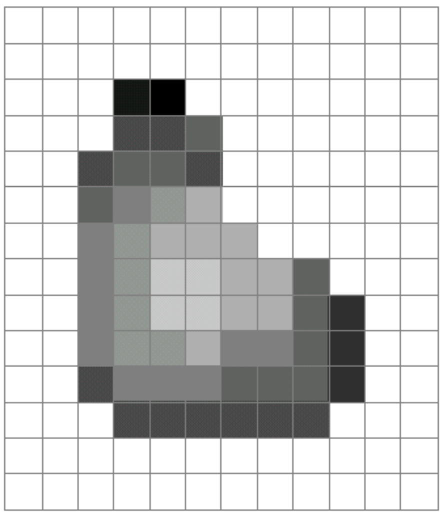
Topics

- Image Filtering
  - Image
  - Linear filter
  - Cross-correlation
  - Convolution
  - Mean filter
  - Gaussian Filter
  - Image Sharpening

What is an image?

- A grid (matrix) of intensity values



=

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

(common to use one byte per value: 0 = black, 255 = white)

Filters

- Filtering
  - Form a new image whose pixels are a combination of the original pixels
- Why?
  - To get useful information from images
    - E.g., extract edges or contours (to understand shape)
  - To enhance the image
    - E.g., to remove noise
    - E.g., to sharpen or to “enhance image”

Image filtering

- Modify the pixels in an image based on some function of a local neighborhood of each pixel

Source: L. Zhang

Linear filtering

- One simple version of filtering: linear filtering (cross-correlation, convolution)
  - Replace each pixel by a linear combination (a weighted sum) of its neighbors
- The prescription for the linear combination is called the “kernel” (or “mask”, “filter”)

10	5	3
4	6	1
1	1	8

Local image data

0	0	0
0	0.5	0
0	1	0.5

kernel

	8	

Modified image data

Source: L. Zhang

Cross-correlation

Let F be the image, H be the kernel (of size 2*k*+1 x 2*k*+1), and G be the output image

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

This is called a **cross-correlation** operation:

$$G = H \otimes F$$

- Can think of as a "dot product" between local neighborhood and kernel for each pixel

Convolution

- Same as cross-correlation, except that the kernel is “flipped” (horizontally and vertically)

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

This is called a **convolution** operation:

$$G = H * F$$

Mean filtering

H

$*$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

F

$=$

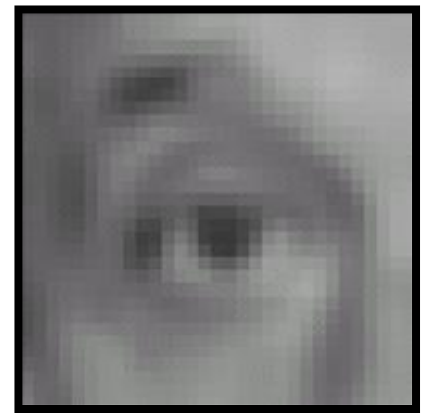
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

G

Linear filters: Mean filter

Original

$$* \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$



Blur (with a mean filter)

Source: D. Lowe

Linear filters: examples

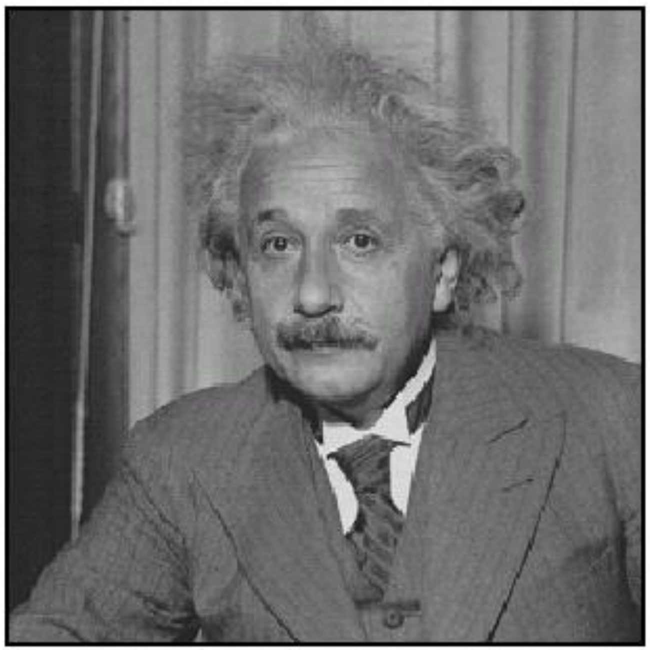
Original

$$* \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) =$$

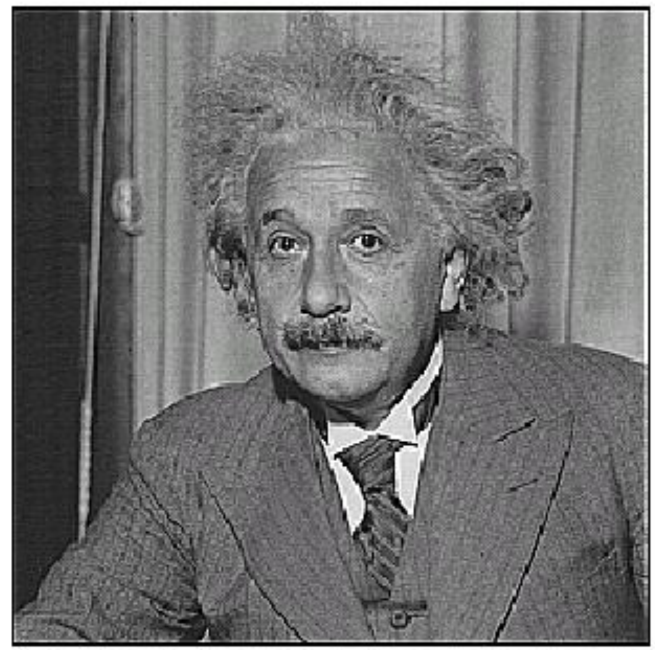
Sharpening filter

Source: D. Lowe

Sharpening



before

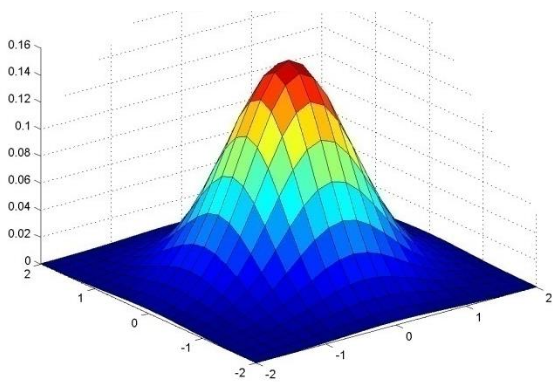


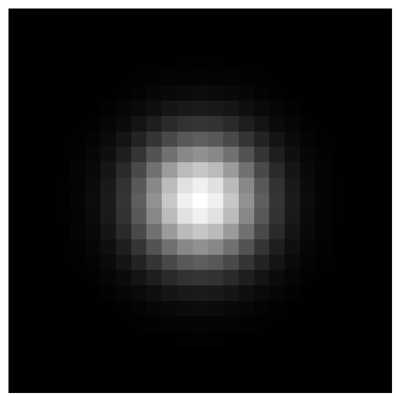
after

Source: D. Lowe

Gaussian Kernel

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$





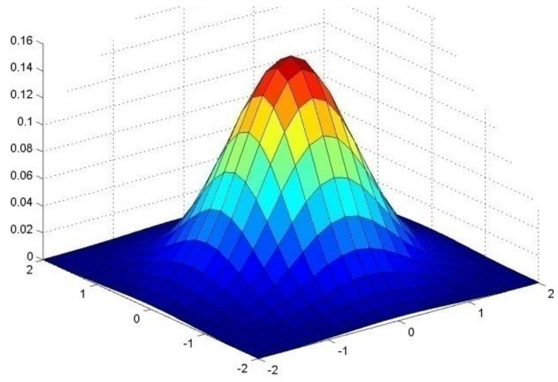
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

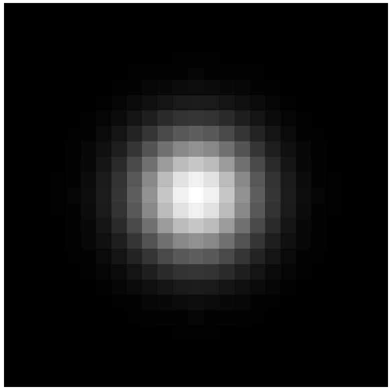
5 x 5, $\sigma = 1$

- Constant factor at front makes volume sum to 1 (can be ignored, as we should re-normalize weights to sum to 1 in any case)

Source: C. Rasmussen

Gaussian Kernel





$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

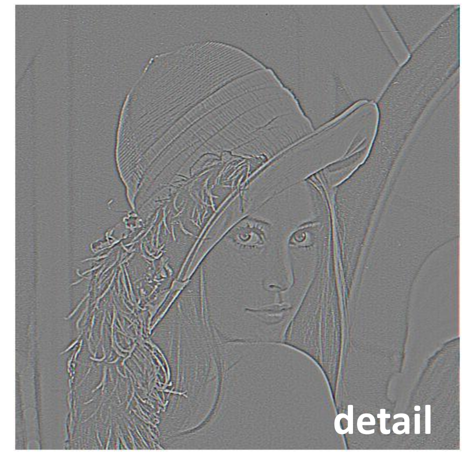
Source: C. Rasmussen

Sharpening revisited

- What does blurring take away?

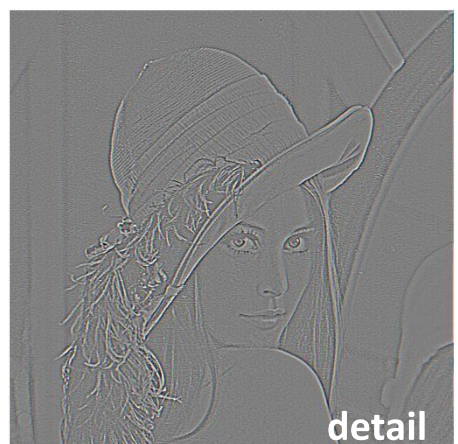
–

=



Let’s add it back:

+ α



=

Source: S. Lazebnik

Topics

- Edge Detection
  - Image derivatives
  - Image gradient
  - Sobel

Image derivatives

- How can we differentiate a *digital* image F[x,y]?
  - Option 1: reconstruct a continuous image, *f,* then compute the derivative
  - Option 2: take discrete derivative (finite difference)

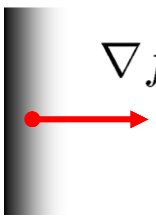
$$\frac{\partial f}{\partial x}[x, y] \approx F[x+1, y] - F[x, y]$$

Source: S. Seitz

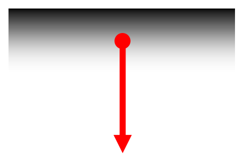
Image gradient

- The *gradient* of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

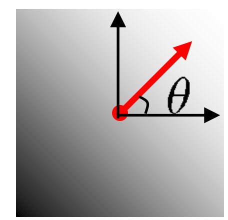
The gradient points in the direction of most rapid increase in intensity



$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$



$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$



$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

The *edge strength* is given by the gradient magnitude:

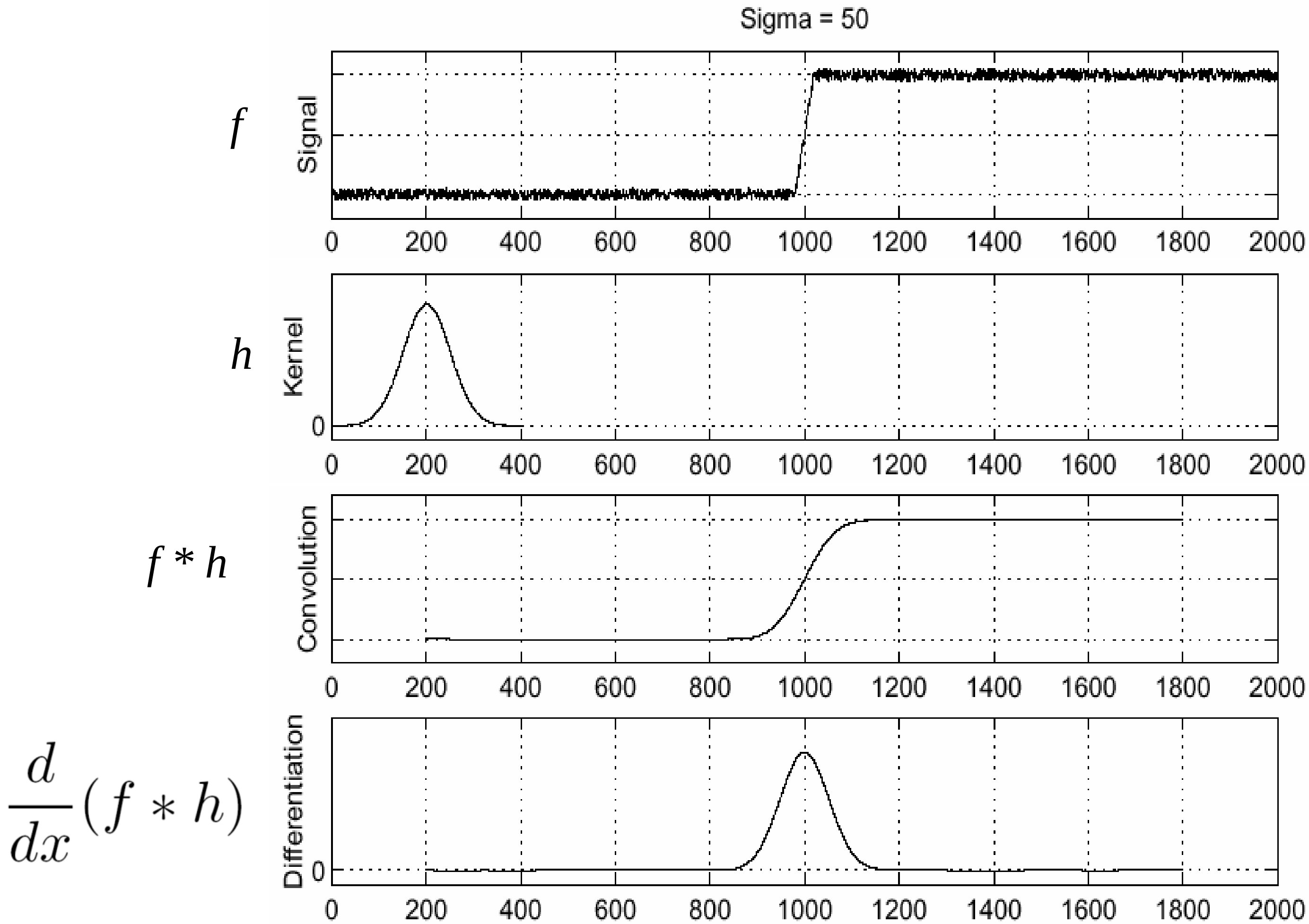
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

Source: Steve Seitz

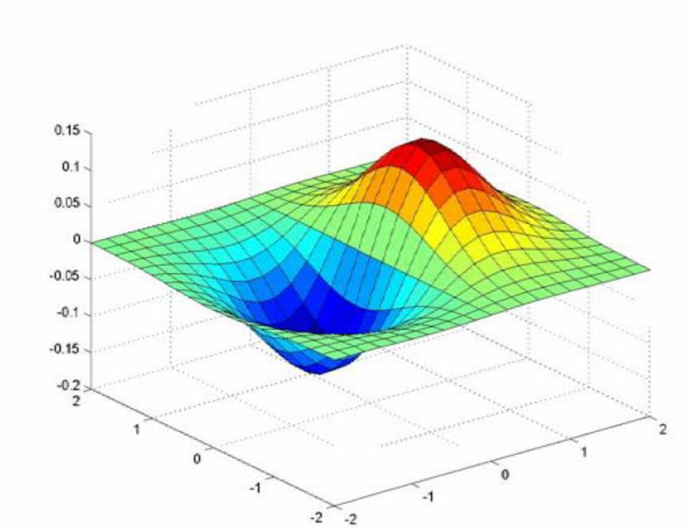
Edge detection: smooth first



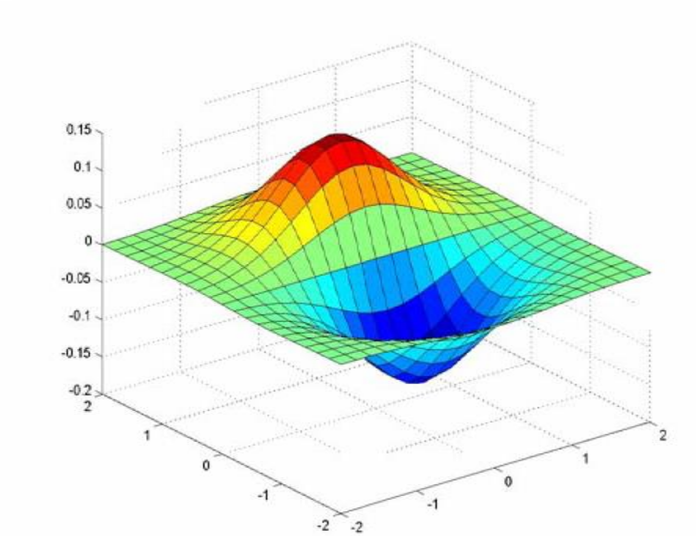
To find edges, look for peaks in $\frac{d}{dx}(f * h)$

Source: S. Seitz

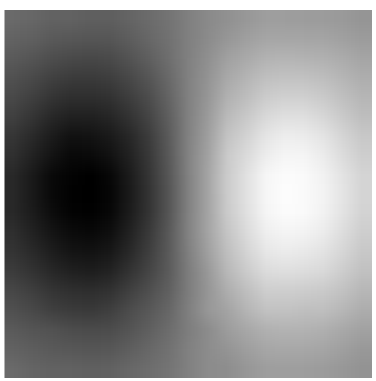
Derivative of Gaussian filter

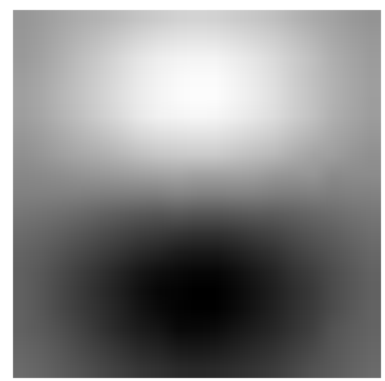


x-direction



y-direction





The Sobel operator

- Common approximation of derivative of Gaussian

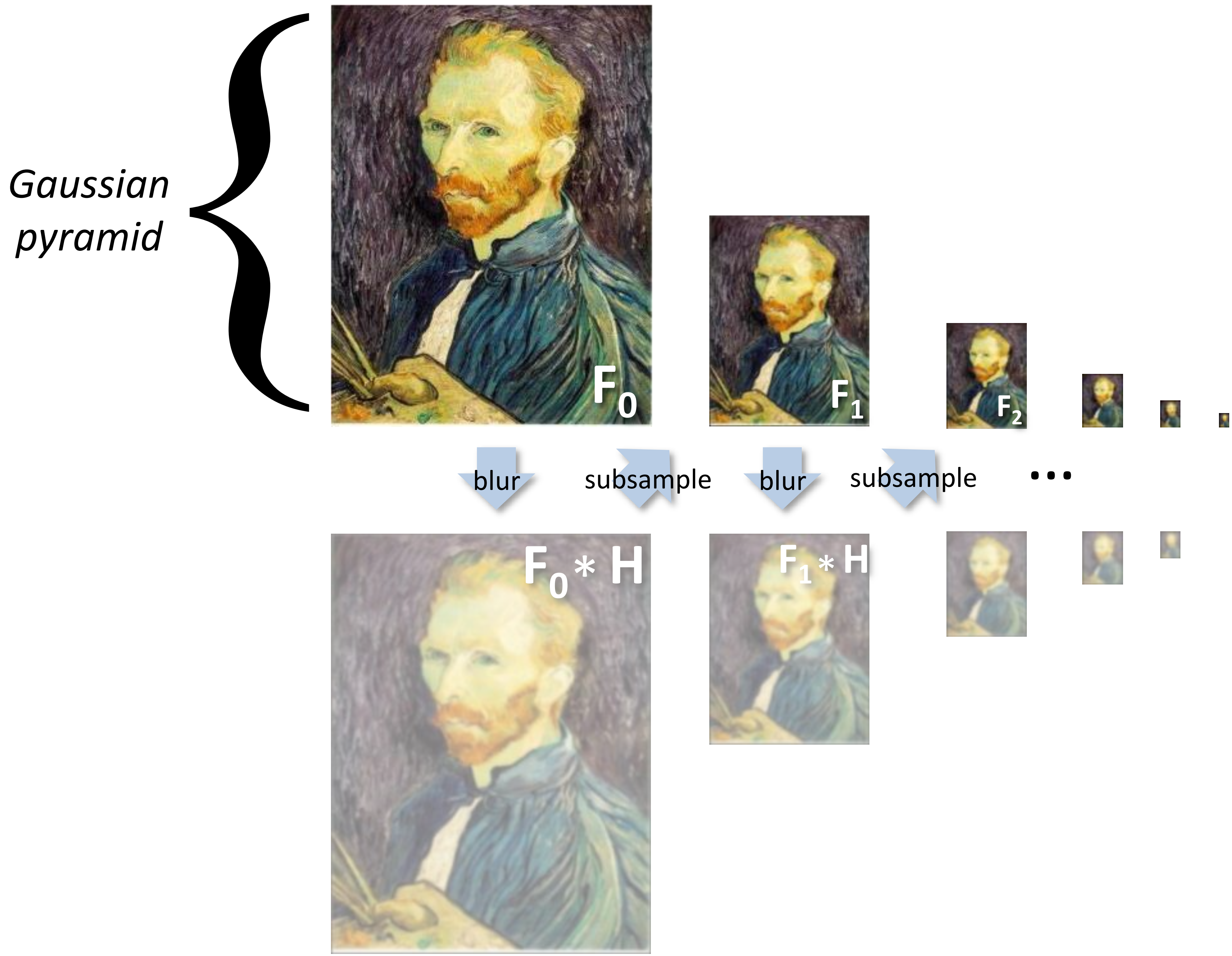
$$\frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad \frac{1}{8}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

$$s_x \qquad\qquad s_y$$

- The standard defn. of the Sobel operator omits the 1/8 term
  - doesn’t make a difference for edge detection
  - the 1/8 term **is** needed to get the right gradient magnitude

Topics

- Image Sampling
  - Gaussian pyramid
  - Image upsampling


Gaussian pyramid
F0
F1
F2
blur
subsample
blur
subsample
...
F0 * H
F1 * H

Upsampling

- This image is too small for this screen: 
- How can we make it 10 times as big?
- Simplest approach:
  repeat each row
  and column 10 times
- ("Nearest neighbor
  interpolation")

Topics

- Color and Texture
  - Histogram
  - Edge-based Texture Measures
  - Local Binary Pattern Measure
  - Co-occurrence Matrix Features

Histograms

- A histogram of a gray-tone image is an array H[*] of bins, one for each gray tone.

- H[i] gives the count of how many pixels of an image have gray tone i.

- P[i] (the normalized histogram) gives the percentage of pixels that have gray tone i.

Two Edge-based Texture Measures

1. edgeness per unit area

$$\boxed{F_{edgeness} = |\{ p \mid gradient\_magnitude(p) \geq threshold\}| / N}$$

where N is the size of the unit area

2. edge magnitude and direction histograms

$$\boxed{F_{magdir} = (H_{magnitude}, H_{direction})}$$

where these are the normalized histograms of gradient magnitudes and gradient directions, respectively.

Local Binary **Pattern** Measure

- For each pixel p, create an 8-bit number $b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7\ b_8$, where $b_i = 0$ if neighbor i has value less than or equal to p's value and 1 otherwise.

- Represent the texture in the image (or a region) by the histogram of these numbers.

	1	2	3	
	100	101	103	
8	40	50	80	4
	50	60	90	5
	7	6		

→ 1 1 1 1 1 1 0 0

Local Binary Pattern (LBP)

$$LBP_{p,r}(N_c) = \sum_{p=0}^{P-1} g\left(N_p - N_c\right)2^p$$

N_c: center pixel
N_p: neighbor pixel
r: radius (for 3x3 cell, it is 1).

binary threshold function $g(x)$ is,

$$g(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Co-occurrence Matrix Features

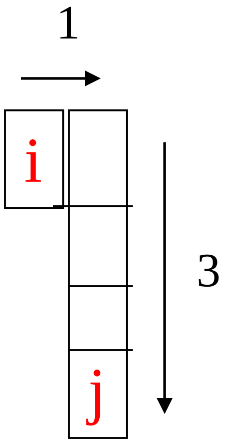
A co-occurrence matrix is a 2D array C in which

- Both the rows and columns represent a set of possible image values.
- $C_d(i,j)$ indicates how many times value i co-occurs with value j in a particular spatial relationship d.
- The spatial relationship is specified by a vector d = (dr,dc).

Co-occurrence Example

$$\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \end{array}$$

gray-tone image



$\boxed{d = (3,1)}$

$$\begin{array}{c|c} & \begin{matrix} 0 & 1 & 2 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \end{matrix} & \begin{bmatrix} 1 & 0 & 3 \\ 2 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \end{array} \quad C_d$$

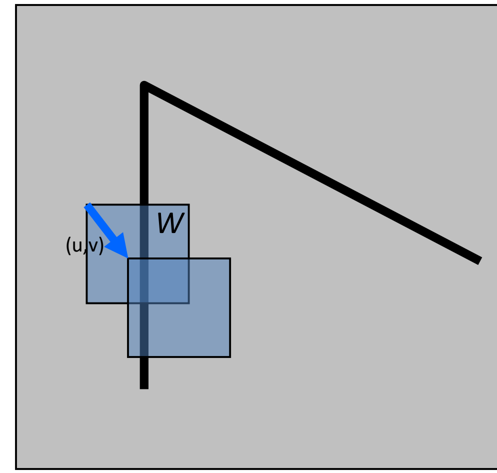
co-occurrence matrix

From C_d we can compute N_d, the normalized co-occurrence matrix, where each value is divided by the sum of all the values.

Topics

- Corners and blobs
  - Harris corner detection
  - Blob detection

Corner detection: the math



Consider shifting the window *W* by (*u,v*)

- define an SSD "error" *E(u,v)*:

$$E(u,v) \approx \sum_{(x,y)\in W} [I_x u + I_y v]^2$$

$$\approx \quad Au^2 + 2Buv + Cv^2$$

$$A = \sum_{(x,y)\in W} I_x^2 \qquad B = \sum_{(x,y)\in W} I_x I_y \qquad C = \sum_{(x,y)\in W} I_y^2$$

- Thus, *E*(*u*,*v*) is locally approximated as a quadratic error function

Corner detection: the math

The surface *E*(*u*,*v*) is locally approximated by a quadratic form.

$$E(u,v) \approx Au^2 + 2Buv + Cv^2$$

$$\approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y)\in W} I_x^2$$

$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$

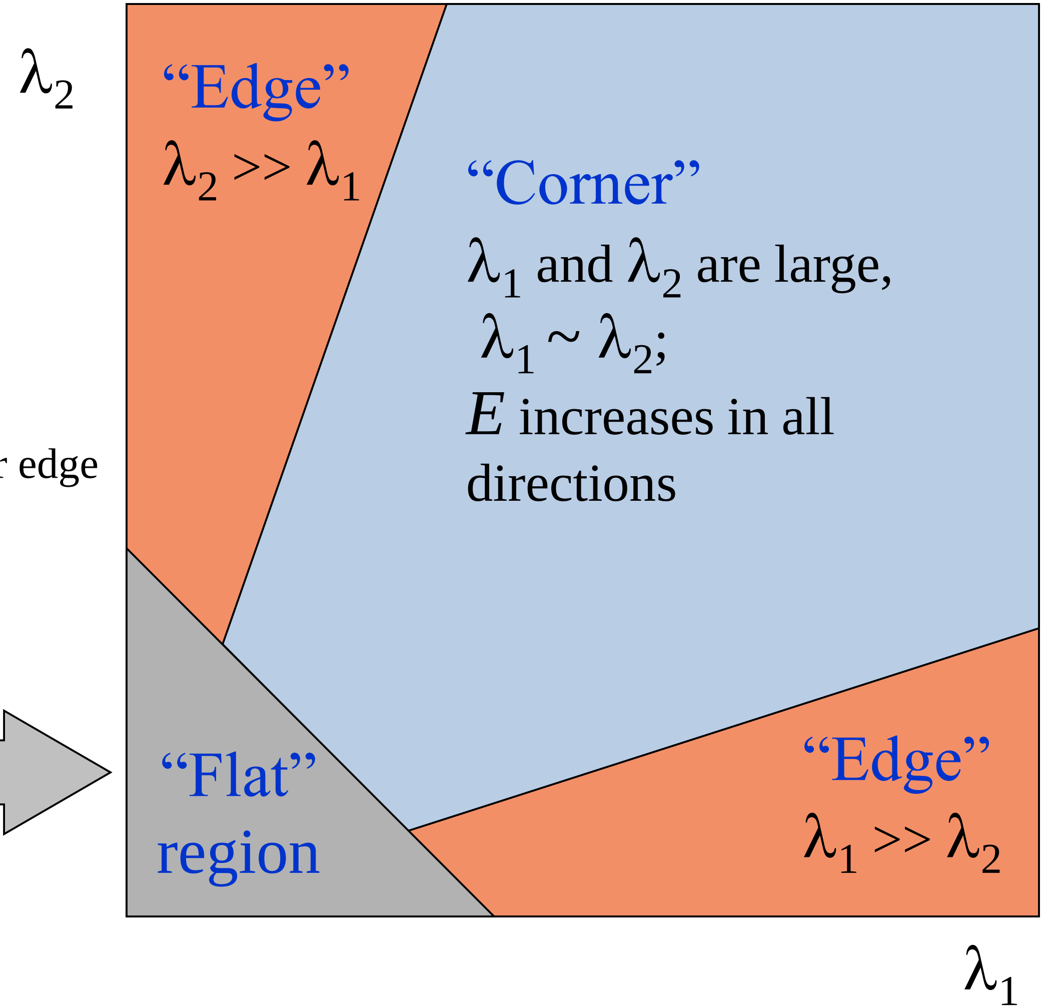
Interpreting the eigenvalues

Classification of image points using eigenvalues of *M*:

$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$

R is large for corner
R is negative (with large magnitude) for edge
R is small for flat region

λ_1 and λ_2 are small;
E is almost constant in all directions

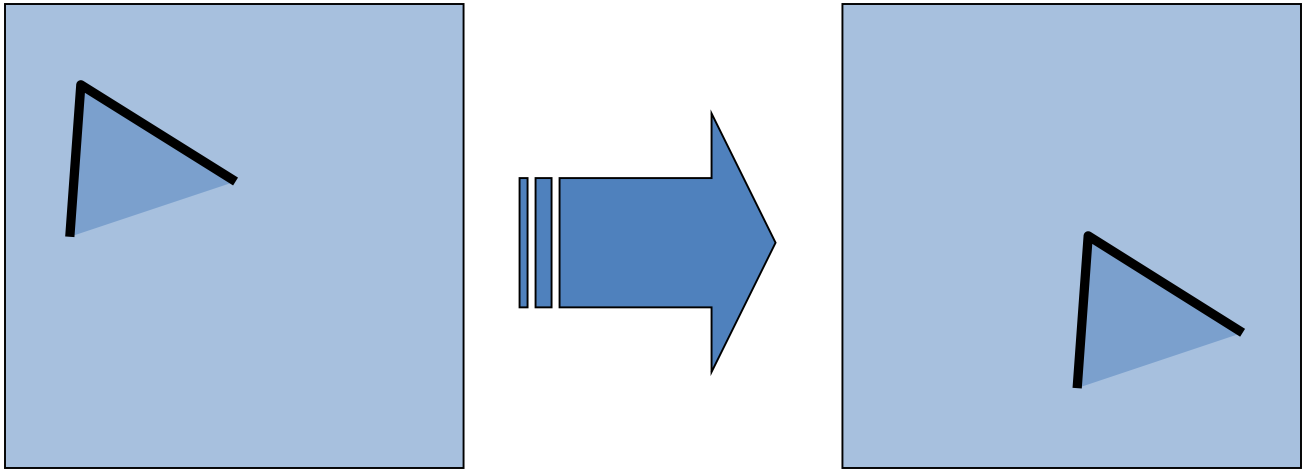


Other Versions of The Harris operator

$$
\begin{aligned}
f &= \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} \\
&= \frac{determinant(H)}{trace(H)}
\end{aligned}
$$

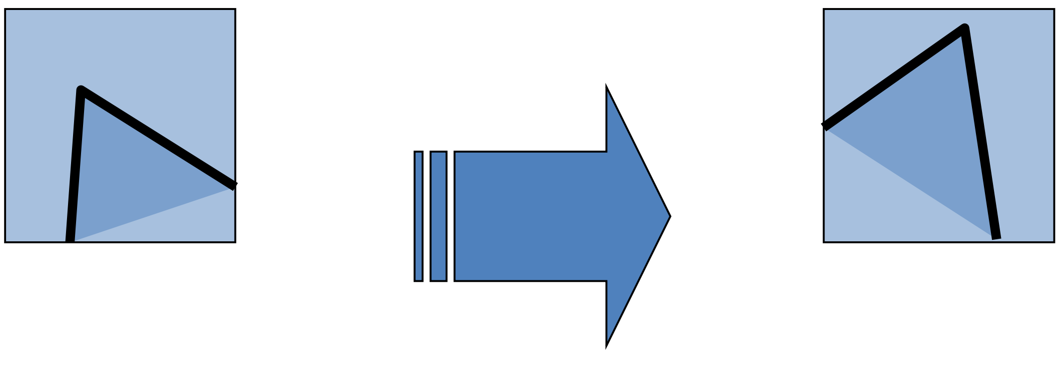
λ_{min} is a variant of the "Harris operator" for feature detection

Harris detector: Invariance properties -- Image translation

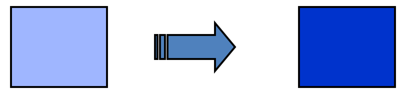


- Derivatives and window function are shift-invariant

Harris detector: Invariance properties -- Image rotation



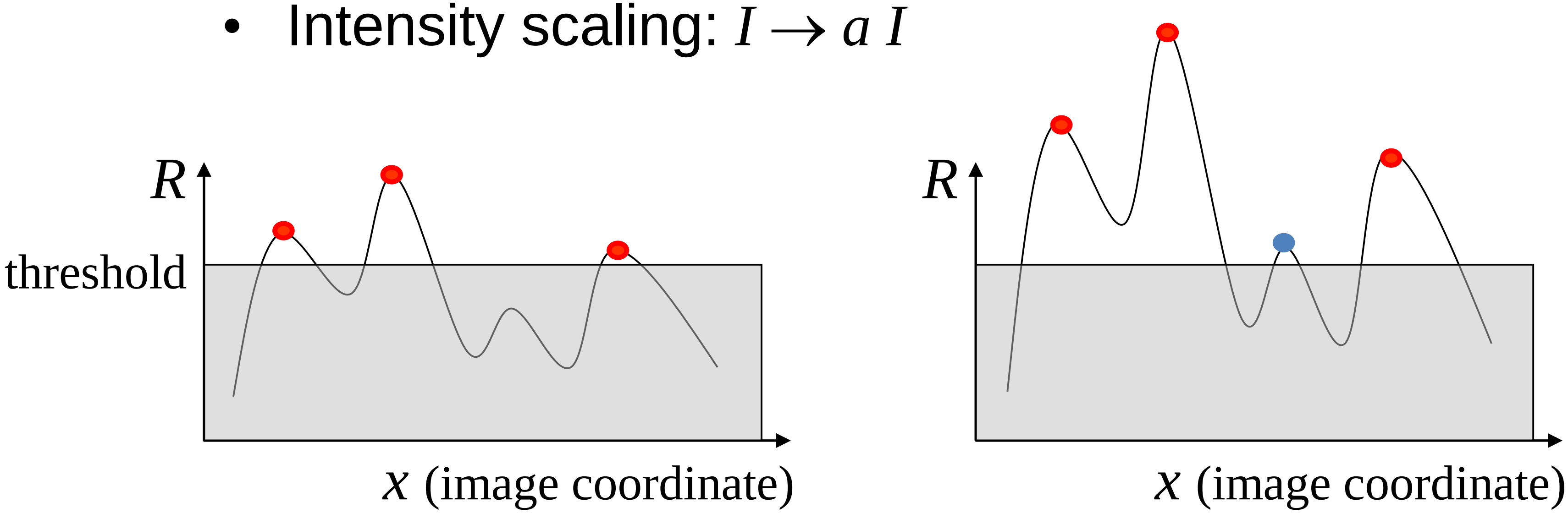
Harris detector: Invariance properties – Affine intensity change



$I \rightarrow a\,I + b$

- Only derivatives are used => invariance to intensity shift $I \rightarrow I + b$

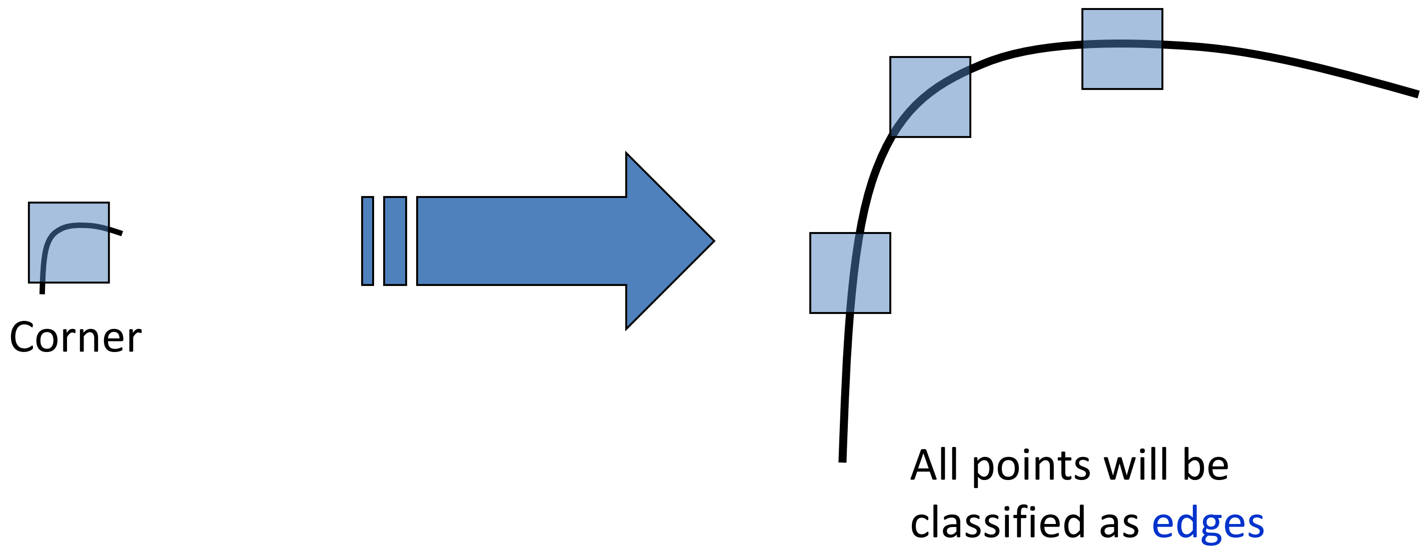
- Intensity scaling: $I \rightarrow a\,I$



Partially invariant to affine intensity change

Harris Detector: Invariance Properties

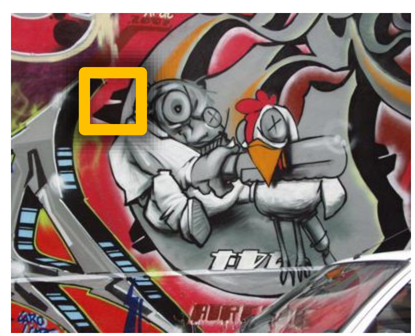
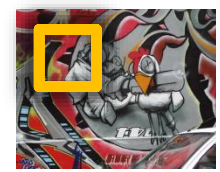
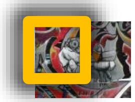
- Scaling



Not invariant to scaling

Scale Selection

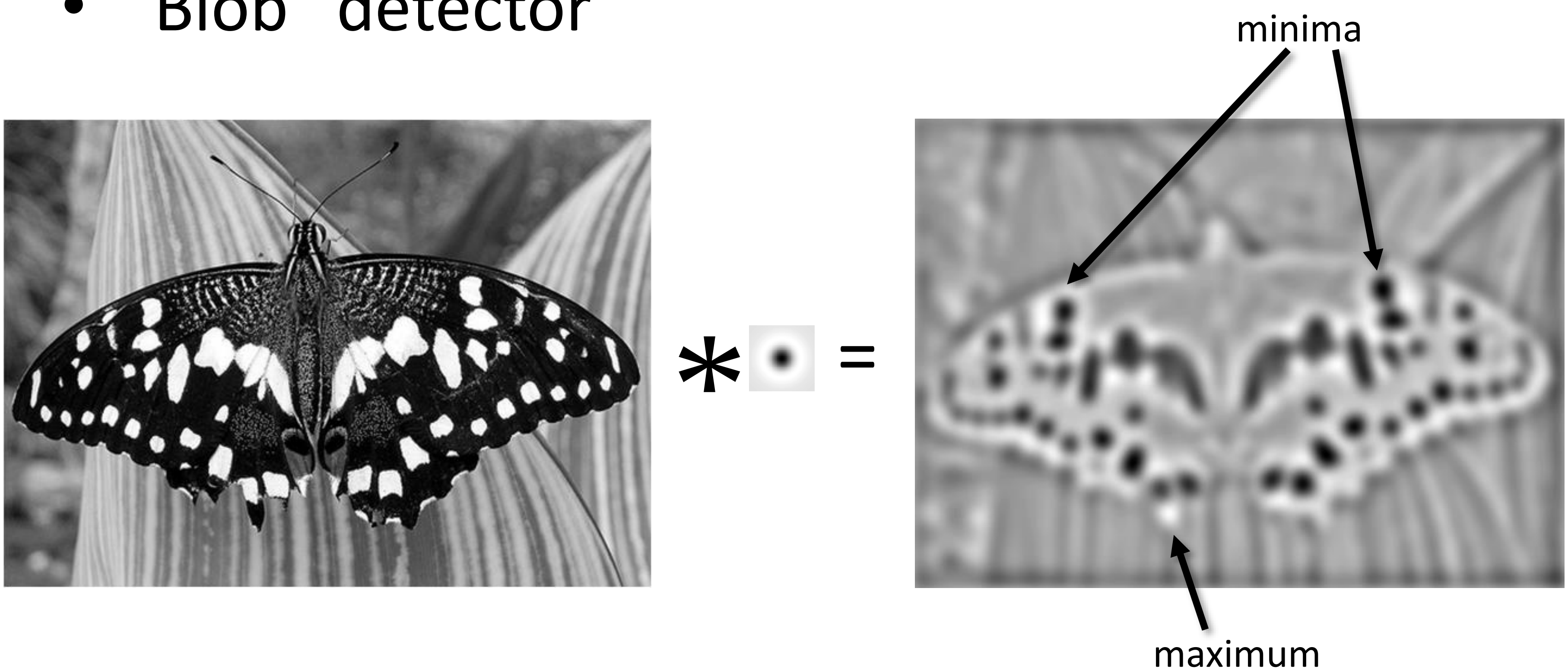
- Instead of computing *f* for larger and larger windows, we can implement using a fixed window size with a Gaussian pyramid

(sometimes need to create in-between levels, e.g. a ¾-size image)

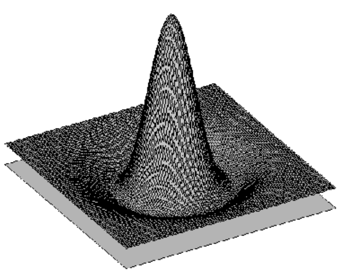
Laplacian of Gaussian

- "Blob" detector



- Find maxima *and minima* of LoG operator in space and scale

Scale-space blob detector



$$L_{xx}(\sigma) + L_{yy}(\sigma)$$

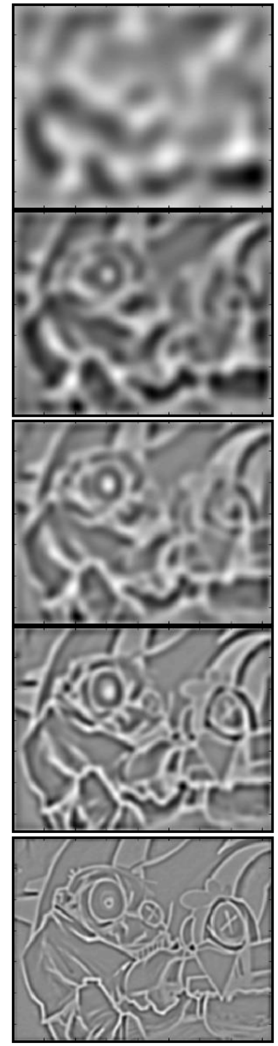
σ^5

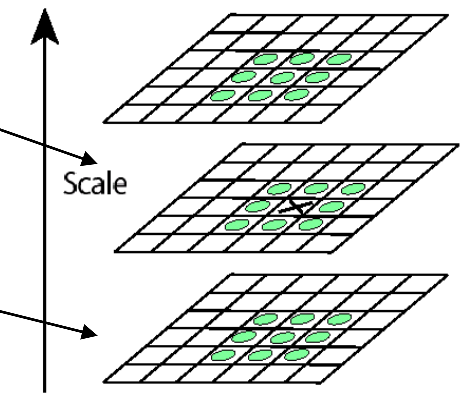
σ^4

σ^3

σ^2

σ





$\Rightarrow$ **List of *(x, y, s)***

Topics

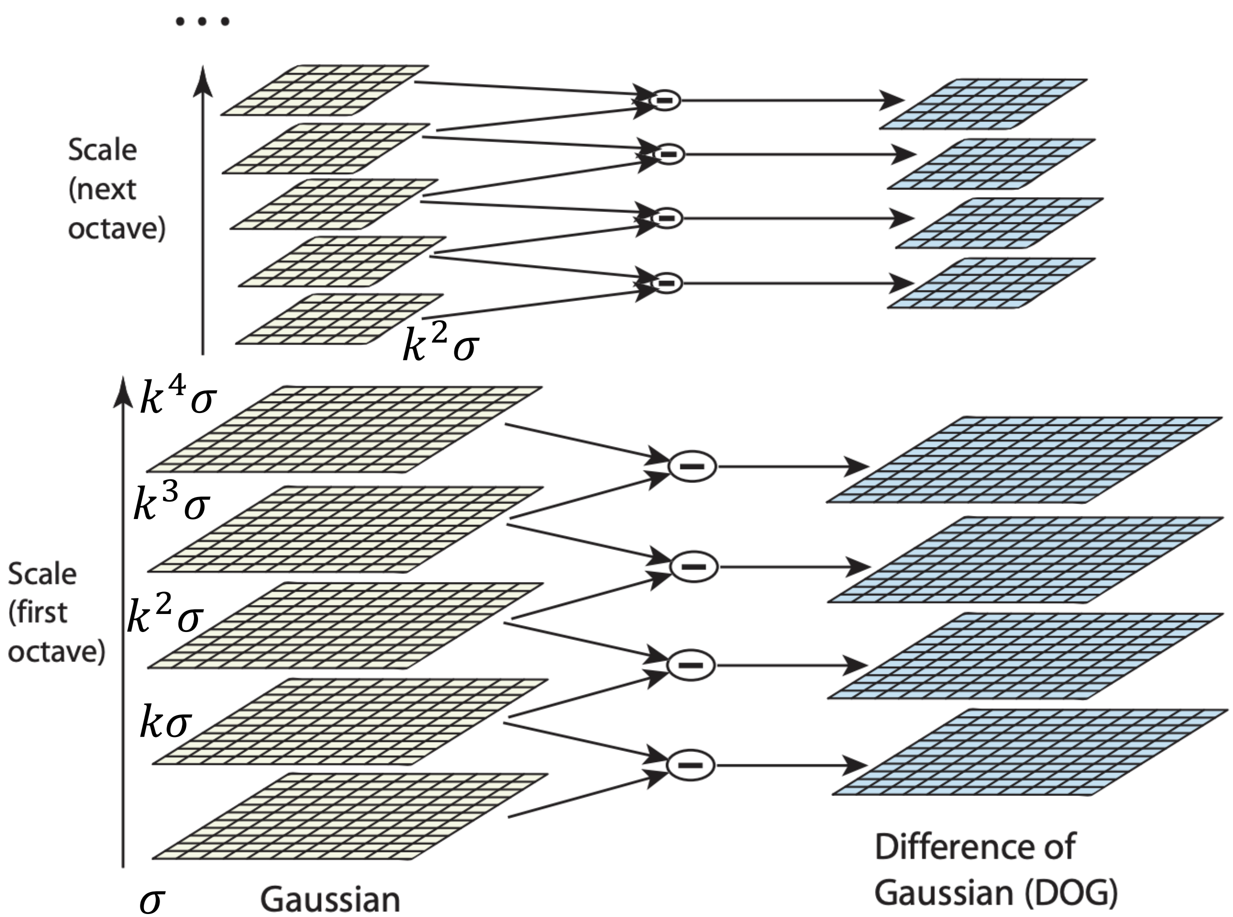
- Scale Invariant Feature Transform
  - Building scale-space
  - Interest point detection
  - Orientation assignment
  - SIFT feature descriptor
  - SIFT distance calculation

Scale Invariant Feature Transform

- Scale space peak selection
  - Potential locations for finding features
- Key point localization
  - Accurately locating the feature key points
- Orientation assignment
  - Assigning orientation to the key points
- Key point descriptor
  - Describing the key point as a high dimensional vector (128) (SIFT Descriptor)

Adapted from slide by Mubarak Shah

Building the Scale Space

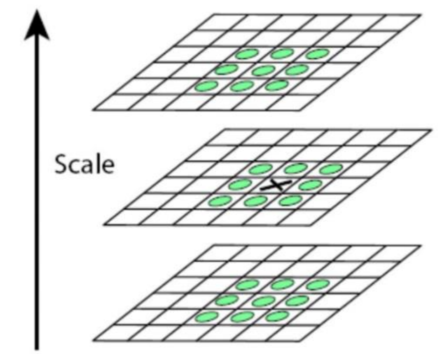


Adapted from IJCV 2004 by David Lowe

Peak Detection

Compare a pixel (**X**) with 26 pixels in current and adjacent scales (Green Circles)

Select a pixel (**X**) if larger/smaller than all 26 pixels



Adapted from slide by Mubarak Shah

Assignment of the Orientation

- An orientation histogram is formed from the gradient orientations of sample points within a region around the keypoint.

- The orientation histogram has 36 bins covering the 360 degree range of orientations.

- The samples added to the histogram is weighted by the gradient magnitude.

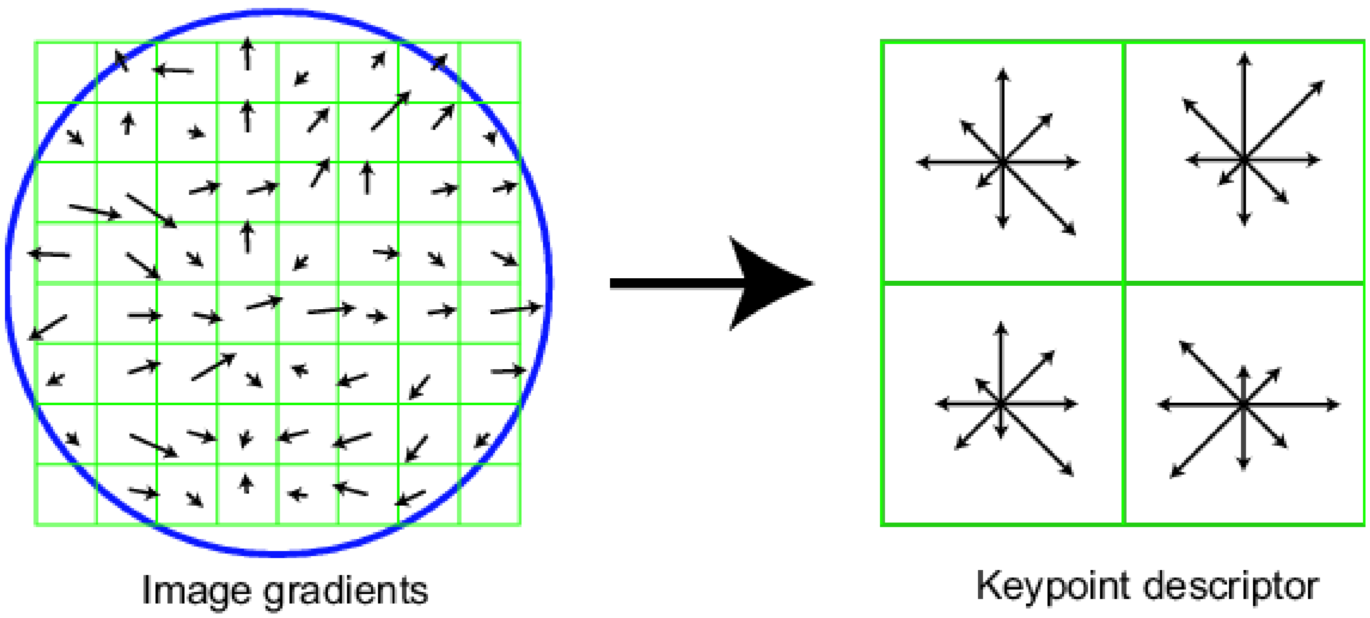
- The dominate direction is the peak in the histogram.

Adapted from IJCV 2004 by David Lowe

SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram (8 bin) for each cell (relative orientation and magnitude)
- 16 cells * 8 orientations = 128 dimensional descriptor

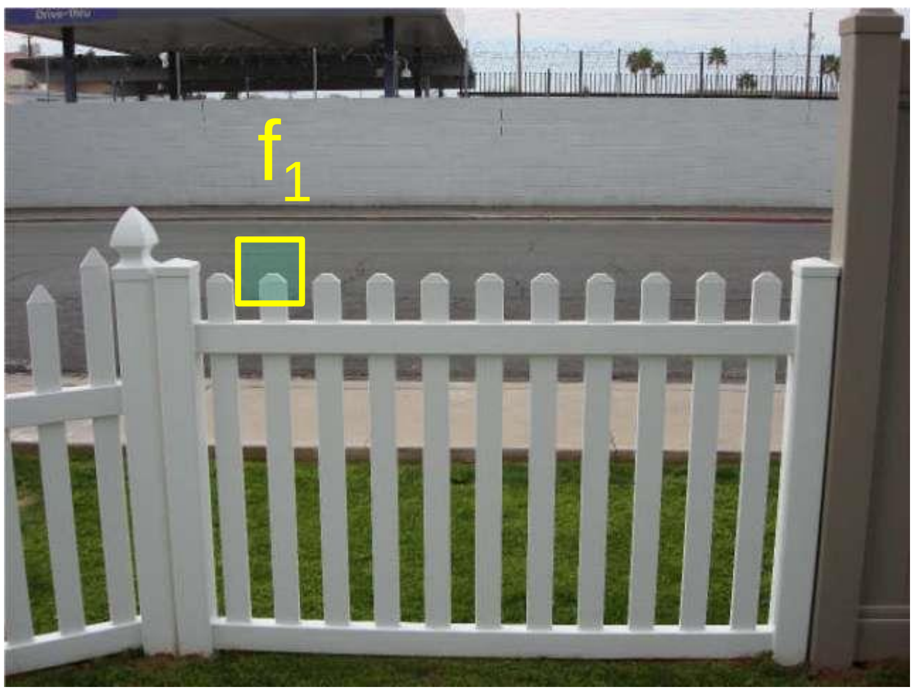


Adapted from slide by David Lowe

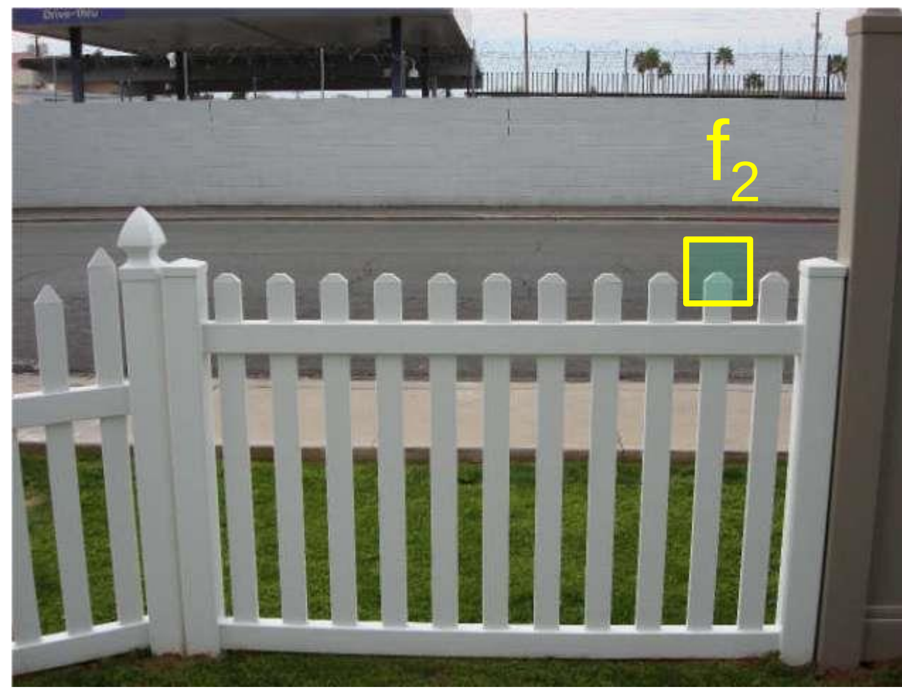
Feature distance

How to define the difference between two features f_1, f_2?

- Simple approach: L_2 distance, $||f_1 - f_2||$ (aka SSD)
- can give good scores to ambiguous (incorrect) matches



I_1

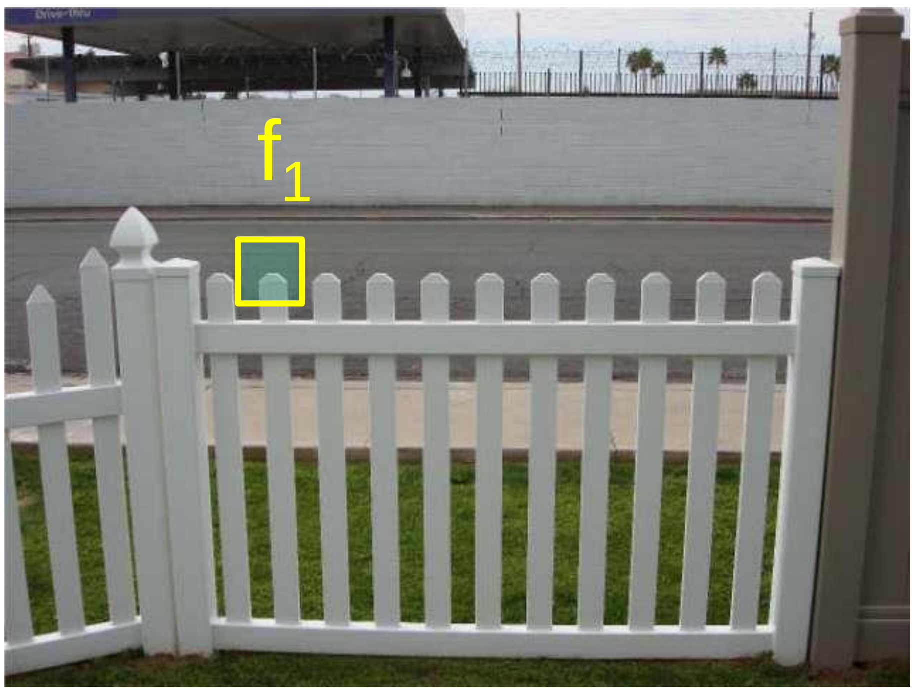


I_2

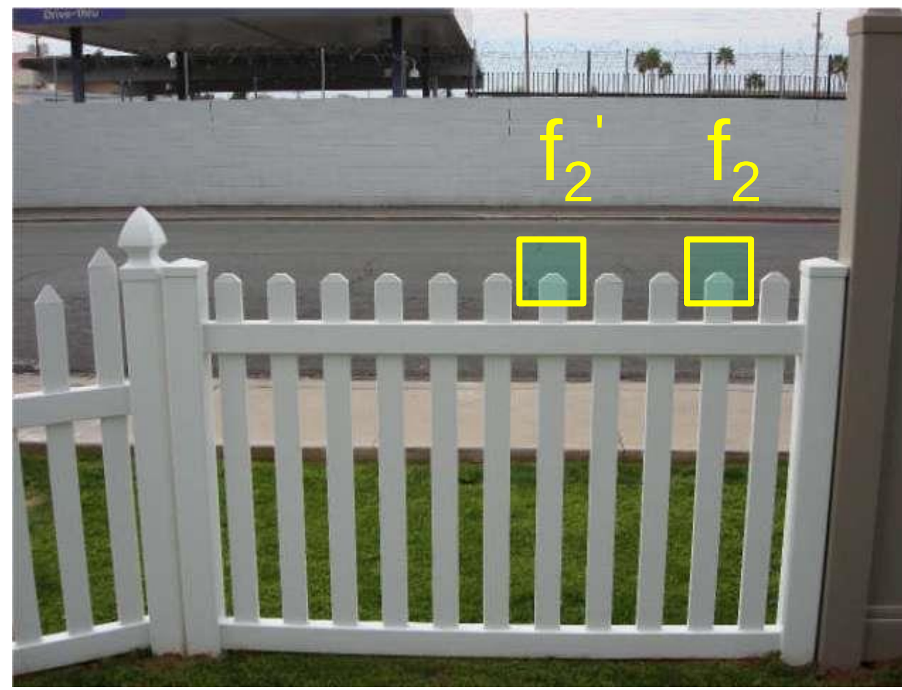
Feature distance

How to define the difference between two features f_1, f_2?

- Better approach: ratio distance = $||f_1 - f_2|| \,/\, ||f_1 - f_2'||$
  - f_2 is best SSD match to f_1 in I_2
  - f_2' is 2nd best SSD match to f_1 in I_2
  - gives large values for ambiguous matches



I_1



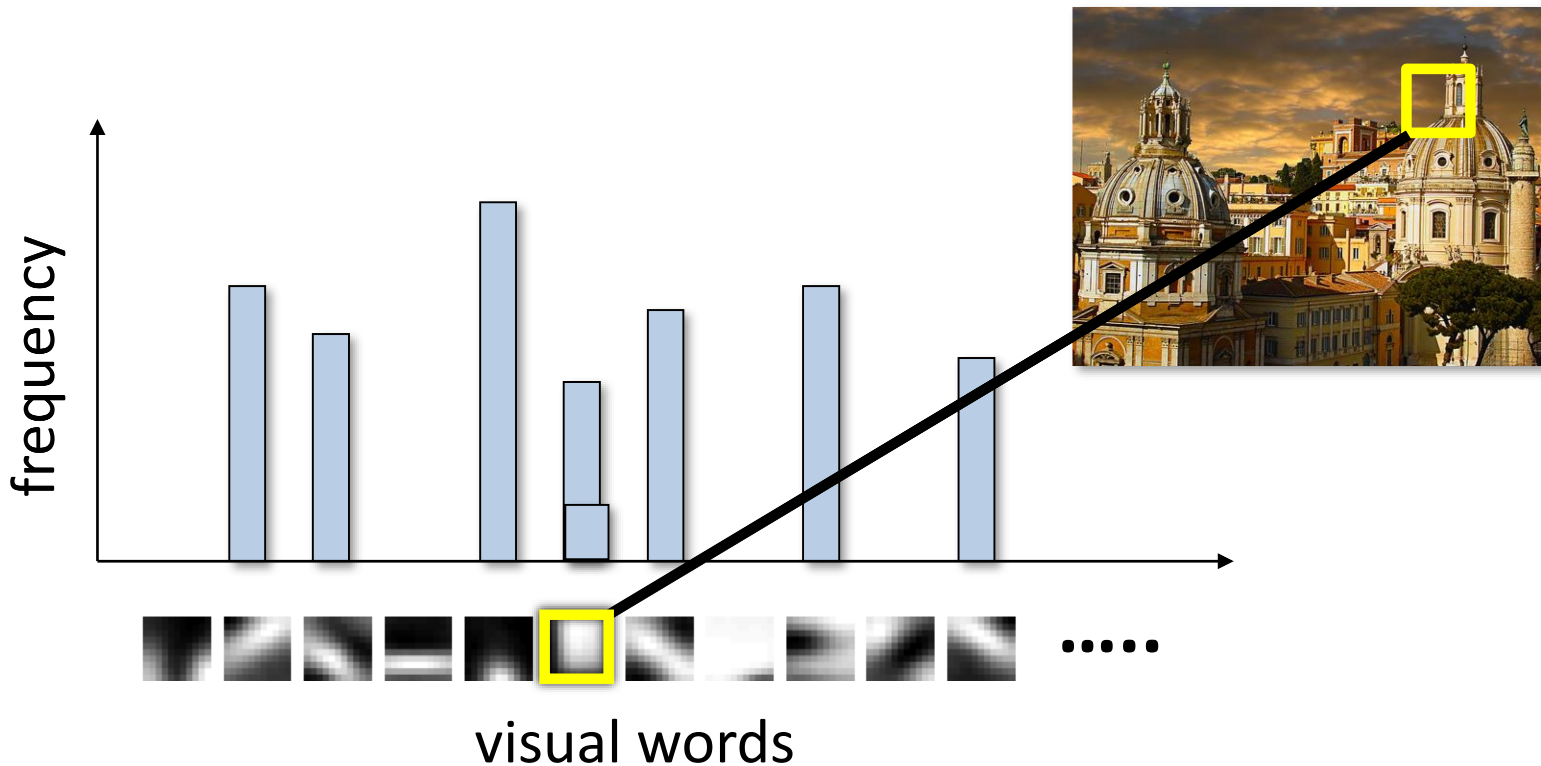
I_2

Topics

- Bag of Words
  - The BoW representation
  - TF-IDF weighting
  - Inverted file

Images as histograms of visual words

- Inspired by ideas from text retrieval
  - [Sivic and Zisserman, ICCV 2003]



TF (term frequency)-
IDF(inverse document frequency) weighting

- Instead of computing a regular histogram distance, we'll weight each word by it's *inverse document frequency*

- inverse document frequency (IDF) of word j =

$$\log \frac{\text{number of documents}}{\text{number of documents in which } j \text{ appears}}$$

TF-IDF weighting

- To compute the value of bin *j* in image *I*:

term frequency of *j* in *I* **x** *inverse document frequency* of *j*

Inverted file

- Each image has ~1,000 features
- We have ~1,000,000 visual words

  →each histogram is extremely sparse (mostly zeros)

- Inverted file
  - mapping from words to documents

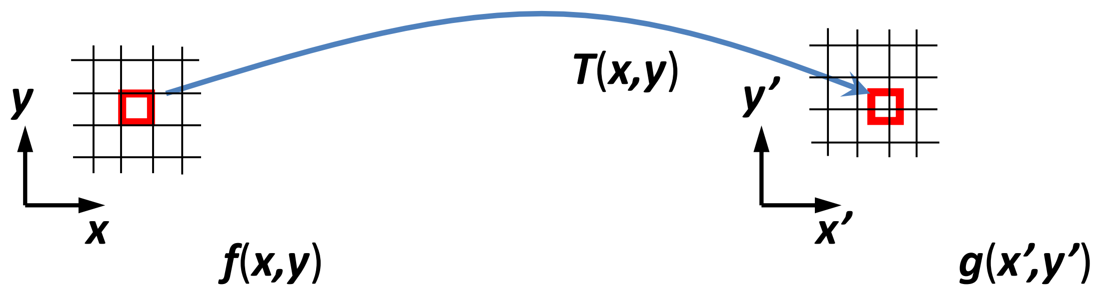
```
"a":      {2}
"banana": {2}
"is":     {0, 1, 2}
"it":     {0, 1, 2}
"what":   {0, 1}
```

Topics

- Transformation and Alignment
  - Image warping
  - All 2D Linear Transformations
  - Homogeneous coordinates
  - Affine Transformations
  - RANSAC

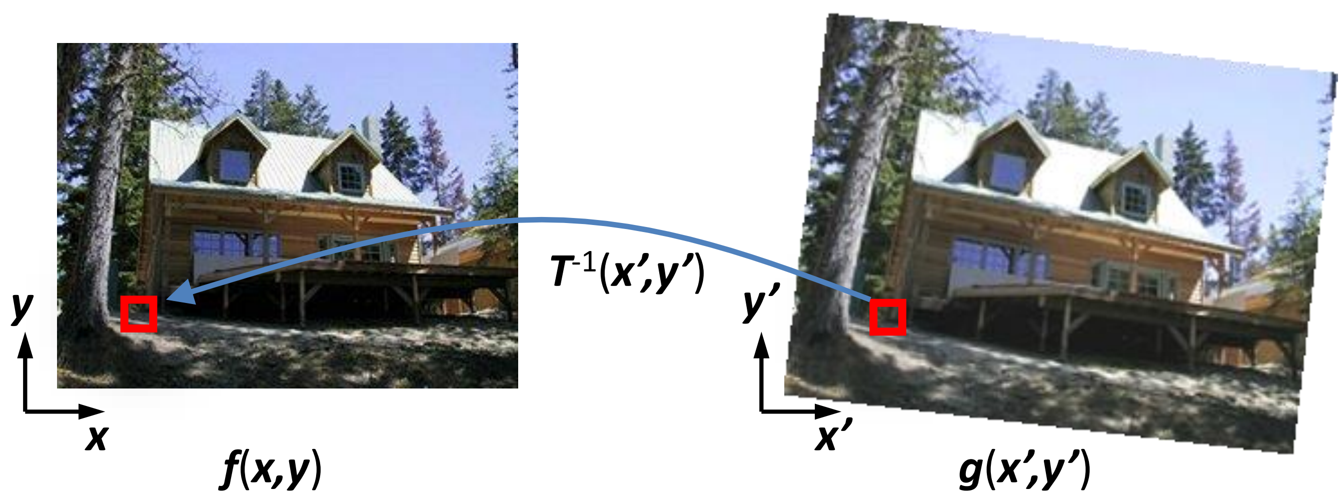
Forward Warping

- Send each pixel $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$ to its corresponding location $(\boldsymbol{x'},\boldsymbol{y'}) = \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y})$ in $\boldsymbol{g}(\boldsymbol{x'},\boldsymbol{y'})$
  - What if pixel lands "between" two pixels?
  - Answer: add "contribution" to several pixels, normalize later
  - Can still result in holes



Inverse Warping

- Get each pixel $\boldsymbol{g}(\boldsymbol{x}',\boldsymbol{y}')$ from its corresponding location $(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{T}^{-1}(\boldsymbol{x}',\boldsymbol{y}')$ in $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$
  - Requires taking the inverse of the transform
  - What if pixel comes from "between" two pixels?



All 2D Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

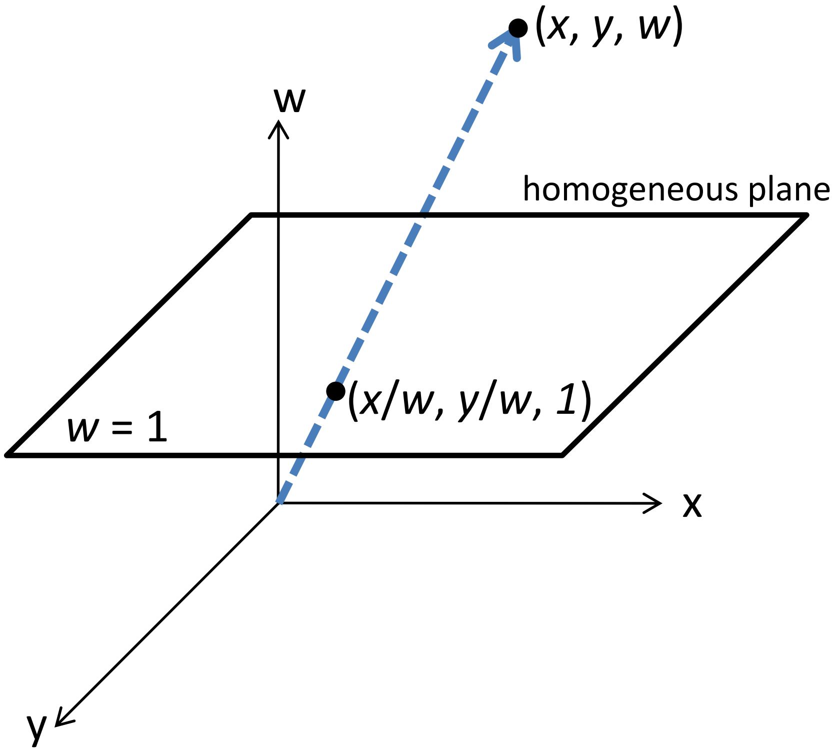
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Homogeneous coordinates

Trick: add one more coordinate:

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates



Converting *from* homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

Affine Transformations

- Affine transformations are combinations of …
  - Linear transformations, and
  - Translations

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

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

Translate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 & 0 \\ 0 & \boldsymbol{s}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Scale

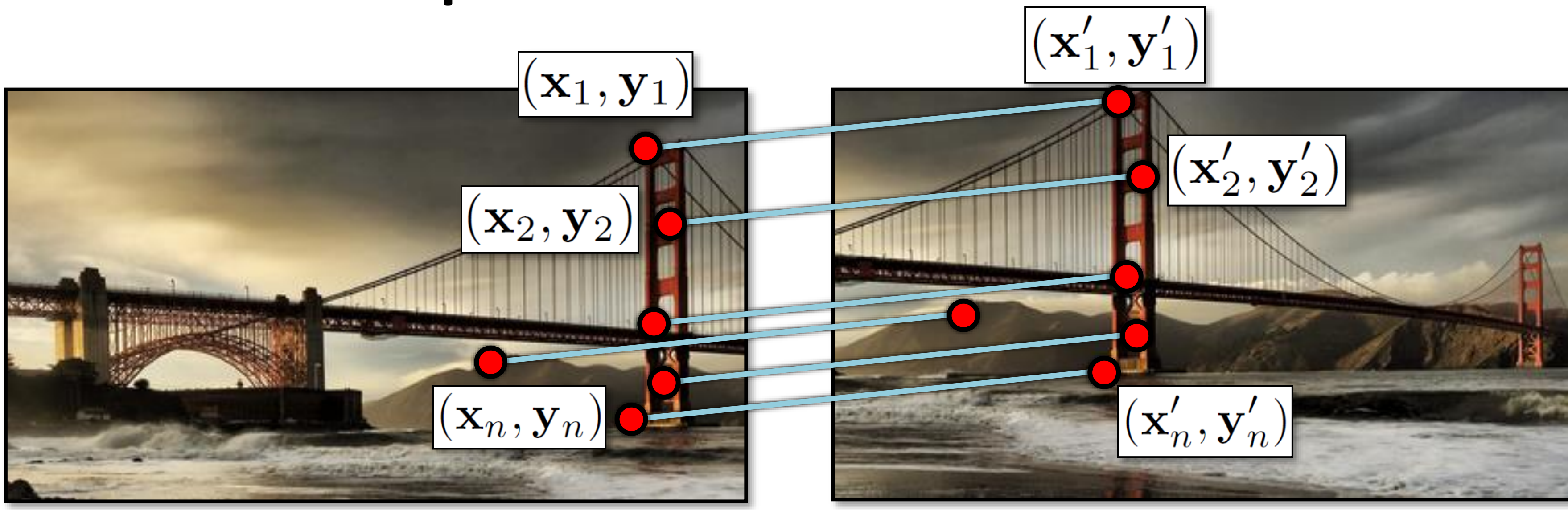
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D *in-plane* rotation

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh}_x & 0 \\ \boldsymbol{sh}_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Shear

Simple case: translations



Displacement of match *i* = $(\mathbf{x}'_i - \mathbf{x}_i, \mathbf{y}'_i - \mathbf{y}_i)$

$$(\mathbf{x}_t, \mathbf{y}_t) = \left(\frac{1}{n} \sum_{i=1}^{n} \mathbf{x}'_i - \mathbf{x}_i, \frac{1}{n} \sum_{i=1}^{n} \mathbf{y}'_i - \mathbf{y}_i \right)$$

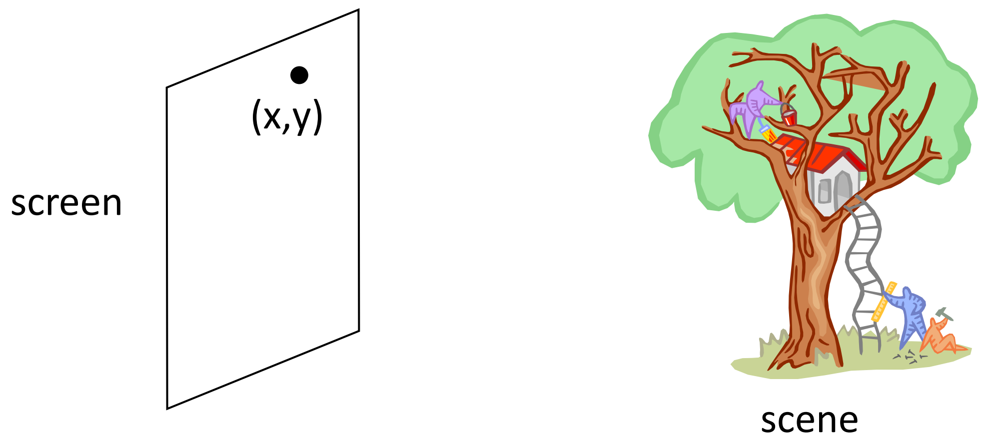
RANSAC

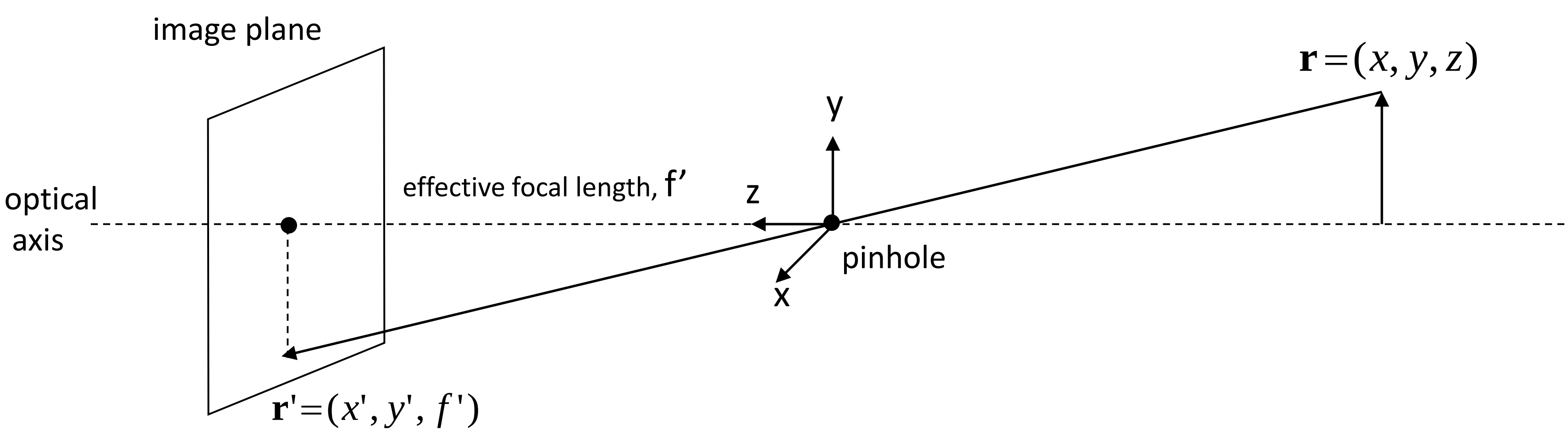
- General version:
  1. Randomly choose *s* samples
     - Typically *s* = minimum sample size that lets you fit a model
  2. Fit a model (e.g., line) to those samples
  3. Count the number of inliers that approximately fit the model
  4. Repeat *N* times
  5. Choose the model that has the largest set of inliers

Topics

- Cameras
  - Pinhole camera
  - Camera parameters
  - Modeling projection

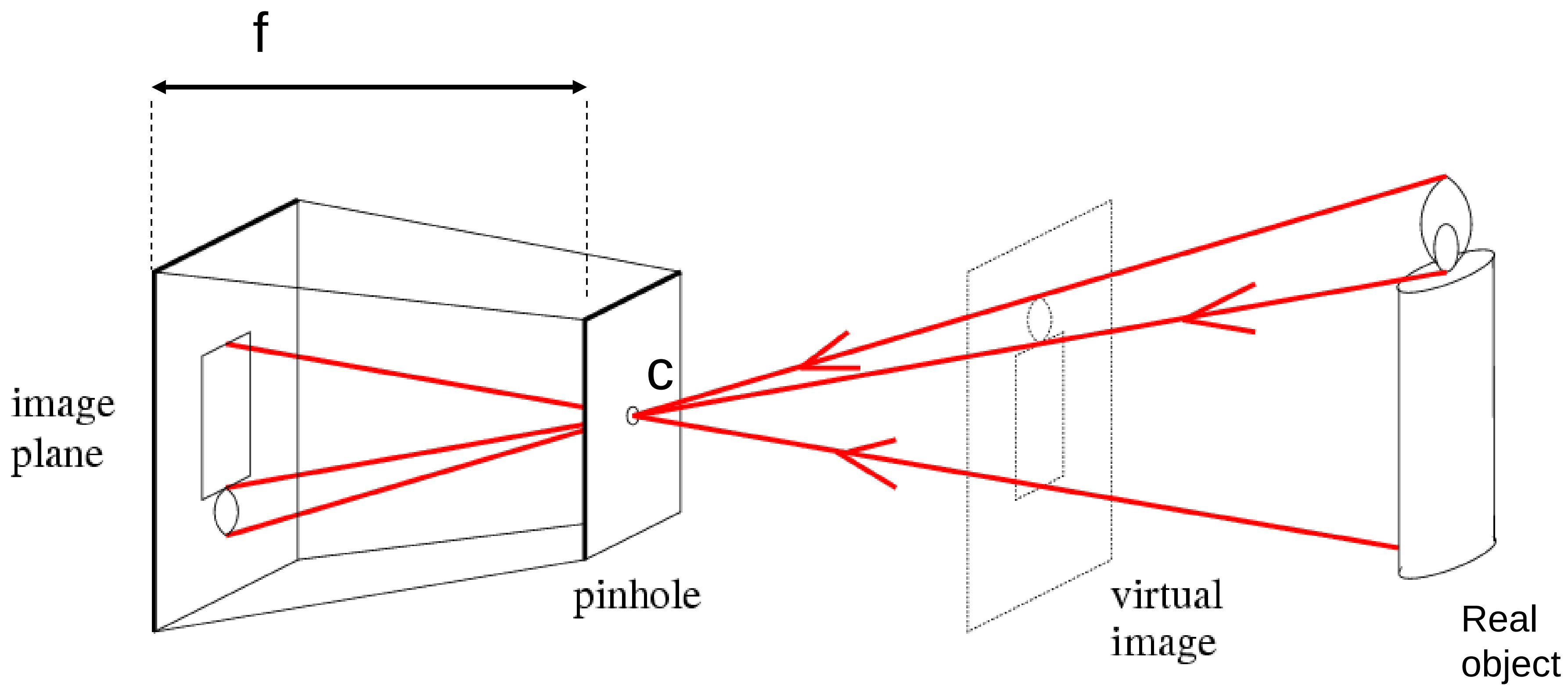
Pinhole and the Perspective Projection





$$\frac{\mathbf{r}'}{f'} = \frac{\mathbf{r}}{z} \quad \Rightarrow \quad \frac{x'}{f'} = \frac{x}{z} \qquad \frac{y'}{f'} = \frac{y}{z}$$

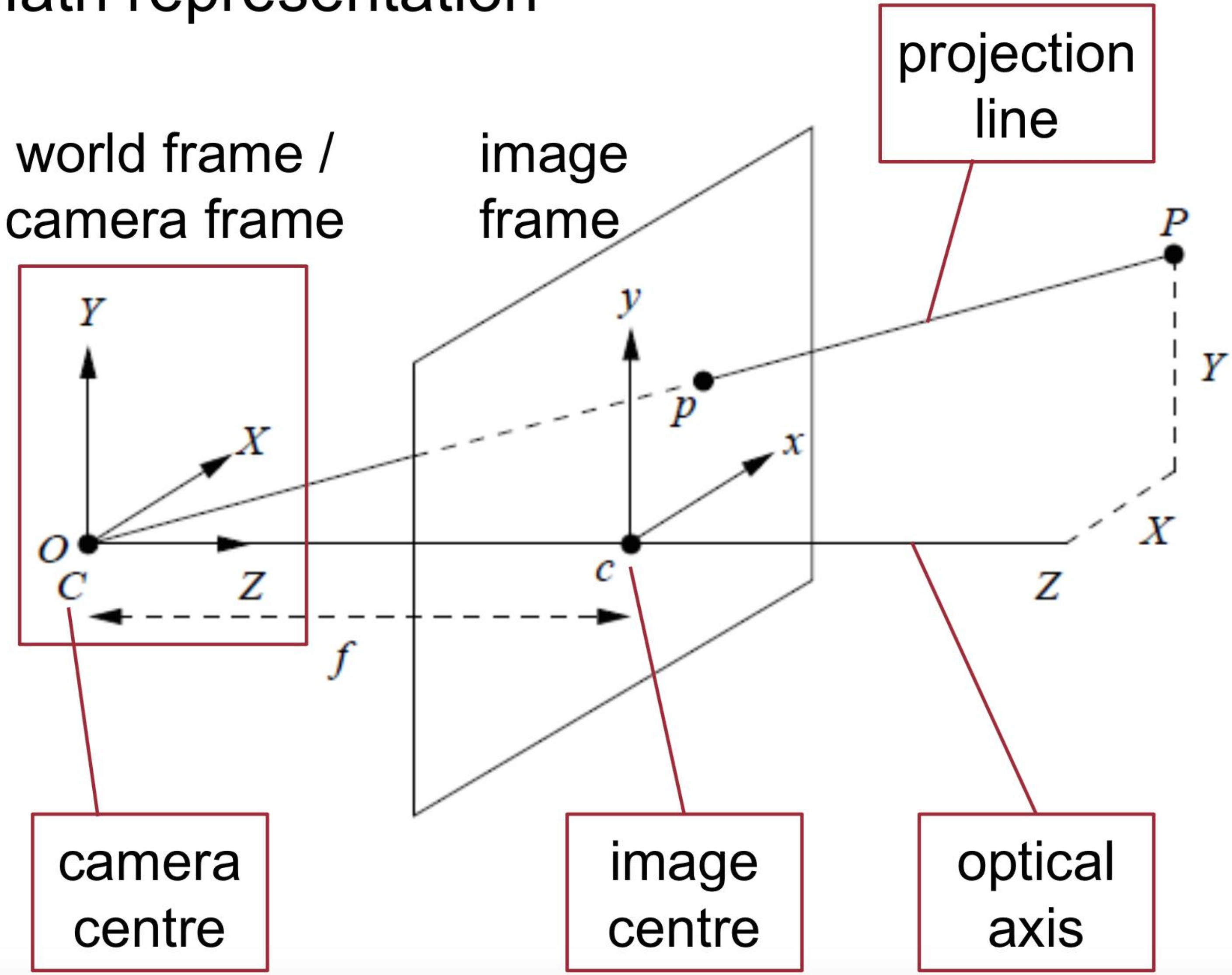
Pinhole camera model



f = Focal length
c = Optical center of the camera

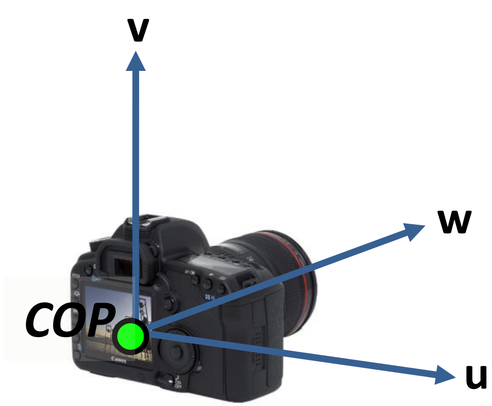
Figure from Forsyth

- Math representation



Camera parameters

- How can we model the geometry of a camera?



Camera

"The World"

Two important coordinate systems:

1. *World* coordinate system
2. *Camera* coordinate system

Camera parameters

- To project a point (x,y,z) in *world* coordinates into a camera
- First transform (x,y,z) into *camera* coordinates
- Need to know
  - Camera position (in world coordinates)
  - Camera orientation (in world coordinates)
- The formation of image frame
  - Need to know camera *intrinsics*

Intrinsic Parameters

- In the image frame, denote location of c $(principle\ point)$ image plane as c_x and c_y
- Image principle point:

Intersection between the camera optical axis and image plane

- Then

$$P' = (x', y') = (f\frac{x}{z} + c_x, f\frac{y}{z} + c_y)$$

Intrinsic Parameters

- Points in digital image are expressed as in pixels
- Points in image plane are represented in physical measurement (e.g., centimeter)
- The mapping between digital image and image plan can be something like $\frac{pixels}{cm}$
- We can use two parameters, *k* and *l*, to describe the mapping. If $k = l$, then the camera has “square pixels”.
- The equation now becomes:

$$P' = (x', y') = (fk\frac{x}{z} + c_x, fl\frac{y}{z} + c_y)$$

$$= (\alpha\frac{x}{z} + c_x, \beta\frac{y}{z} + c_y)$$

Intrinsic Parameters

$$P' = (x', y') = (\alpha \frac{x}{z} + c_x, \beta \frac{y}{z} + c_y)$$

In matrix form:

$$P' = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = MP$$

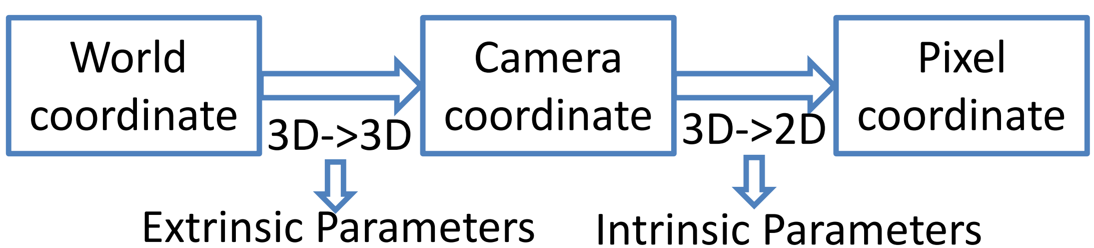
$$P' = MP = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix} [I \quad 0]P = K[I \quad 0]P$$

<u>*K: Camera matrix (or calibration matrix)*</u>

Extrinsic Parameters

- What if the information about the 3D world is available in a different coordinate system?
- We need to relate the points from world reference system to the camera reference system
- Given a point in world reference system P_w, the camera coordinate is computed as

$$P = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} P_w$$



Projection Matrix

- Combining intrinsic and extrinsic parameters, we have

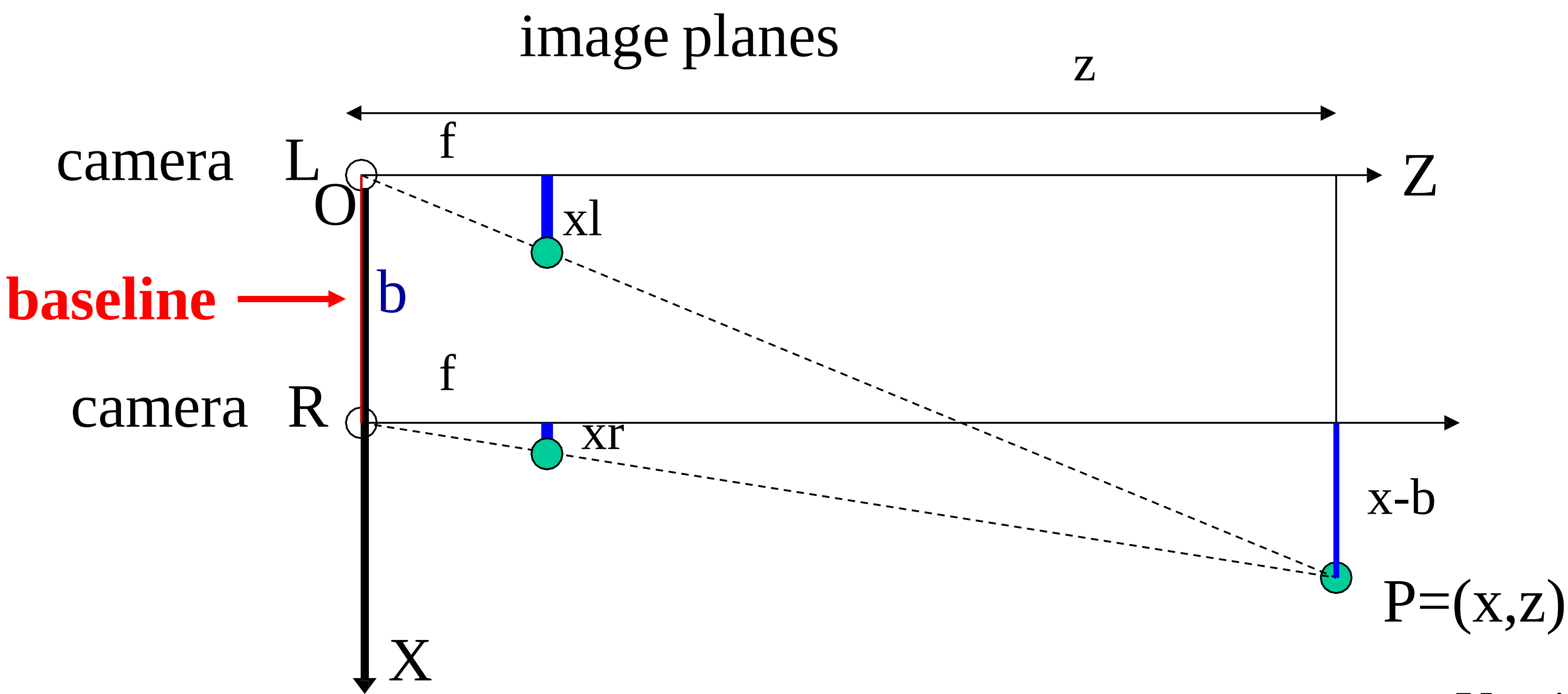
$$P' = \underbrace{K}_{\text{intrinsic parameters}} \overbrace{[R \quad T]}^{\text{extrinsic parameters}} P_w = M P_w$$

- K changes as the type of camera changes
- Extrinsic parameters are independent of camera

Topics

- Stereo Vision and Structure from Motion
  - Depth and Disparity
  - Epipolar geometry
  - Stereo matching
  - Structure from motion: problem definition

Optic axes of 2 cameras are parallel



$$\frac{z}{f} = \frac{x}{xl}$$

$$\frac{z}{f} = \frac{x-b}{xr}$$

$$\frac{z}{f} = \frac{y}{yl} = \frac{y}{yr}$$

Y-axis is perpendicular to the page.

(from similar triangles)

3D from Stereo Images

For stereo cameras with parallel optical axes, focal length f, baseline b, corresponding image points (xl,yl) and (xr,yr), the location of the 3D point can be derived from previous slide's equations:

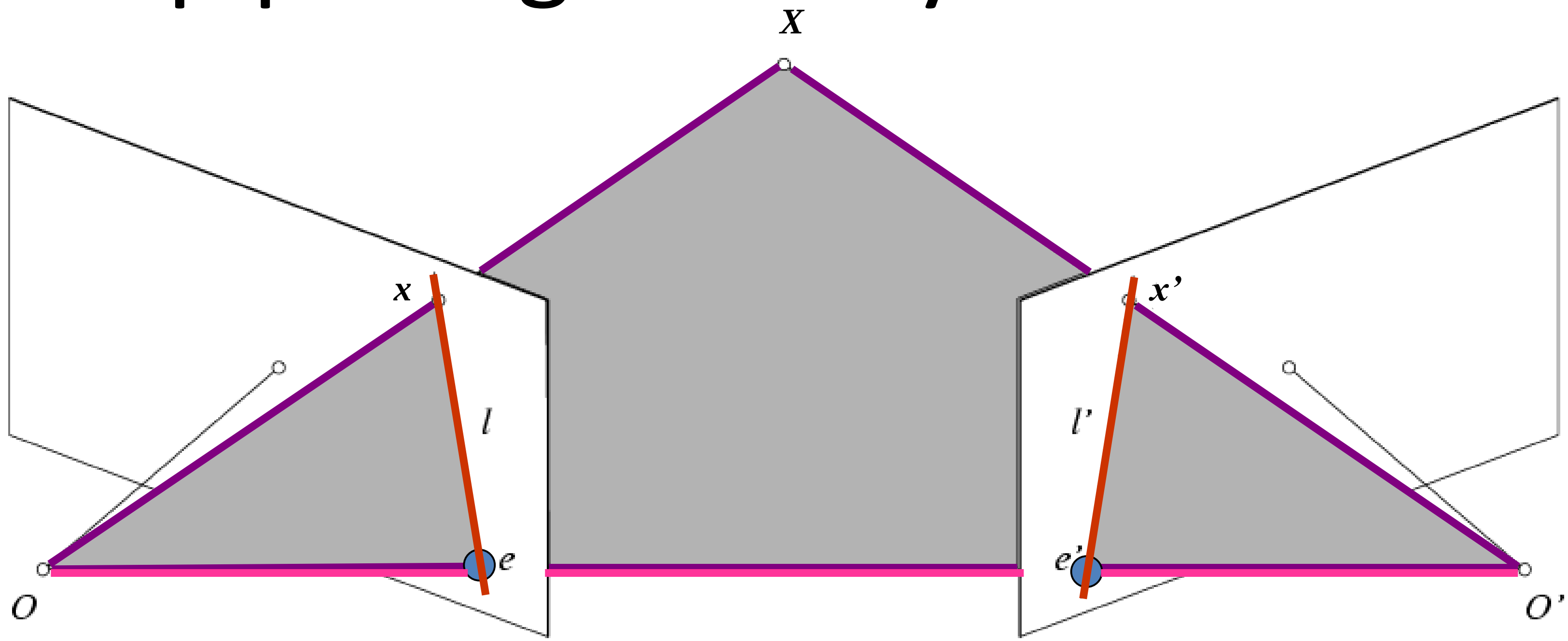
Depth z = f*b / (xl - xr) = f*b/d

x = xl*z/f or b + xr*z/f

y = yl*z/f or yr*z/f

Note that **depth is inversely proportional to disparity**

Epipolar geometry: notation

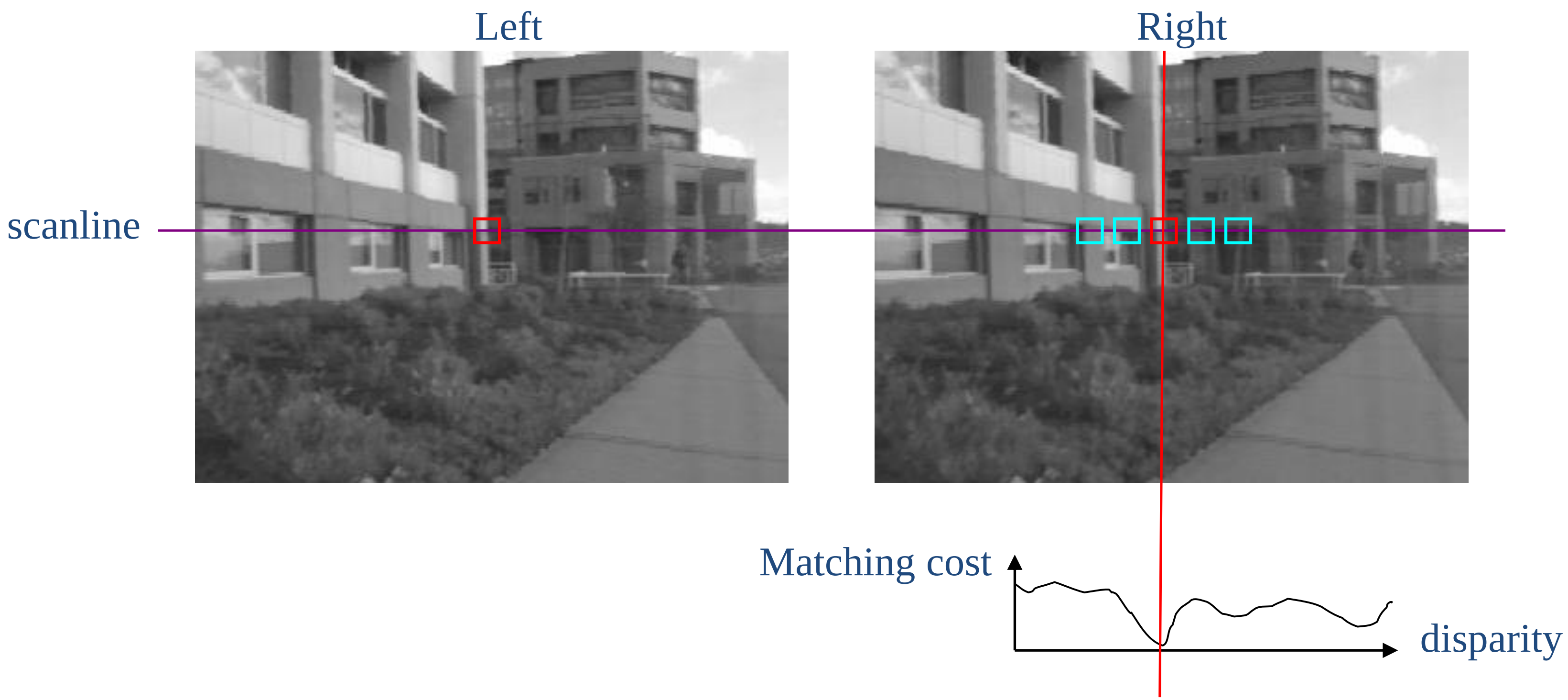


- **Baseline** – line connecting the two camera centers
- **Epipoles**

= intersections of baseline with image planes

= projections of the other camera center

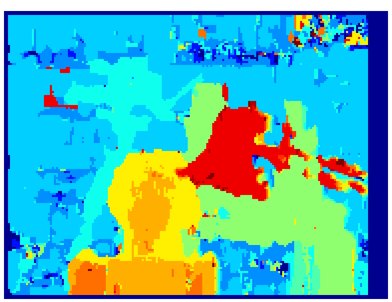
- **Epipolar Plane** – plane containing baseline (1D family)
- **Epipolar Lines -** intersections of epipolar plane with image planes (always come in corresponding pairs)



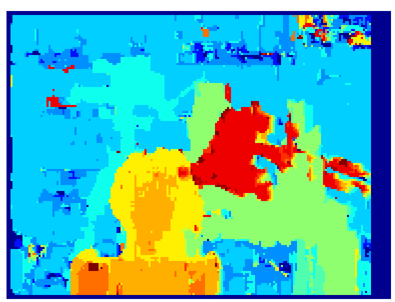
- Slide a window along the right scanline and compare contents of that window with the reference window in the left image
- Matching cost: SSD, SAD, or normalized cross correlation

Matching windows:

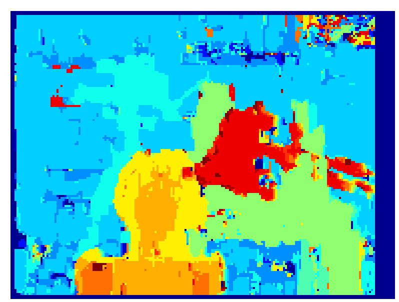
Similarity Measure	Formula
Sum of Absolute Differences (SAD)	$\sum_{(i,j)\in W} \lvert I_1(i,j) - I_2(x+i, y+j) \rvert$
Sum of Squared Differences (SSD)	$\sum_{(i,j)\in W} \left(I_1(i,j) - I_2(x+i, y+j)\right)^2$
Zero-mean SAD	$\sum_{(i,j)\in W} \lvert I_1(i,j) - \bar{I}_1(i,j) - I_2(x+i,y+j) + \bar{I}_2(x+i,y+j) \rvert$
Normalized Cross Correlation (NCC)	$\dfrac{\sum_{(i,j)\in W} I_1(i,j) . I_2(x+i, y+j)}{\sqrt[2]{\sum_{(i,j)\in W} I_1^2(i,j) . \sum_{(i,j)\in W} I_2^2(x+i, y+j)}}$



SAD



SSD

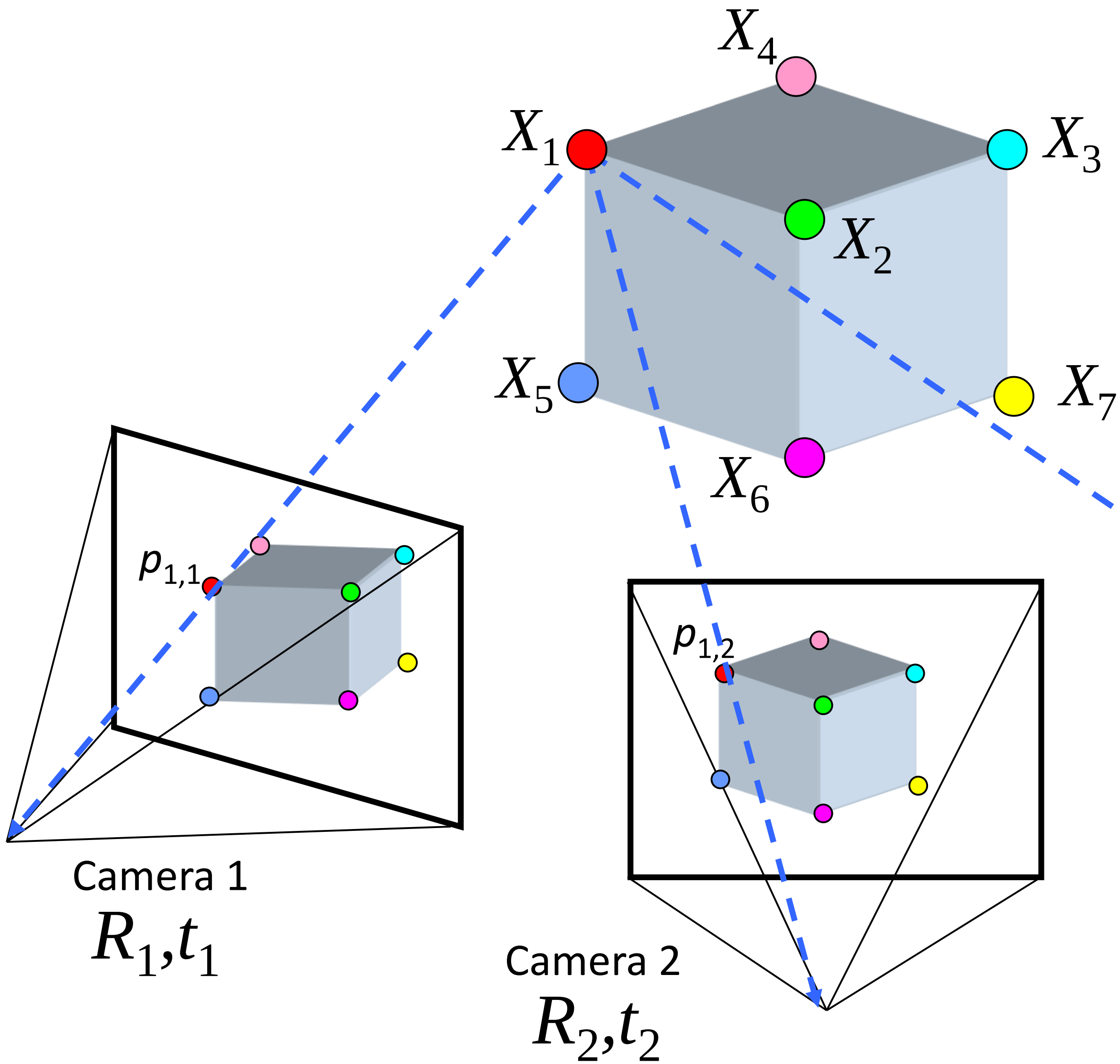


NCC

Ground truth

http://siddhantahuja.wordpress.com/category/stereo-vision/

Structure from motion



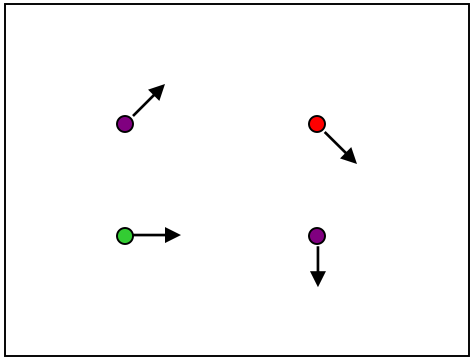
minimize

$$g(\mathbf{R}, \mathbf{T}, \mathbf{X})$$

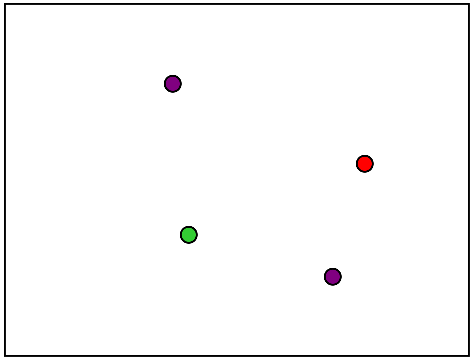
Topics

- Optical flow
  - Assumptions in Lucas-Kanade method
  - Lucas-Kanade Algorithm (Brightness Constancy Equation)

Optical Flow



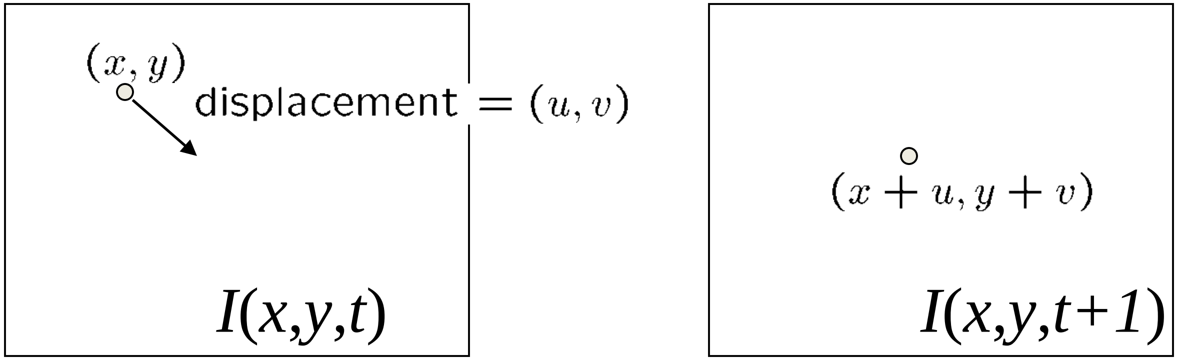
$I(x,y,t)$



$I(x,y,t+1)$

- Given two subsequent frames, estimate the point translation
- Key assumptions of Lucas-Kanade Tracker
  - **Brightness constancy:** projection of the same point looks the same in every frame
  - **Small motion:** points do not move very far
  - **Spatial coherence:** points move like their neighbors

The brightness constancy constraint



- Brightness Constancy Equation:

$$I(x,y,t) = I(x+u, y+v, t+1)$$

Take Taylor expansion of *I(x+u, y+v, t+1)* at *(x,y,t)* to linearize the right side:

Image derivative along x Difference over frames

$$I(x+u, y+v, t+1) \approx I(x,y,t) + \boxed{I_x} \cdot u + I_y \cdot v + \boxed{I_t}$$

$$I(x+u, y+v, t+1) - I(x,y,t) = +I_x \cdot u + I_y \cdot v + I_t$$

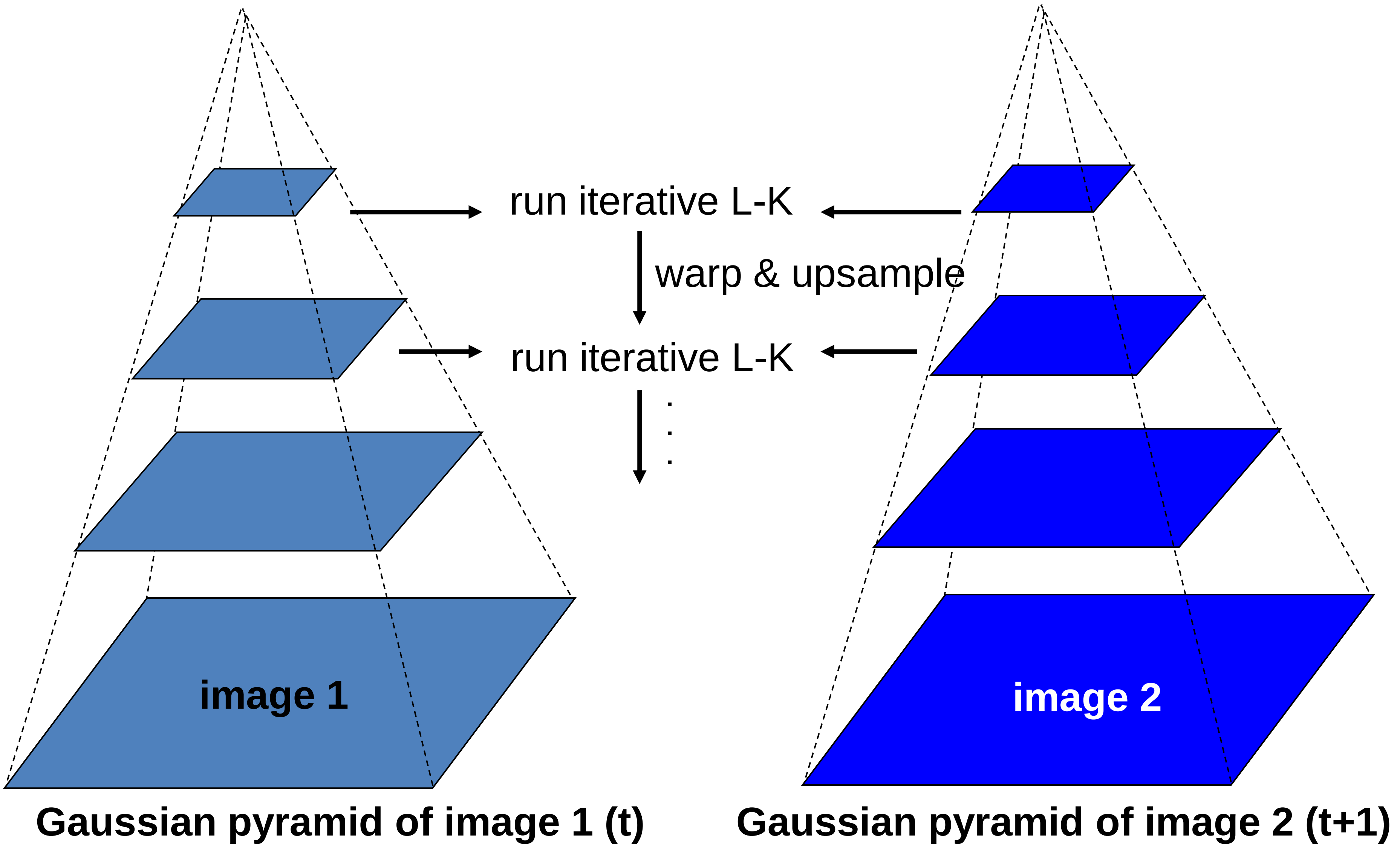
So: $$I_x \cdot u + I_y \cdot v + I_t \approx 0 \quad \rightarrow \nabla \mathrm{I} \cdot [\mathrm{u} \ \ \mathrm{v}]^{\mathrm{T}} + \mathrm{I_t} = 0$$

Iterative Refinement

- Iterative Lucas-Kanade Algorithm
  1. Estimate velocity at each pixel by solving Lucas-Kanade equations
  2. Warp I(t-1) towards I(t) using the estimated flow field

     - use image warping techniques
  3. Repeat until convergence

Coarse-to-fine optical flow estimation



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.
  - Return 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.