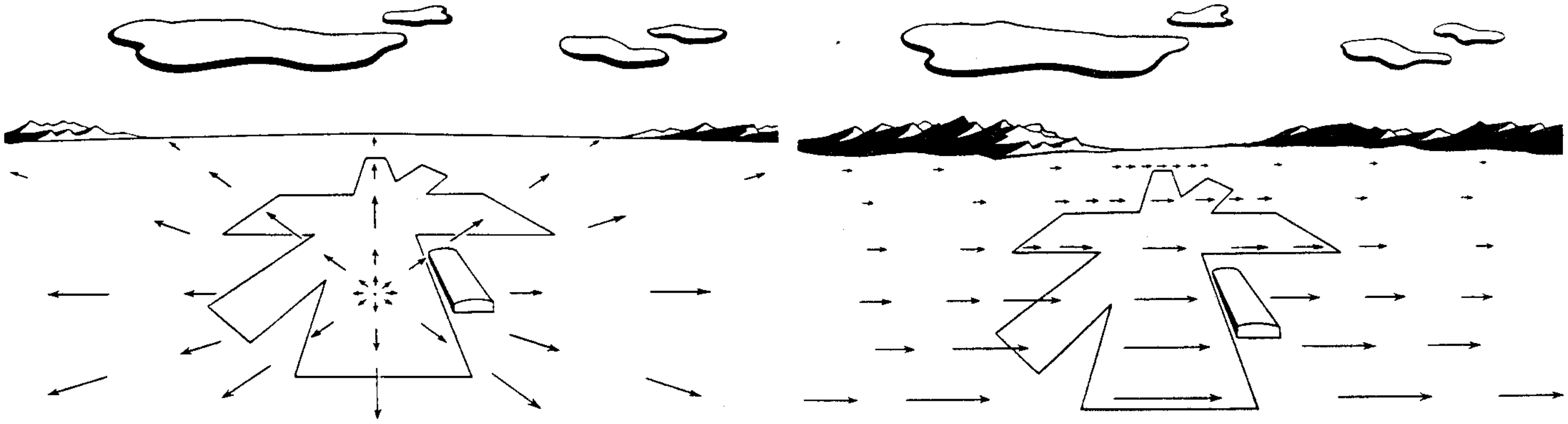
$I(x, y)$

- Estimate pixel motion from image H to image I

Key assumptions

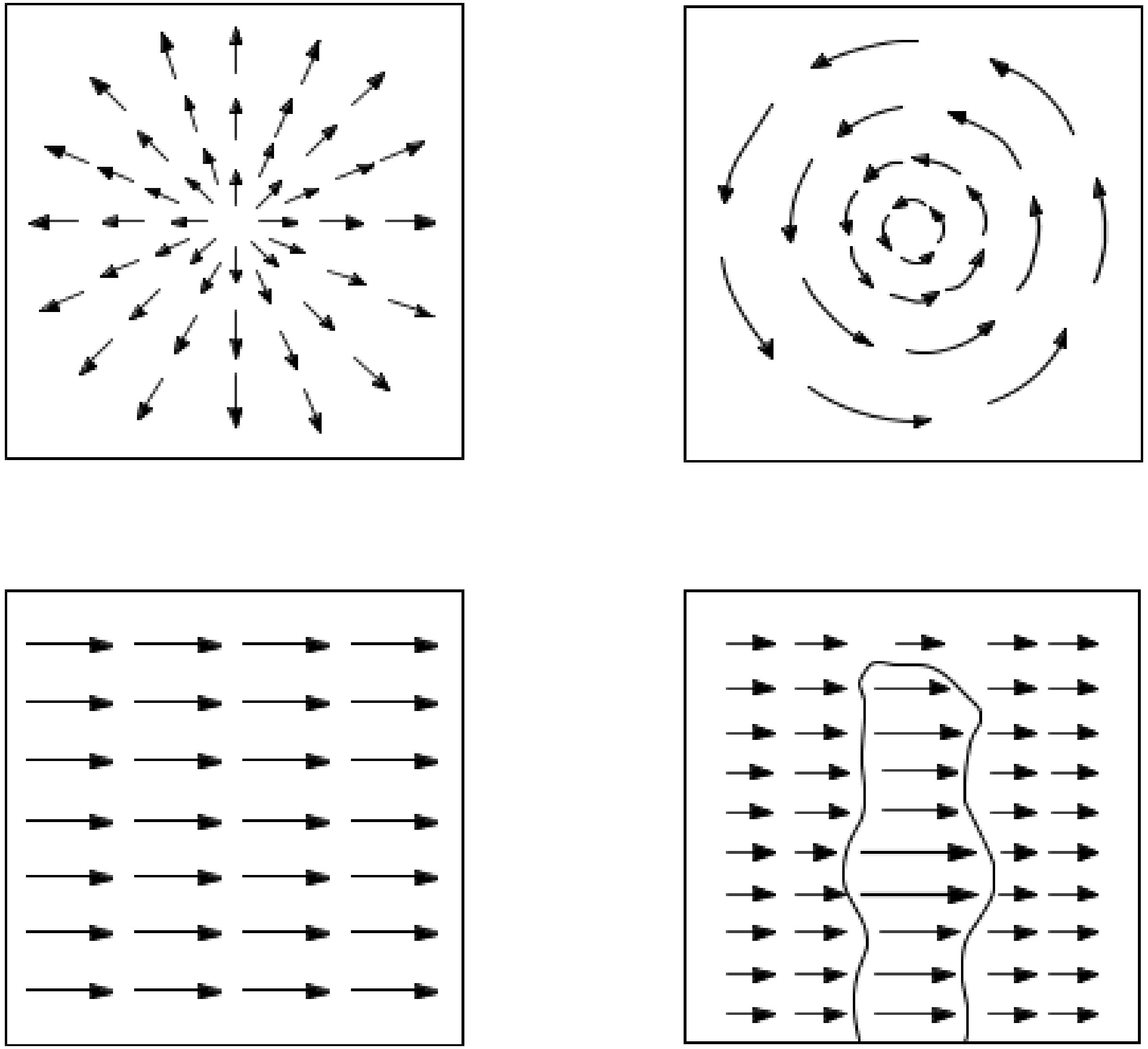
- Color Constancy: No change on color
  - Grayscale images: *Brightness Constancy*
- Small Motion: points do not move very far

Examples of Optical Flow

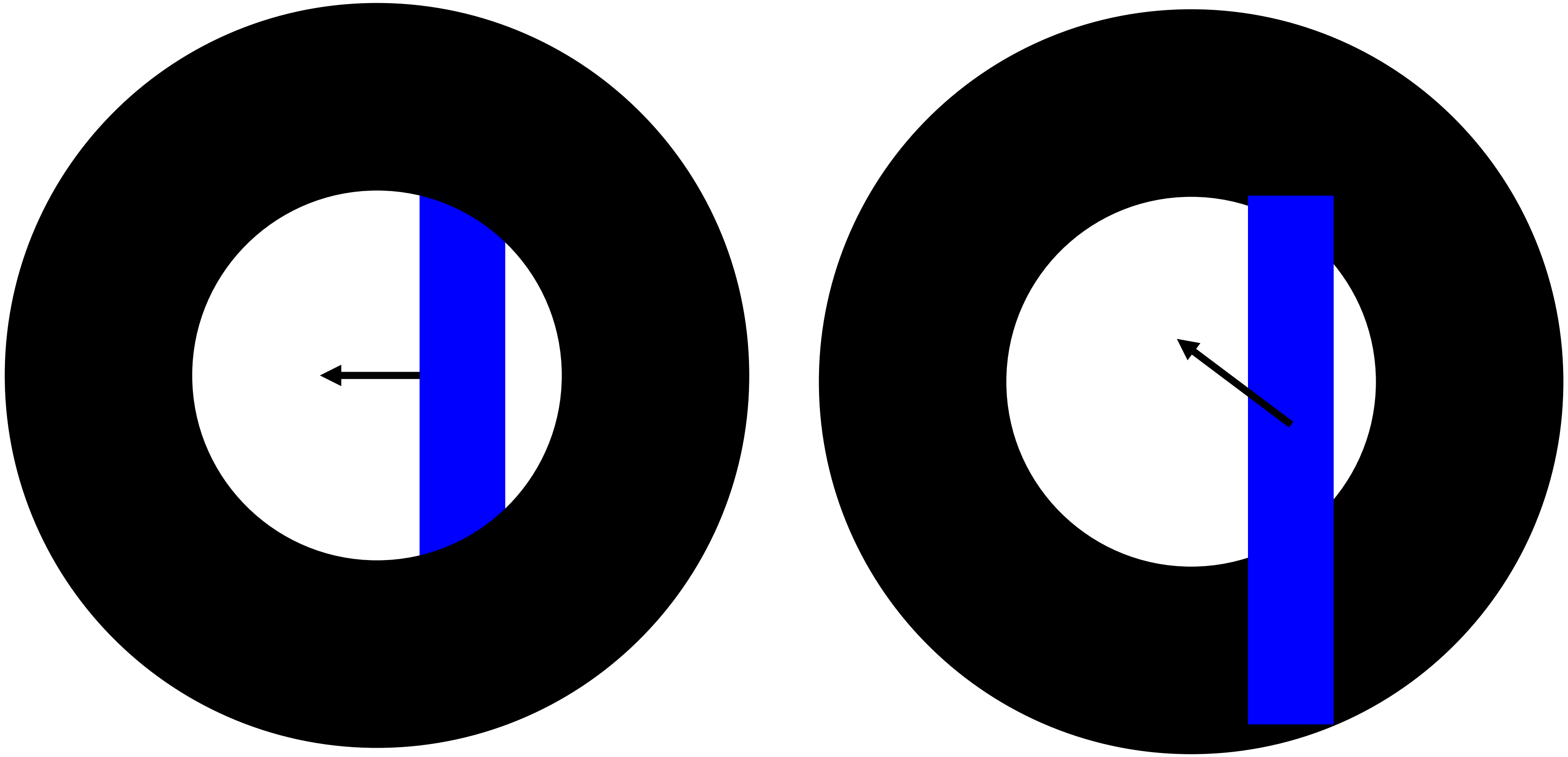


What motions generated these optical flow vectors?

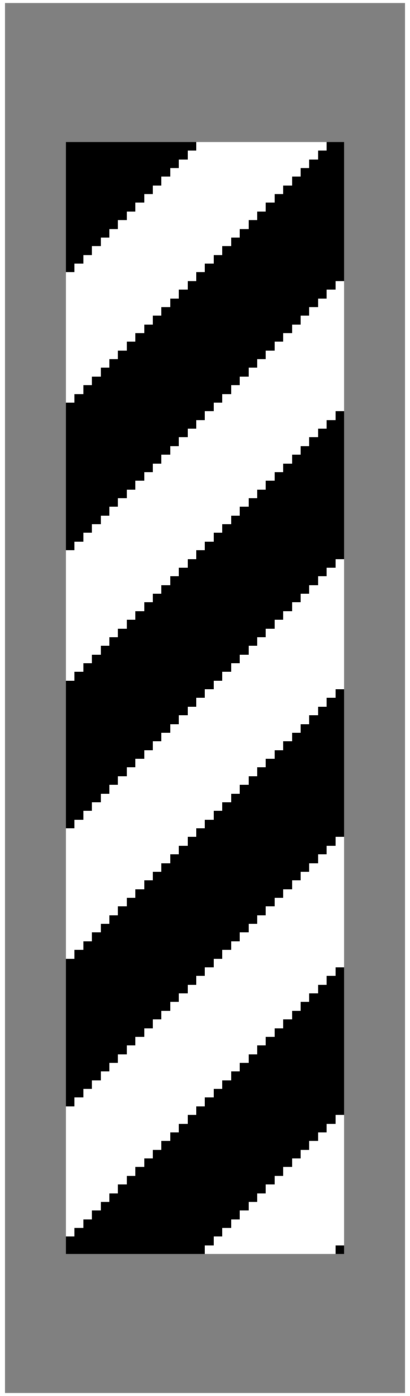
Examples of Optical Flow



The Aperture Problem



- Examining a small windows around a pixel may not provide accurate motion
  - A straight edge with smooth areas



Barberpole Illusion

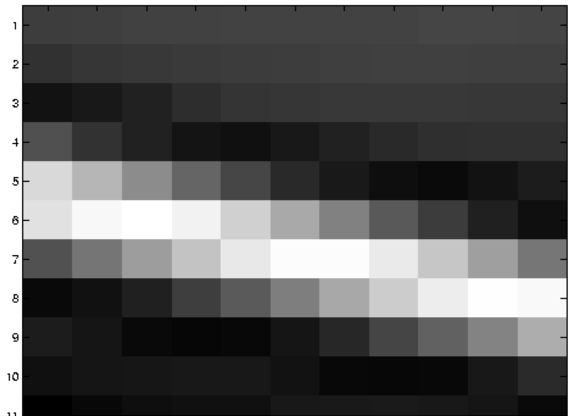
Correlation Based Optical Flow

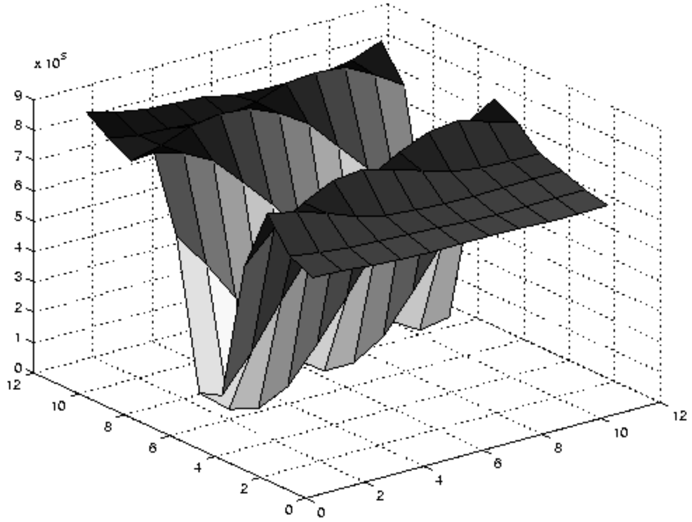
- For each small region in one image, search for best correlation at the second image

- Large region: Accurate motion. Poor localization.
- Small region: Good localization. Poor motion.
- Use pyramids to reduce search area.

Special Case - Edge

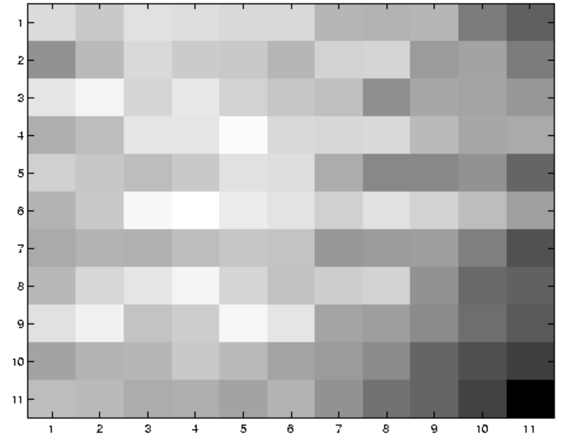
Correlation in Local Search Area

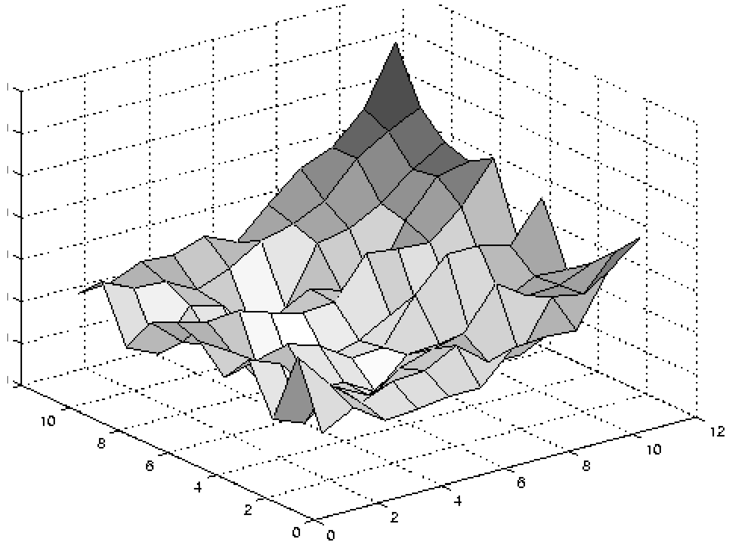




Special Case – Smooth Region

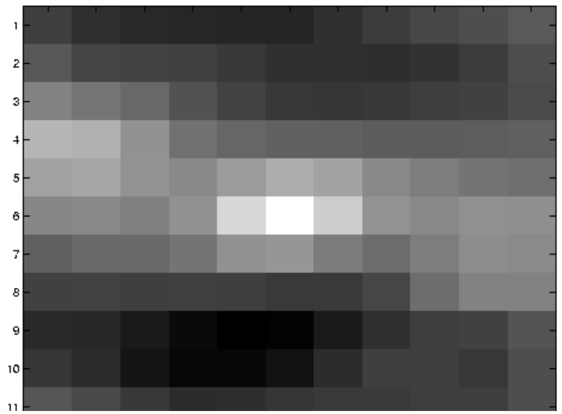
Correlation in Local Search Area

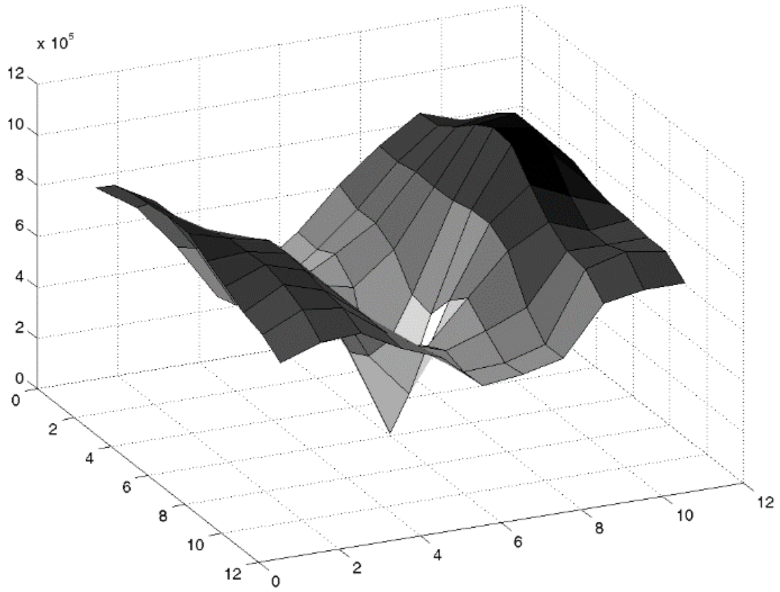




Special Case - Texture

Correlation in Local Search Area



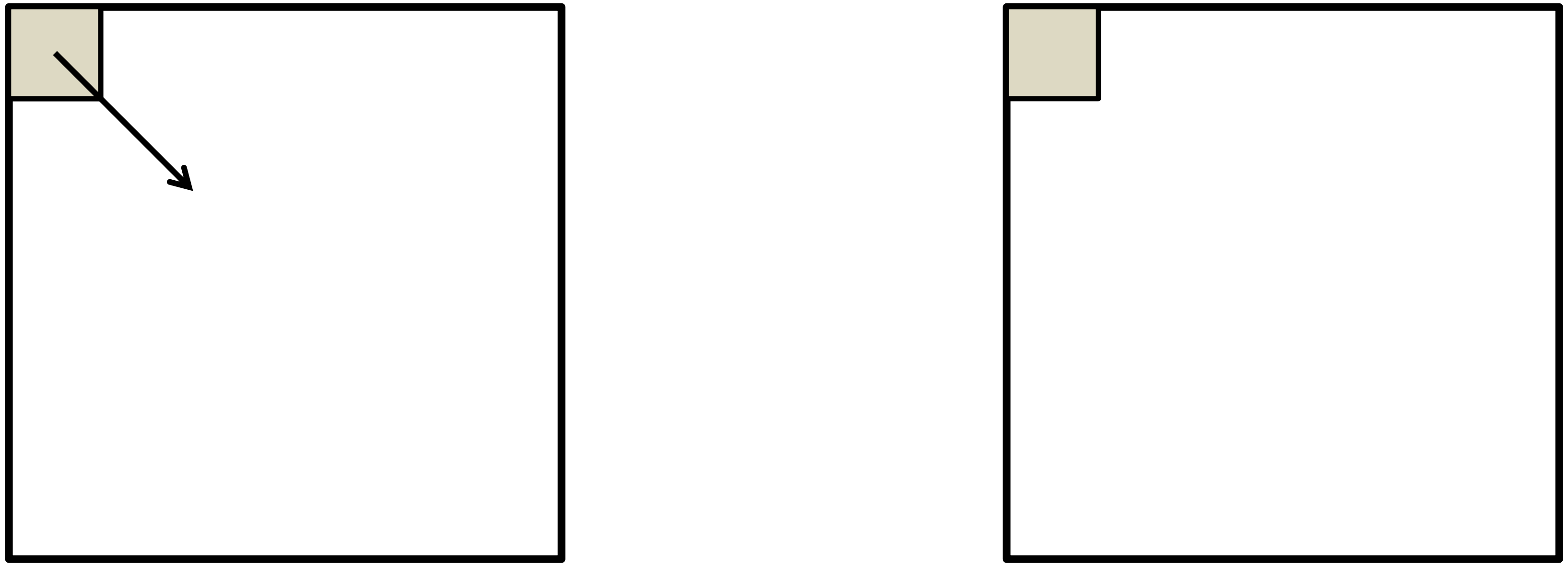


Correlation Based Pyramids for Optical Flow

- Create two Gaussian pyramids from the two input images
- Compute optical flow using “5×5” regions on smallest pyramid level
- Smooth the optical flow, and use it as initial guess for higher resolution
- Continue with next level. Search close to guess from higher resolution

Gradient Based Optical Flow

- Compute (u,v) using Lucas-Kanade between two corresponding regions



- Large region: Accurate motion. Poor localization.
- Small region: Good localization. Poor motion.
- Use pyramids to reduce search area.

Smoothness Constraint

- Assume that the optical flow is piecewise constant.
- Assume that the optical flow is smooth, and minimize the sum of its squared first derivatives.
- Given $\nabla I=(I_x, I_y)$, find $\mathbf{v}=(u,v)$ that Minimize:

$$E^2 = \int\int \underbrace{(\nabla I \bullet \mathbf{v} + I_t)}_{\text{Optical Flow Constraint Equation}} + \underbrace{\alpha^2 \left(\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \right)}_{\text{Smoothness Constraint}} \mathrm{dxdy}$$

Pyramids & Iterative Refinement

- Create two Gaussian pyramids from the two input images
- Iterative Lukas-Kanade on smallest images
  1. Estimate velocity at each pixel by solving Lucas-Kanade equations in its neighborhood
  2. Warp I_2 towards I_1 using the estimated flow field
  3. Repeat until convergence
- Continue to next pyramid level.

Smoothness Constraint

- The smoothness constraint is violated on the boundaries of moving objects, and on motion discontinuities.
- Replace square error by more robust error measures, absolute value, etc.