חזרות

1. Email: Give Examples Before Theory

2. MRI and Fourier

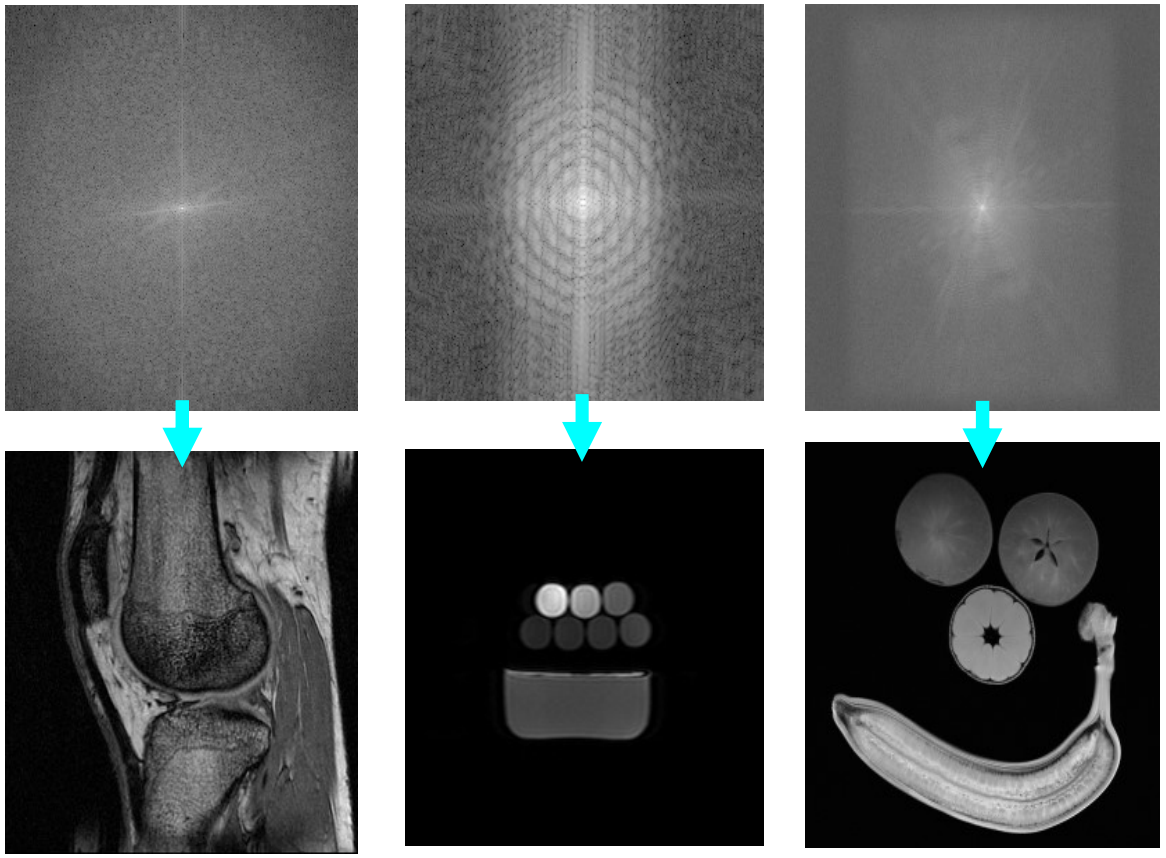
3. Need a Student to Implement Deblurring
   a) Maybe an exercise next year
   b) Get 2 or 4 credit points
   c) Come to me at the break

MRI IS ACQUIRED IN FOURIER DOMAIN!

Sampling domain
(k-space)

$\mathcal{F}^{-1}$

Image domain



Measurement model:

$$\mathbf{y} = \mathcal{F}\{\mathbf{x}\}$$

Acquired data

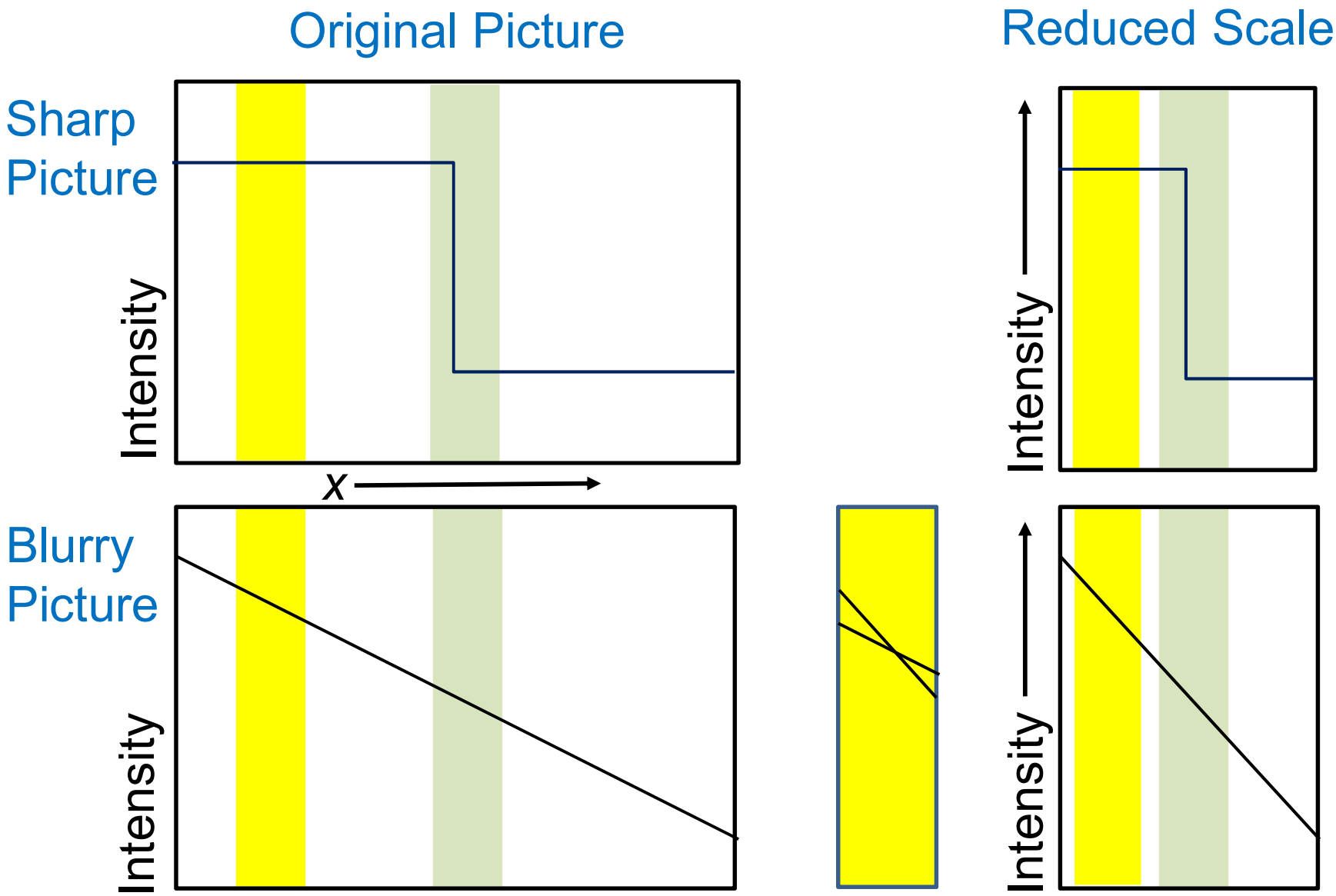
Fourier transform

MR image (as column vector)

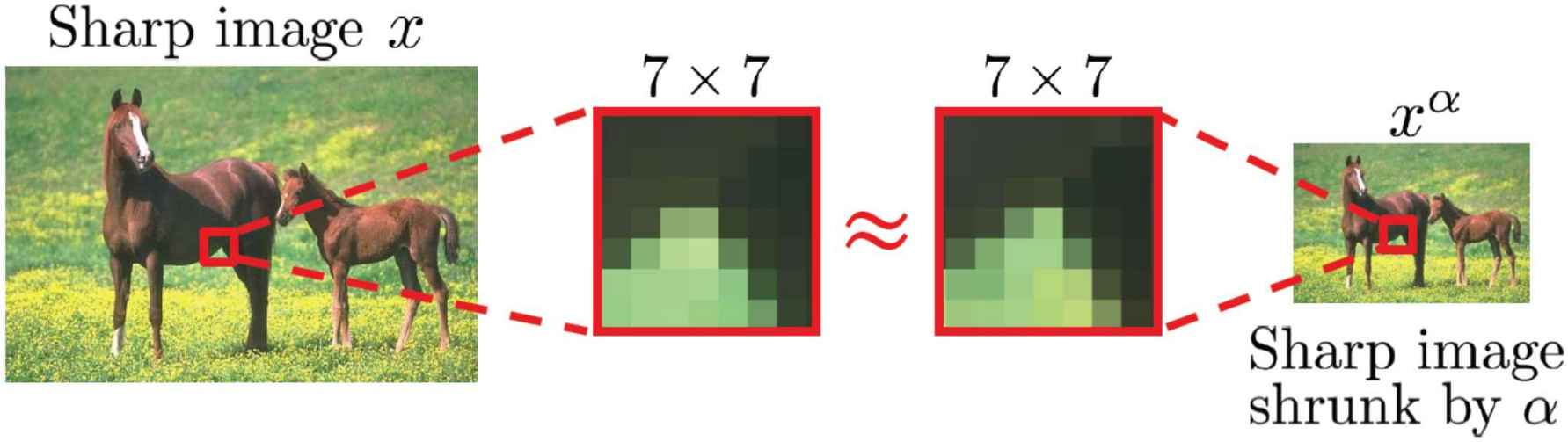
Blind Image Deblurring (Michaeli & Irani)

- Given a blurred image *y* we look for a sharp image *x* and a blur kernel *k* such that *y=x*k*
- This is an ill-conditioned problem. Priors needed to help solve it.
- E.g. Allow only a limited set of sharp patches
- Observations used for Priors in this approach:
  - High patch recurrence across scales in sharp images
  - Lower patch recurrence across scales in blurred images

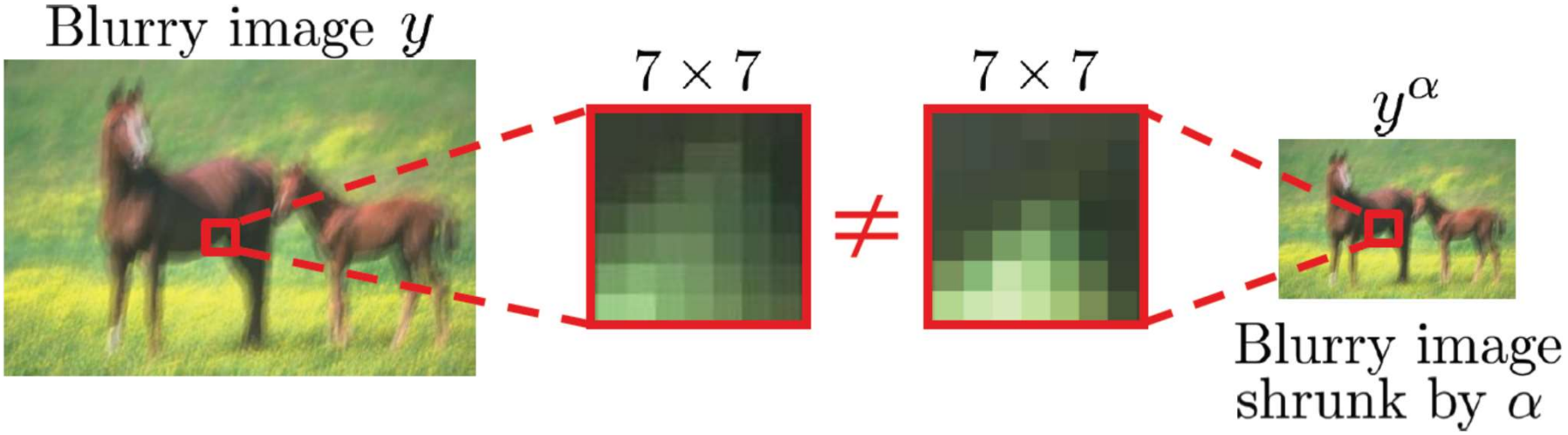
Cross Scale Similarity: One Row

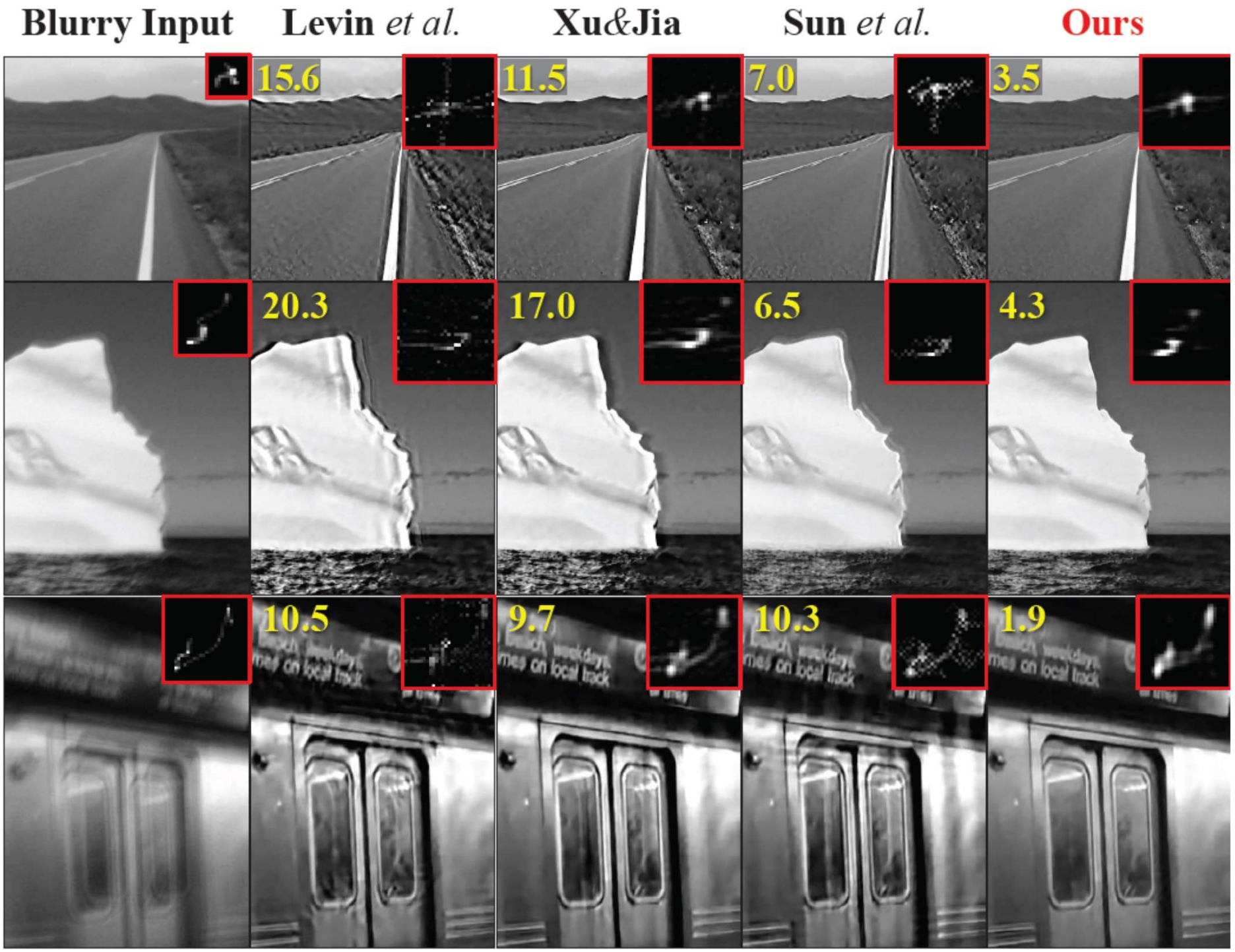


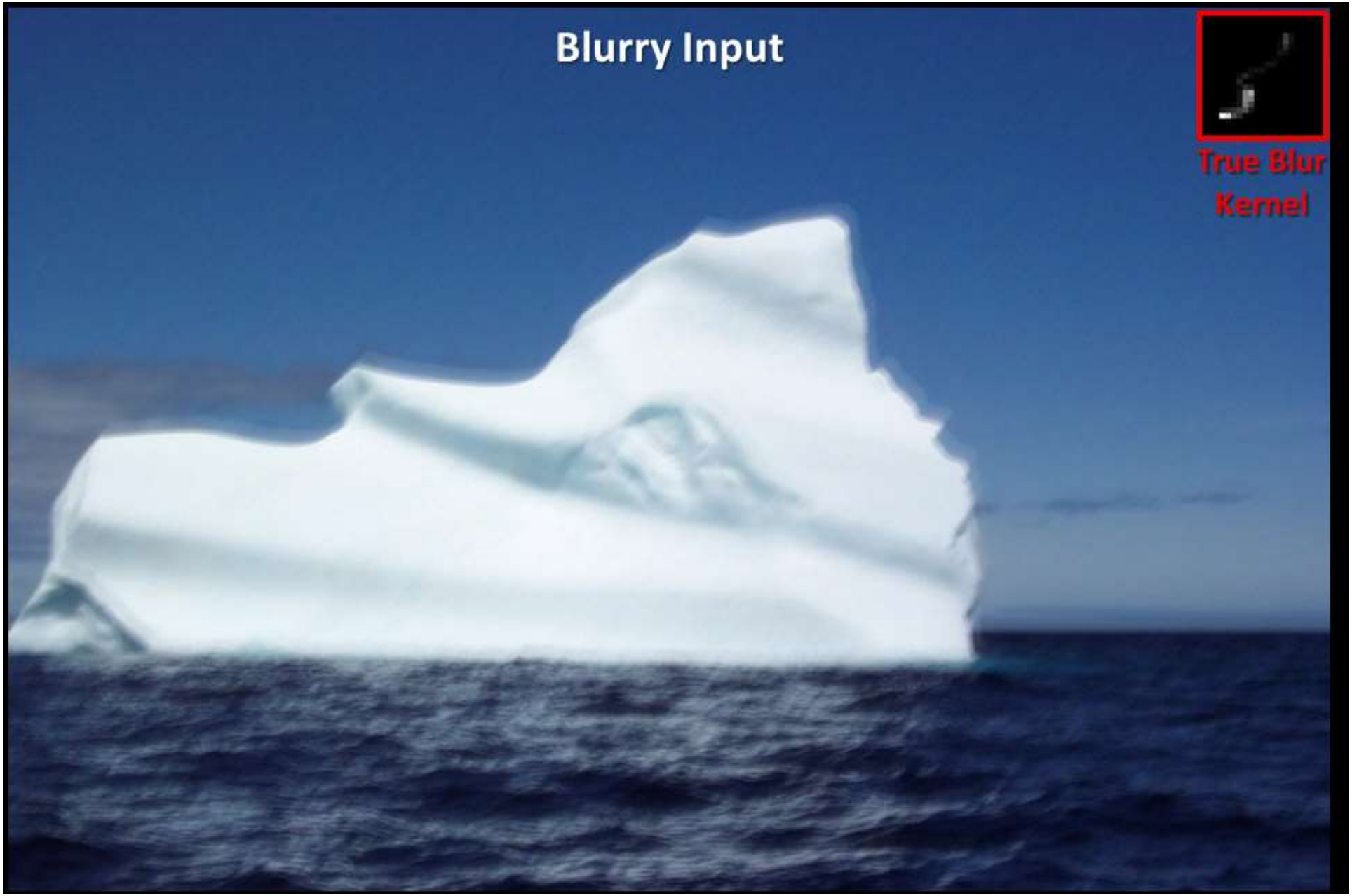
Patch Recurrence Across Scales in Sharp Images

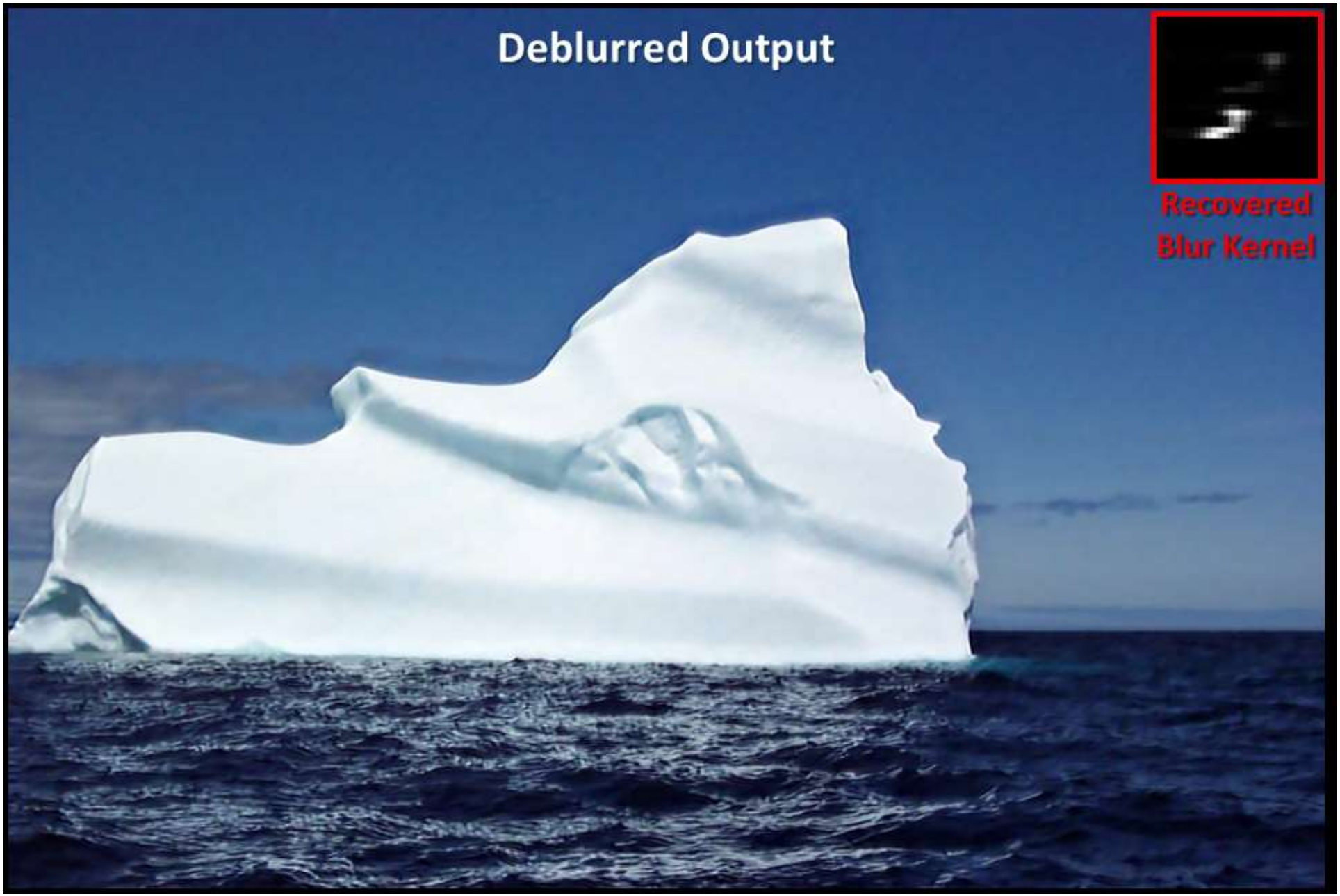


Patch Recurrence Across Scales in Blurry Images




Blurry Input
Levin et al.
Xu&Jia
Sun et al.
Ours
15.6
11.5
7.0
3.5
20.3
17.0
6.5
4.3
10.5
9.7
10.3
1.9


Blurry Input
True Blur Kernel


Deblurred Output
Recovered
Blur Kernel

Deblurring by Optimization

$$\arg\min_{\hat{x},\hat{k}} \underbrace{\|y - \hat{k} * \hat{x}\|^2}_{\text{data term}} + \lambda_1 \underbrace{\rho(\hat{x}, \hat{x}^{\alpha})}_{\text{image prior}} + \lambda_2 \underbrace{\|\hat{k}\|^2}_{\text{kernel prior}}$$

- Looking for sharp image *x* and blur kernel *k* s.t.
  - Data Term: Blurring *x* by *k* gives *y*
  - Image Prior: Low L^2 distance between patches in *x* and their Nearest Neighbor in scaled-down *x*.
  - Kernel Prior: Sharpest Possible Kernel

Kernel Estimation

- Input: Blurry image y
- Output: Blur Kernel k
- Initialize $k=\delta$, $x=y$
- For $t=1,\ldots, T$
  1. Image Prior Update: Scale down x to get x^a
     - x^a Has a pool of sharper patches
  2. Deblurring: Minimize for x (k and x^a fixed)
     - What happens in Iteration 1?
  3. Kernel Update: Minimize for k (x and x^a fixed)

Details of the steps

1. Scale down in Fourier domain (Sinc kernel)
2. Deblur using sharp patches and *k*

$$\arg\min_{\hat{x}} \|y - \hat{\boldsymbol{K}}\hat{x}\|^2 + \lambda_1 \rho(\hat{x}, \hat{x}^{\alpha})$$

   – Replace each patch by its NN (Nearest Neighbor)

   – Enforce data term. Repeat.

3. Compute *k*

$$\arg\min_{\hat{k}\geq 0} \|y - \hat{\boldsymbol{X}}\hat{k}\|^2 + \lambda_2 \|\hat{k}\|^2$$

Implementation Details

- Build a pyramid, where each level is downscaled by a factor of 4/3, using Sinc
- Image level kernel is 51×51. Build pyramid levels until kernel size is 5×5.
- Processing starts at smallest pyramid level. Each recovered *x* and *k* are interpolated to higher levels as initial guesses.
- Solving deblurring equations using Fourier


Blurry Input Image


Deblurred
34 St
34 St

Image Alignment

- Find the transformation between two images
  - Translation, Rotation, zoom
  - Affine, Homography
  - Assumption: No effects of 3D parallax
- Good for:
  - Video Stabilization, Video Mosaicing…

Parametric (global) warping

- Examples of parametric warps:

Translation (2 Parameters)

Scaling (1)
Keeps Angles

Rotation (1)
Keeps Distances

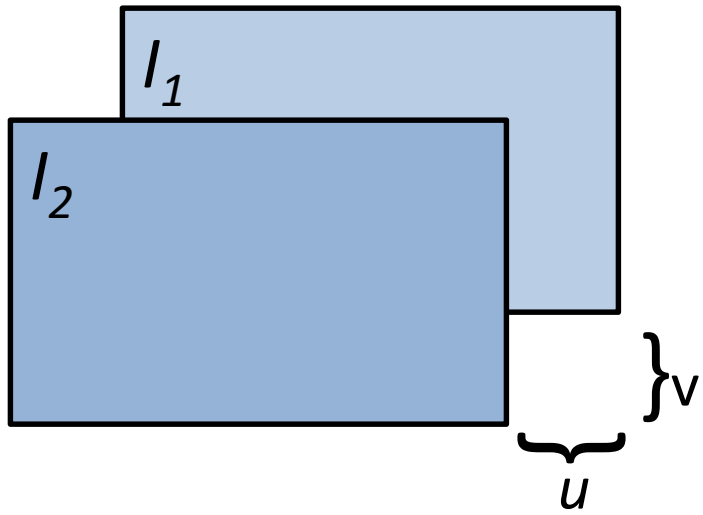
Affine (6)
Keeps Parallel

Projective (8)
Keeps straight lines

Computing Translation, Direct Methods

Assumption: Constant Brightness

- Given images I_1 and I_2, we can find the translation *(u,v)* that will minimize the squared error $$E(u,v) = \sum_x \sum_y \left(I_1(x,y) - I_2(x+u, y+v)\right)^2$$
- Average over area of overlap
- Can also search for rotations: *(u,v,α)*



Cross Correlation

- Starting from the *SSD*

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

- Since $(a-b)^2 = a^2 - 2ab + b^2$
- 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 $\Sigma I_1 I_2$

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

Normalized Cross Correlation (NCC)

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

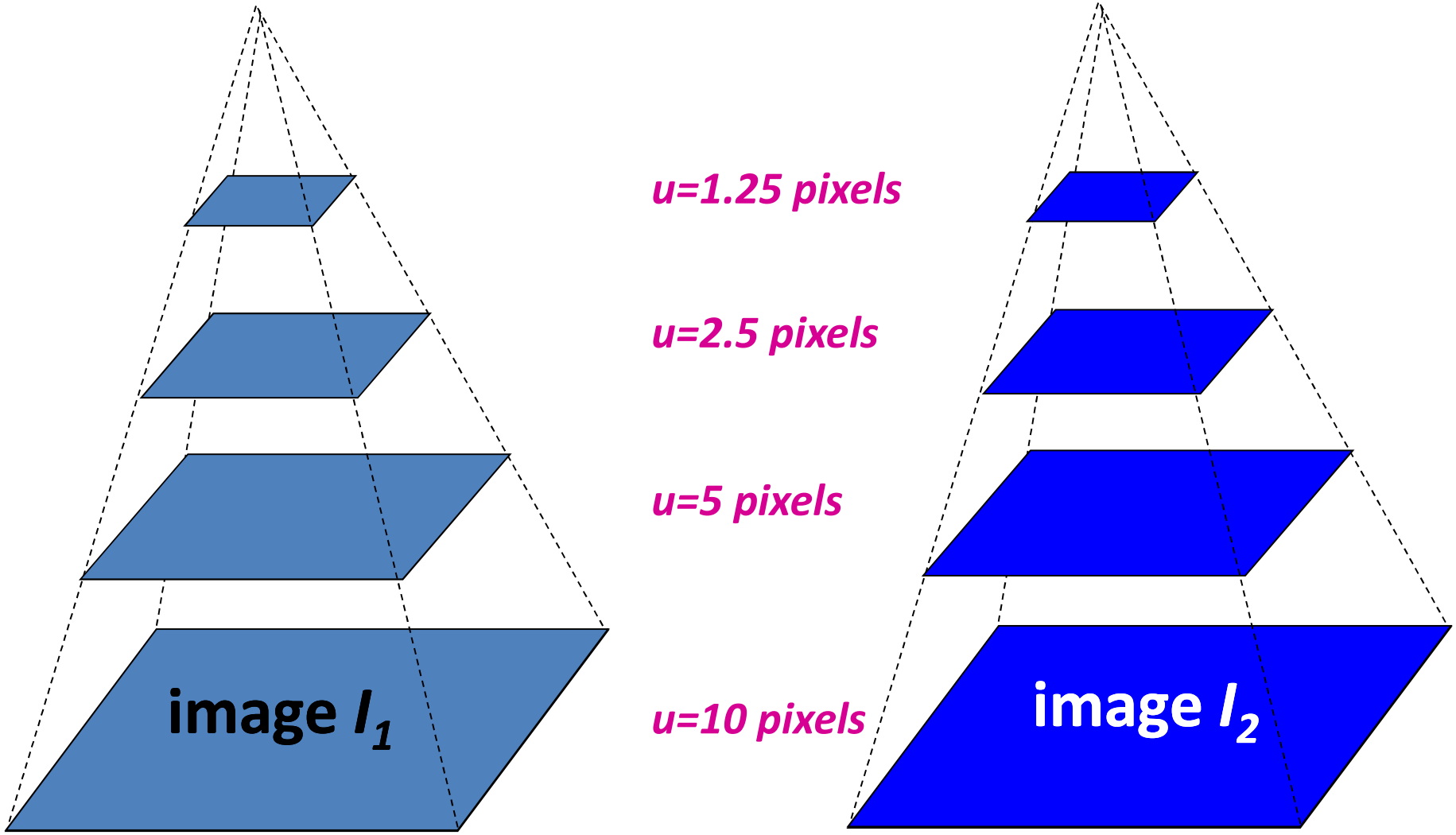
$$C(u,v) = \sum_x \sum_y I_1(x,y) \cdot I_2(x+u, y+v)$$

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

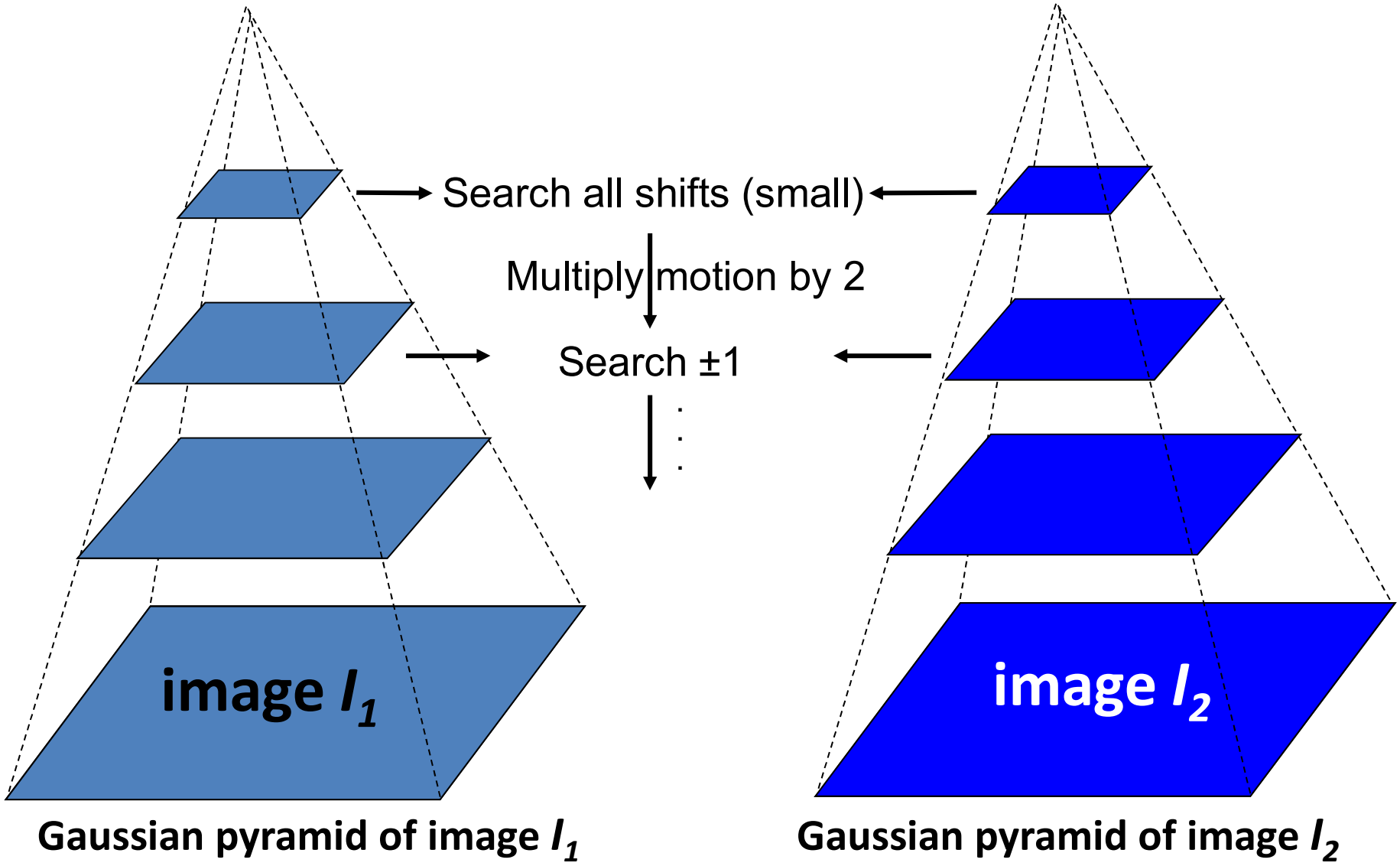
$$NC(u,v) = \frac{\sum (I_1(x,y) - \hat{I}_1) \cdot (I_2(x+u, y+v) - \hat{I}_2)}{\sqrt{\sum (I_1(x,y) - \hat{I}_1)^2} \sqrt{\sum (I_2(x,y) - \hat{I}_2)^2}}$$

- Multiresolution search (Pyramids) increases efficiency.

Coarse-to-fine motion estimation



Coarse-to-fine Image Alignment



Pattern Matching / Tracking: Normalized Cross Correlation

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

- 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 necessary in tracking.

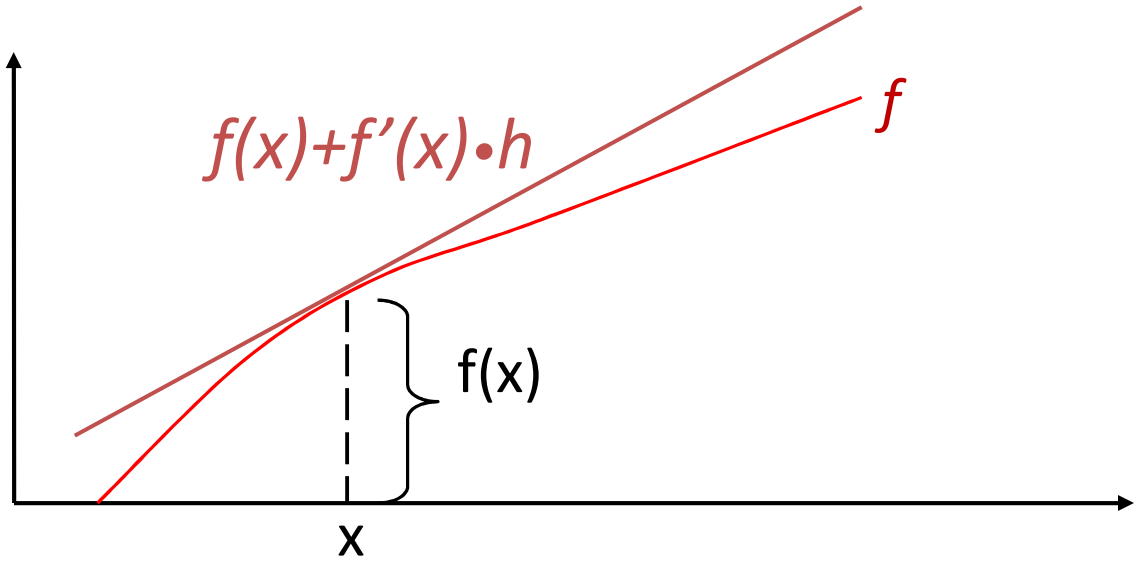
Limitations of Correlation Search

- Discrete accuracy: checking every possible translation in integer pixel values
- 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

Continuous Approximation
(Lucas Kanade)

- Local Taylor approximation in 1D:

$$f(x+h) \approx f(x) + f'(x) \cdot h$$



- Local Taylor approximation in 2D for images:

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

Error Minimization

- Accurate only for very small *(u,v), approx. 1 pixel*

- MSE when shifting I_2 relative to I_1 by *(u,v):*

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

- To simplify, we look at a single pixel (No ∑ ∑) *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

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

  - 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

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

Minimization: Zero Derivatives

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

- Finding *(u,v)* by setting derivatives to zero:

$$\frac{\partial E}{\partial u} = \sum_{x,y} I_x \cdot (I_x \cdot u + I_y \cdot v + I_t) = 0$$

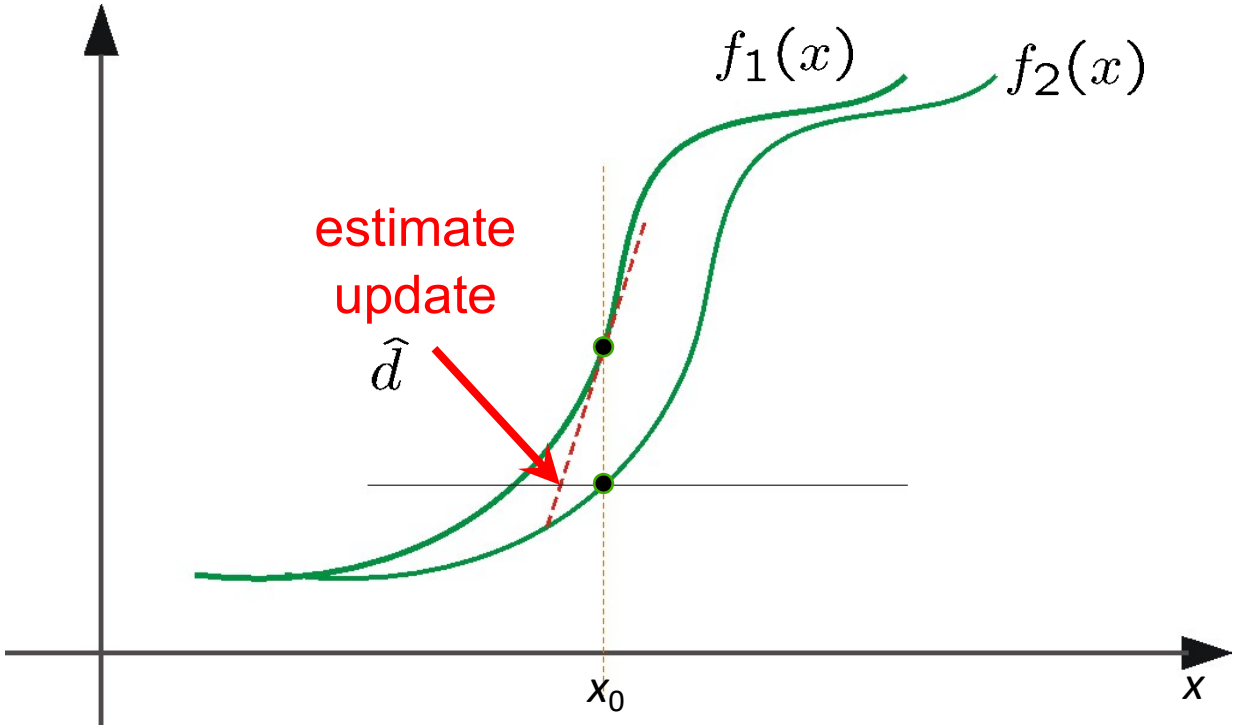
$$\frac{\partial E}{\partial v} = \sum_{x,y} I_y \cdot (I_x \cdot u + I_y \cdot v + I_t) = 0$$

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

Iterative Approach (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 total motion with residual motion: *u+=du*, *v+=dv*
  - **Warp** I_2 towards I_1 with total motion *(u,v)*.

Iterative Motion Estimation



Initial: $u_0=0$

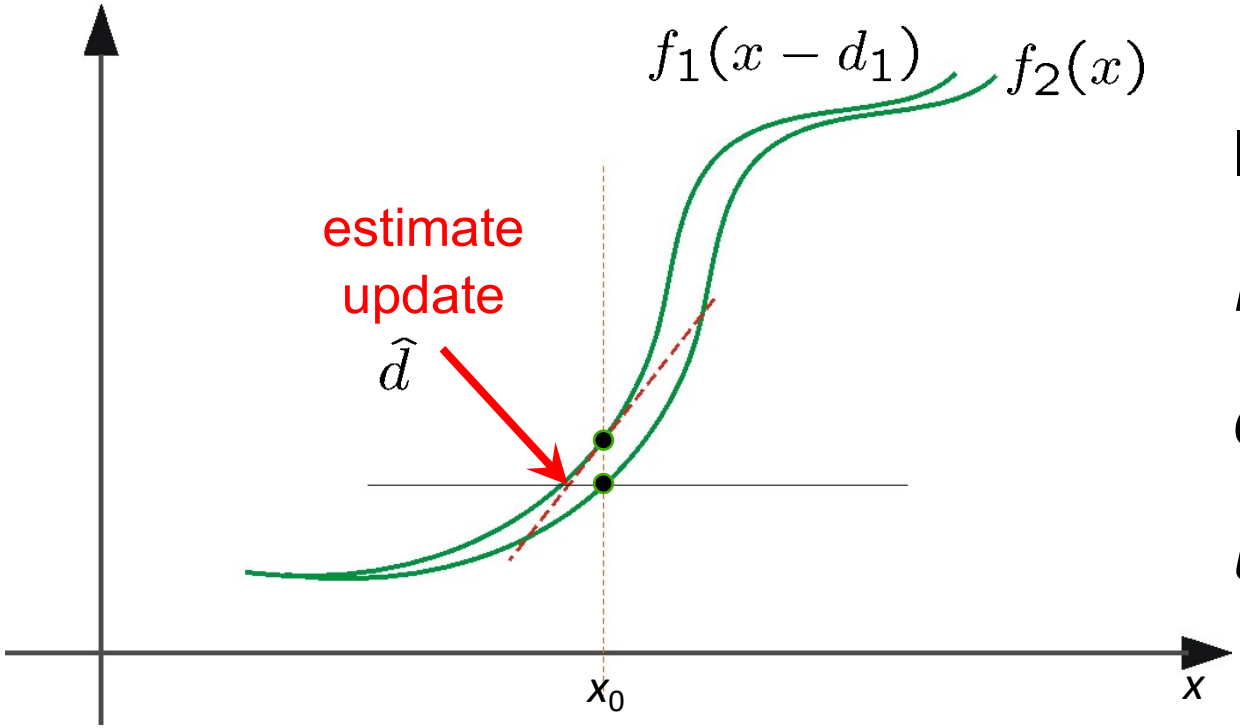
$I_t=f_1(x_0)-f_2(x_0)$

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

$u_1=u_0+du$

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

Iterative Motion Estimation



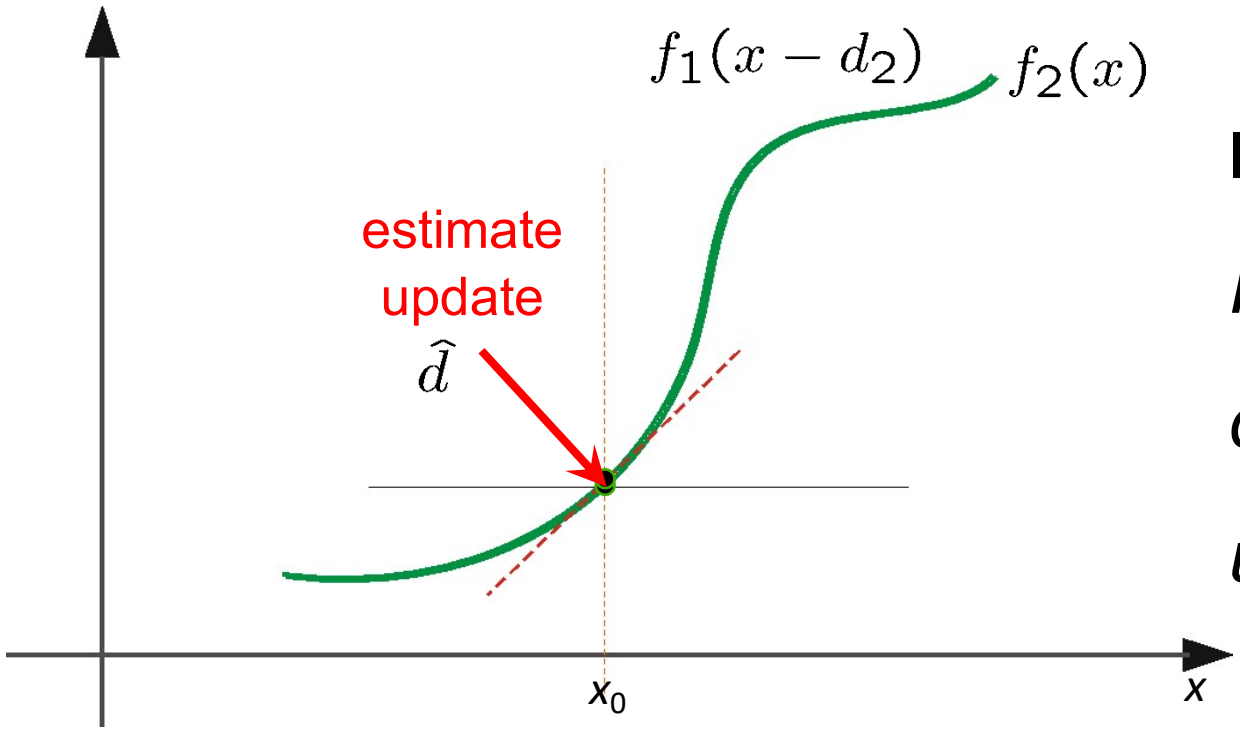
Initial: u_1

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

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

$u_2=u_1+du$

Iterative Motion Estimation



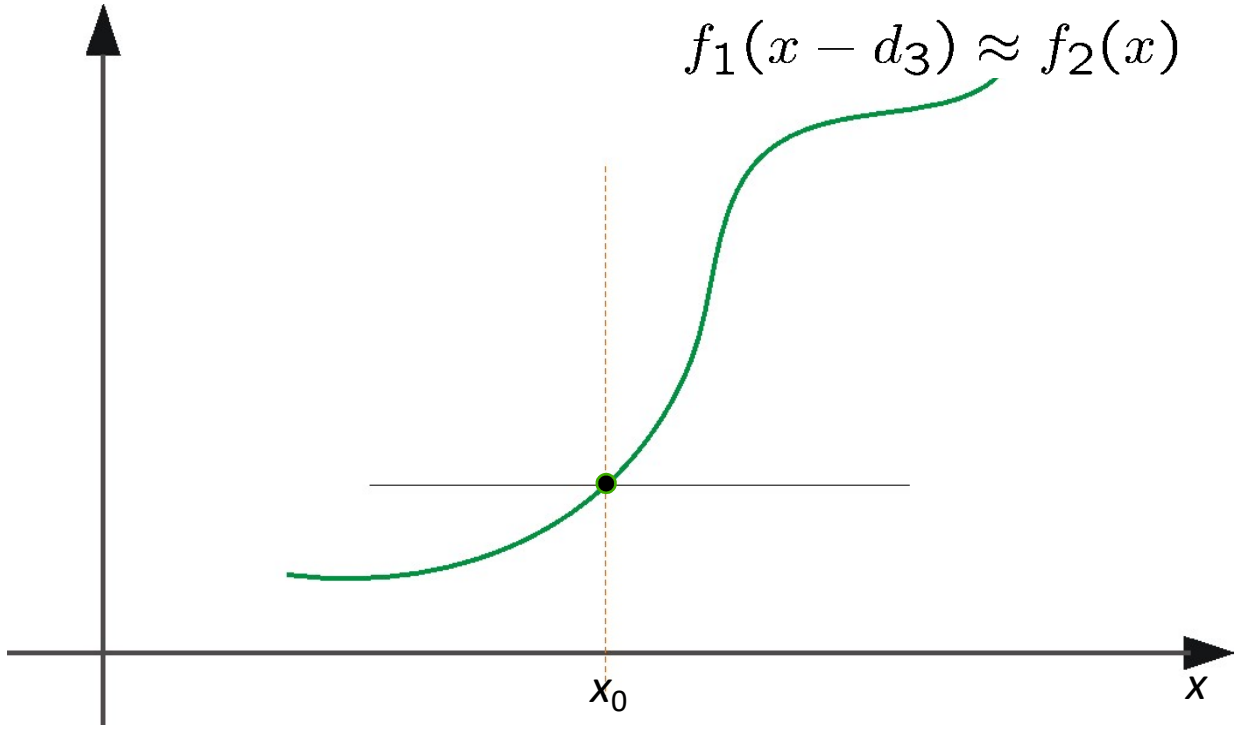
Initial: u_2

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

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

$u_3=u_2+du$

Iterative Motion Estimation

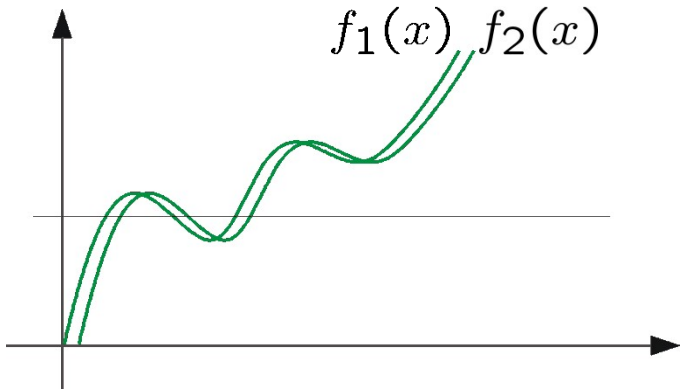


Power of Iterations

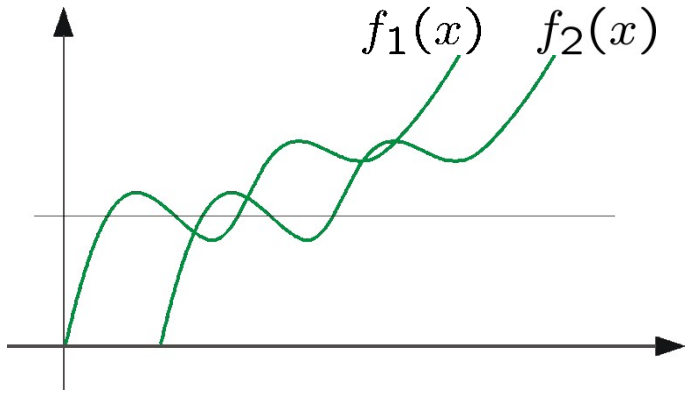
- Compute the image derivatives only once
- Has two stages in each iteration:
  - Motion Estimation
  - Warping
- Works even with poor motion estimation, as long as it reduces the residual error
- Warping of one image towards the other is done from original image using total motion, and not from previous 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 is incorrect for very large motions.



Most areas have similar derivatives

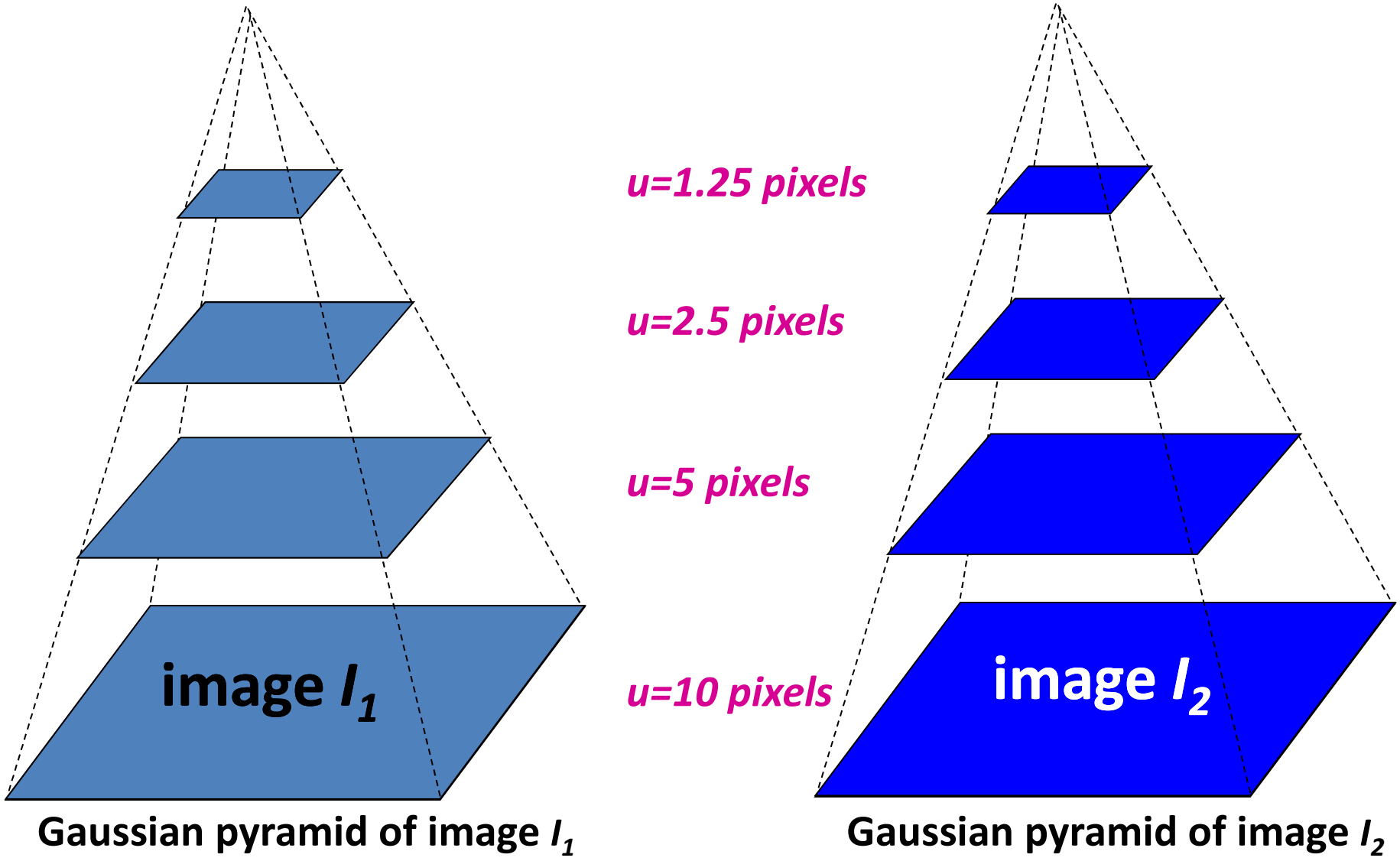


Uncorrelated derivatives (opposite signs)

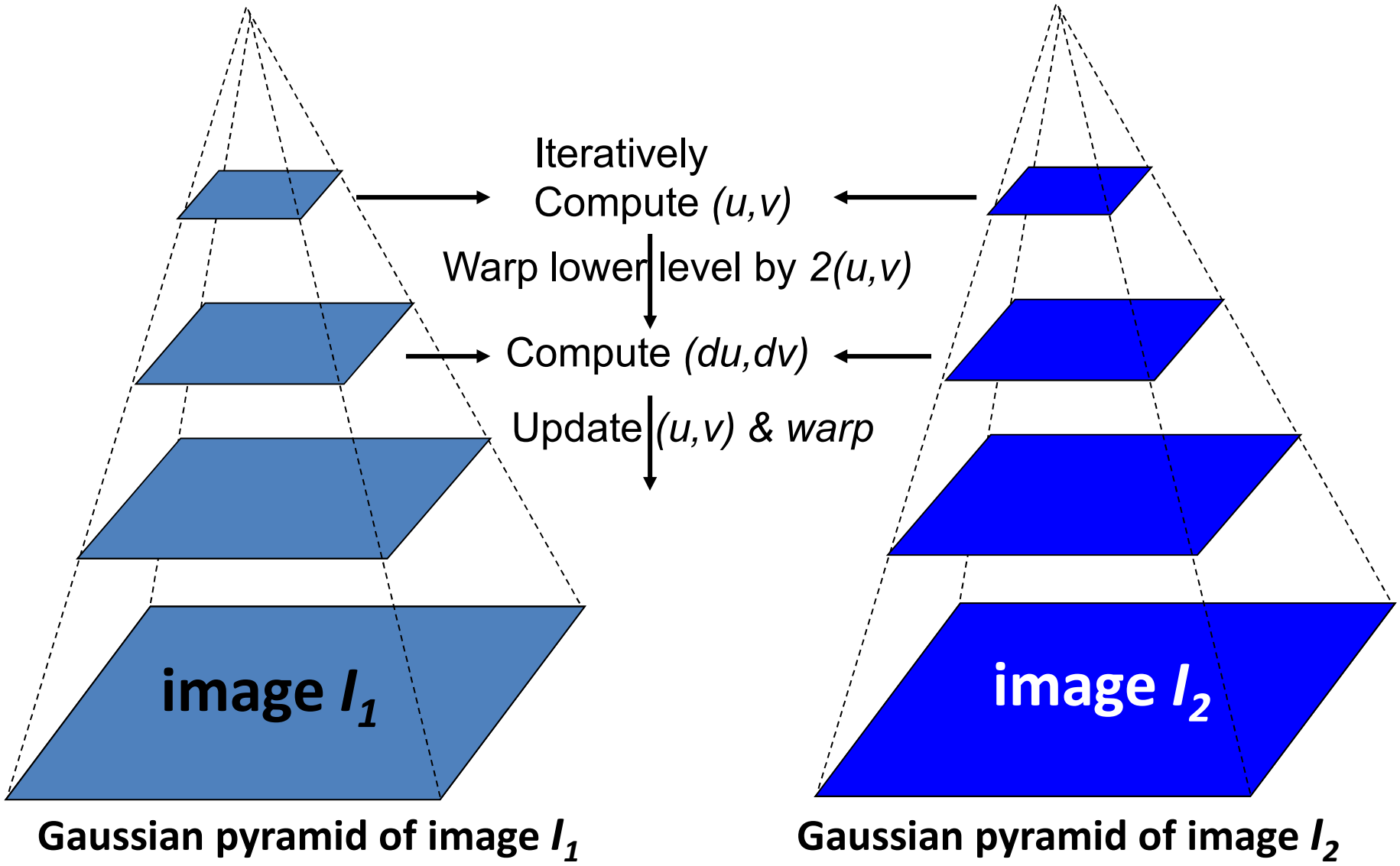
To overcome this problem: coarse-to-fine estimation.

Coarse-to-fine motion estimation

Needs very small (u,v)



Coarse-to-fine Image Alignment



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,v)* for translation and scale

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

- 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 *dx*, *dy*, and *s* by using derivatives

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

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

Translation + Rotation (Small α)

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

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

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

$$\sin(\alpha) \rightarrow \alpha \quad \text{(Small } \alpha\text{)}$$

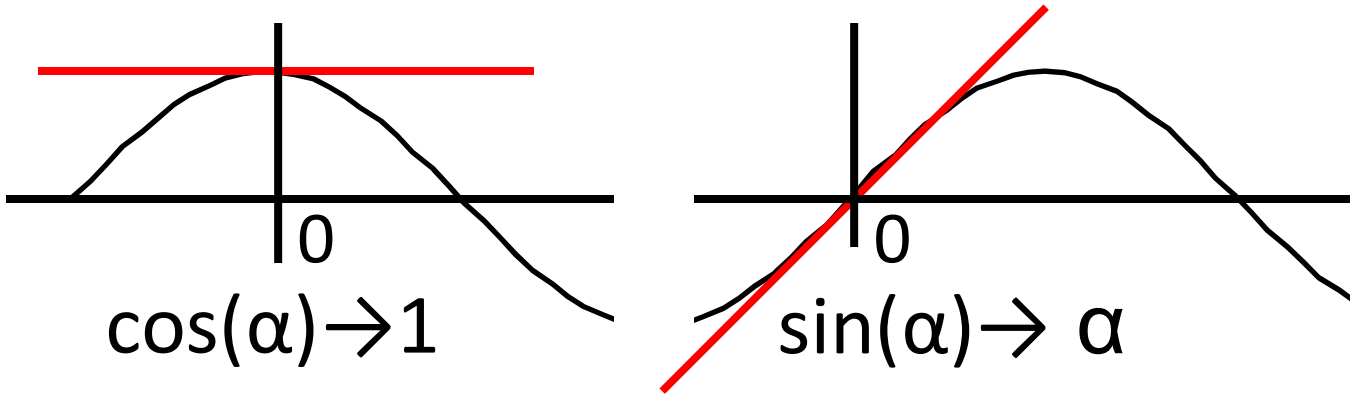
$$\cos(\alpha) \rightarrow 1 \quad \text{(Small } \alpha\text{)}$$

$$u = x_2 - x_1 = -\alpha\cdot y_1 + dx$$

$$v = y_2 - y_1 = \alpha\cdot x_1 + dy$$

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

Small α Assumption



- The “small α assumption” is used only for motion estimation (solving the equations)
- Warping is done with full accuracy of sin and cos
- Iterations converge to an accurate solution

Translation + Rotation

(unverified)

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

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

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

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

- Iterations: Solve with “small α assumption”
- Warp with full accuracy of sin and cos.
- Pyramids: Angle 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

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

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