Visual Recognition: Transformations and Features

Raquel Urtasun

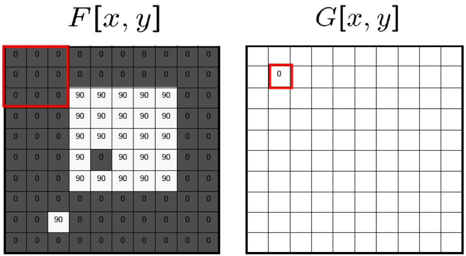
TTI Chicago

Jan 17, 2012

What did we see in class last week?

Last week

- Image formation.
- Filtering: convolution vs correlation



- Separable filters.
- Computing edges.
- Steerable filters.
- Midwest Vision Workshop.

First Recognition System: Template matching

- What if the template is not identical to some subimage in the scene?
- Match can be meaningful, if scale, orientation, and general appearance is right.
- How can I find the right scale?

Scene

Template

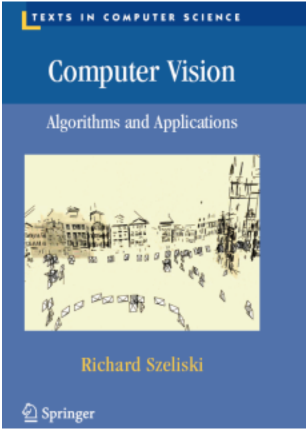
[Source: K. Grauman]

Today's lecture ...

- Additional transformations
- Local features: Interest point detection and descriptors

Material

- Chapter 3 and 4 of Rich Szeliski book



- Available online here

Other transformations

Integral Images

- If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.
- It is the running sum of all the pixel values from the origin

$$s(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k,l)$$

Integral Images

- If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.
- It is the running sum of all the pixel values from the origin

$$s(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k,l)$$

- This can be efficiently computed using a recursive (raster-scan) algorithm

$$s(i,j) = s(i-1,j) + s(i,j-1) - s(i-1,j-1) + f(i,j)$$

Integral Images

- If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.
- It is the running sum of all the pixel values from the origin

$$s(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k,l)$$

- This can be efficiently computed using a recursive (raster-scan) algorithm

$$s(i,j) = s(i-1,j) + s(i,j-1) - s(i-1,j-1) + f(i,j)$$

- The image $s(i,j)$ is called an **integral image** and can actually be computed using only two additions per pixel if separate row sums are used.

Integral Images

- If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.
- It is the running sum of all the pixel values from the origin

$$s(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k,l)$$

- This can be efficiently computed using a recursive (raster-scan) algorithm

$$s(i,j) = s(i-1,j) + s(i,j-1) - s(i-1,j-1) + f(i,j)$$

- The image $s(i,j)$ is called an **integral image** and can actually be computed using only two additions per pixel if separate row sums are used.
- To find the summed area (integral) inside a rectangle $[i_0, i_1] \times [j_0, j_1]$ we simply combine four samples from the summed area table.

$$S([i_0, i_1] \times [j_0, j_1]) = s(i_1, j_1) - s(i_1, j_0 - 1) - s(i_0 - 1, j_1) + s(i_0 - 1, j_0 - 1)$$

Integral Images

- If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.
- It is the running sum of all the pixel values from the origin

$$s(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k,l)$$

- This can be efficiently computed using a recursive (raster-scan) algorithm

$$s(i,j) = s(i-1,j) + s(i,j-1) - s(i-1,j-1) + f(i,j)$$

- The image $s(i,j)$ is called an **integral image** and can actually be computed using only two additions per pixel if separate row sums are used.
- To find the summed area (integral) inside a rectangle $[i_0, i_1] \times [j_0, j_1]$ we simply combine four samples from the summed area table.

$$S([i_0,i_1] \times [j_0,j_1]) = s(i_1,j_1) - s(i_1,j_0-1) - s(i_0-1,j_1) + s(i_0-1,j_0-1)$$

- Summed area tables have been used in face detection [Viola & Jones, 04]

Integral Images

- If an image is going to be repeatedly convolved with different box filters, it is useful to compute the **summed area table**.
- It is the running sum of all the pixel values from the origin

$$s(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k,l)$$

- This can be efficiently computed using a recursive (raster-scan) algorithm

$$s(i,j) = s(i-1,j) + s(i,j-1) - s(i-1,j-1) + f(i,j)$$

- The image $s(i,j)$ is called an **integral image** and can actually be computed using only two additions per pixel if separate row sums are used.
- To find the summed area (integral) inside a rectangle $[i_0, i_1] \times [j_0, j_1]$ we simply combine four samples from the summed area table.

$$S([i_0,i_1] \times [j_0,j_1]) = s(i_1,j_1) - s(i_1,j_0-1) - s(i_0-1,j_1) + s(i_0-1,j_0-1)$$

- Summed area tables have been used in face detection [Viola & Jones, 04]

Example of Integral Images

3	2	7	2	3
1	5	1	3	4
5	1	**3**	5	1
4	3	2	1	6
2	4	1	4	8

(a) S = 24

3	5	12	14	17
4	*11*	**19**	24	31
9	**17**	28	38	46
13	24	37	48	62
15	30	44	59	81

(b) s = 28

3	5	12	*14*	17
4	11	19	24	31
9	17	28	38	46
13	24	37	**48**	62
15	30	44	59	81

(c) S = 24

Figure 3.17 Summed area tables: (a) original image; (b) summed area table; (c) computation of area sum. Each value in the summed area table $s(i,j)$ (red) is computed recursively from its three adjacent (blue) neighbors (3.31). Area sums S (green) are computed by combining the four values at the rectangle corners (purple) (3.32). Positive values are shown in **bold** and negative values in *italics*.

Non-linear filters: Median filter

- We have seen linear filters, i.e., their response to a sum of two signals is the same as the sum of the individual responses.
- **Median filter**: Non linear filter that selects the median value from each pixels neighborhood.

Non-linear filters: Median filter

- We have seen linear filters, i.e., their response to a sum of two signals is the same as the sum of the individual responses.
- **Median filter**: Non linear filter that selects the median value from each pixels neighborhood.
- Robust to outliers, but not good for Gaussian noise.

Non-linear filters: Median filter

- We have seen linear filters, i.e., their response to a sum of two signals is the same as the sum of the individual responses.
- **Median filter**: Non linear filter that selects the median value from each pixels neighborhood.
- Robust to outliers, but not good for Gaussian noise.
- α-**trimmed mean**: averages together all of the pixels except for the α fraction that are the smallest and the largest.

Non-linear filters: Median filter

- We have seen linear filters, i.e., their response to a sum of two signals is the same as the sum of the individual responses.
- **Median filter**: Non linear filter that selects the median value from each pixels neighborhood.
- Robust to outliers, but not good for Gaussian noise.
- α-**trimmed mean**: averages together all of the pixels except for the α fraction that are the smallest and the largest.

Example of non-linear filters

1	2	1	2	4
2	1	3	5	8
1	3	7	6	9
3	4	8	6	7
4	5	7	8	9

(Median filter)

1	2	1	2	4
2	1	3	5	8
1	3	7	6	9
3	4	8	6	7
4	5	7	8	9

(α-trimmed mean)

Bilateral Filtering

- Weighted filter kernel with a better outlier rejection.
- Instead of rejecting a fixed percentage, we reject (in a soft way) pixels whose values differ too much from the central pixel value.

Bilateral Filtering

- Weighted filter kernel with a better outlier rejection.
- Instead of rejecting a fixed percentage, we reject (in a soft way) pixels whose values differ too much from the central pixel value.
- The output pixel value depends on a weighted combination of neighboring pixel values

$$g(i,j) = \frac{\sum_{k,l} f(k,l) w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}$$

Bilateral Filtering

- Weighted filter kernel with a better outlier rejection.
- Instead of rejecting a fixed percentage, we reject (in a soft way) pixels whose values differ too much from the central pixel value.
- The output pixel value depends on a weighted combination of neighboring pixel values

$$g(i,j) = \frac{\sum_{k,l} f(k,l) w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}$$

- Data-dependent bilateral weight function

$$w(i,j,k,l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{||f(i,j) - f(k,l)||^2}{2\sigma_r^2}\right)$$

composed of the **domain kernel** and the **range kernel**.

Bilateral Filtering

- Weighted filter kernel with a better outlier rejection.
- Instead of rejecting a fixed percentage, we reject (in a soft way) pixels whose values differ too much from the central pixel value.
- The output pixel value depends on a weighted combination of neighboring pixel values

$$g(i,j) = \frac{\sum_{k,l} f(k,l) w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}$$

- Data-dependent bilateral weight function

$$w(i,j,k,l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{||f(i,j) - f(k,l)||^2}{2\sigma_r^2}\right)$$

composed of the **domain kernel** and the **range kernel**.

Example Bilateral Filtering

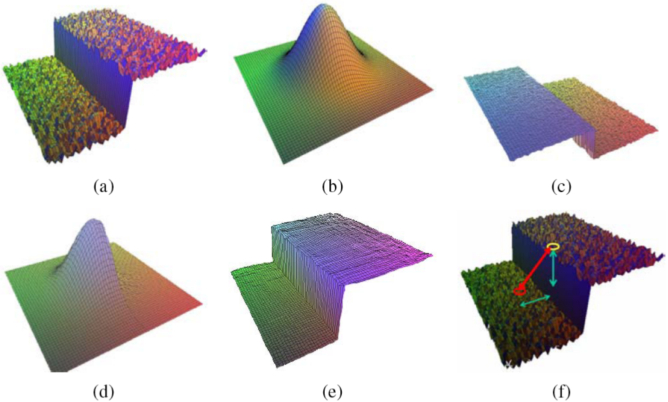


Figure: Bilateral filtering [Durand & Dorsey, 02]. (a) noisy step edge input. (b) domain filter (Gaussian). (c) range filter (similarity to center pixel value). (d) bilateral filter. (e) filtered step edge output. (f) 3D distance between pixels

[Source: R. Szeliski]

Distance Transform

- Useful to quickly precomputing the distance to a curve or a set of points.
- Let $d(k, l)$ be some distance metric between pixel offsets, e.g., Manhattan distance
$$d(k, l) = |k| + |l|$$
or Euclidean distance
$$d(k, l) = \sqrt{k^2 + l^2}$$

Distance Transform

- Useful to quickly precomputing the distance to a curve or a set of points.
- Let $d(k, l)$ be some distance metric between pixel offsets, e.g., Manhattan distance

$$d(k, l) = |k| + |l|$$

  or Euclidean distance

$$d(k, l) = \sqrt{k^2 + l^2}$$

- The distance transform $D(i, j)$ of a binary image $b(i, j)$ is defined as

$$D(i, j) = \min_{k,l;b(k,l)=0} d(i - k, j - l)$$

  it is the distance to the nearest pixel whose value is 0.

Distance Transform

- Useful to quickly precomputing the distance to a curve or a set of points.
- Let $d(k, l)$ be some distance metric between pixel offsets, e.g., Manhattan distance

$$d(k, l) = |k| + |l|$$

  or Euclidean distance

$$d(k, l) = \sqrt{k^2 + l^2}$$

- The distance transform $D(i, j)$ of a binary image $b(i, j)$ is defined as

$$D(i, j) = \min_{k,l;b(k,l)=0} d(i - k, j - l)$$

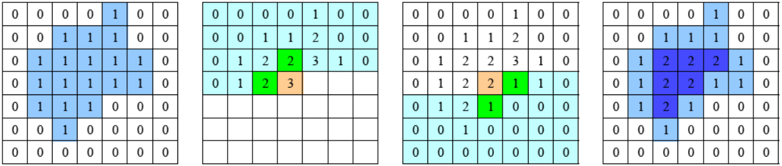
  it is the distance to the nearest pixel whose value is 0.

Distance Transform Algorithm

- The Manhattan distance can be computed using a forward and backward pass of a simple raster-scan algorithm.
- Forward pass:, each non-zero pixel in b is replaced by the minimum of 1 + the distance of its north or west neighbor.

Distance Transform Algorithm

- The Manhattan distance can be computed using a forward and backward pass of a simple raster-scan algorithm.
- Forward pass:, each non-zero pixel in b is replaced by the minimum of 1 + the distance of its north or west neighbor.
- Backward pass: the same, but the minimum is both over the current value D and 1 + the distance of the south and east neighbors.



(a) (b) (c) (d)

Figure: City block distance transform: (a) original binary image; (b) top to bottom (forward) raster sweep: green values are used to compute the orange value; (c) bottom to top (backward) raster sweep: green values are merged with old orange value; (d) final distance transform.

[Source: R. Szeliski]

Distance Transform Algorithm

- The Manhattan distance can be computed using a forward and backward pass of a simple raster-scan algorithm.
- Forward pass:, each non-zero pixel in b is replaced by the minimum of 1 + the distance of its north or west neighbor.
- Backward pass: the same, but the minimum is both over the current value D and 1 + the distance of the south and east neighbors.

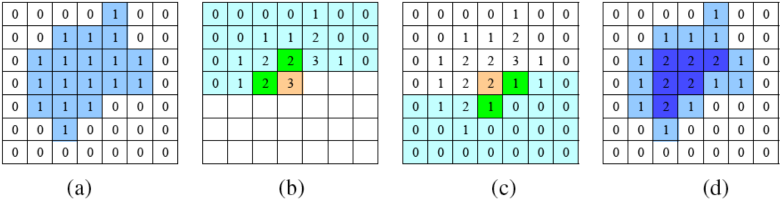
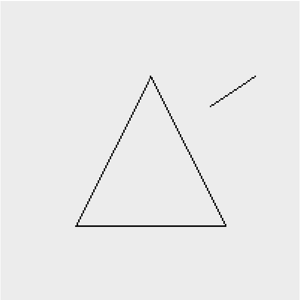


Figure: City block distance transform: (a) original binary image; (b) top to bottom (forward) raster sweep: green values are used to compute the orange value; (c) bottom to top (backward) raster sweep: green values are merged with old orange value; (d) final distance transform.

[Source: R. Szeliski]

Example of Distance Transform

- More complicated in the Euclidean case.
- Example of a distance transform



- The ridges is the **skeleton** or **medial axis**.
- Extension: Signed distance transform.

[Source: P. Felzenszwalb]

Fourier Transform

- Fourier analysis could be used to analyze the frequency characteristics of various filters.
- How can we analyze what a given filter does to high, medium, and low frequencies?

Fourier Transform

- Fourier analysis could be used to analyze the frequency characteristics of various filters.
- How can we analyze what a given filter does to high, medium, and low frequencies?
- Pass a sinusoid of known frequency through the filter and to observe by how much it is attenuated

$$s(x) = \sin(2\pi fx + \phi_i) = \sin(\omega x + \phi_i)$$

with frequency f, angular frequency ω and phase ϕ_i.

Fourier Transform

- Fourier analysis could be used to analyze the frequency characteristics of various filters.
- How can we analyze what a given filter does to high, medium, and low frequencies?
- Pass a sinusoid of known frequency through the filter and to observe by how much it is attenuated

$$s(x) = \sin(2\pi fx + \phi_i) = \sin(\omega x + \phi_i)$$

  with frequency f, angular frequency ω and phase ϕ_i.

- If we convolve the sinusoidal signal $s(x)$ with a filter whose impulse response is $h(x)$, we get another sinusoid of the same frequency but different magnitude and phase

$$o(x) = h(x) * s(x) = A\sin(\omega x + \phi_o)$$

Fourier Transform

- Fourier analysis could be used to analyze the frequency characteristics of various filters.
- How can we analyze what a given filter does to high, medium, and low frequencies?
- Pass a sinusoid of known frequency through the filter and to observe by how much it is attenuated

$$s(x) = \sin(2\pi f x + \phi_i) = \sin(\omega x + \phi_i)$$

  with frequency f, angular frequency ω and phase ϕ_i.
- If we convolve the sinusoidal signal $s(x)$ with a filter whose impulse response is $h(x)$, we get another sinusoid of the same frequency but different magnitude and phase

$$o(x) = h(x) * s(x) = A \sin(\omega x + \phi_o)$$

Filtering and Fourier

- Convolution can be expressed as a weighted summation of shifted input signals (sinusoids); so it is just a single sinusoid at that frequency.

$$o(x) = h(x) * s(x) = A \sin(\omega x + \phi_o)$$

A is the **gain** or **magnitude** of the filter, while the phase difference $\Delta\phi = \phi_o - \phi_i$i is the **shift** or **phase**

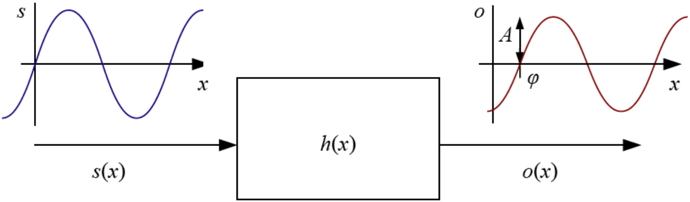


Figure 3.24 The Fourier Transform as the response of a filter $h(x)$ to an input sinusoid $s(x) = e^{j\omega x}$ yielding an output sinusoid $o(x) = h(x) * s(x) = Ae^{j\omega x+\phi}$.

Complex notation

- The sinusoid is express as $s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x$ and the filter sinusoid as

$$o(x) = h(x) * s(x) = Ae^{j\omega x + \phi}$$

- The Fourier transform pair is

$$h(x) \longleftrightarrow H(\omega)$$

Complex notation

- The sinusoid is express as $s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x$ and the filter sinusoid as
$$o(x) = h(x) * s(x) = Ae^{j\omega x + \phi}$$
- The Fourier transform pair is
$$h(x) \longleftrightarrow H(\omega)$$
- The Fourier transform in continuous domain
$$H(\omega) = \int_{-\infty}^{\infty} h(x) e^{-j\omega x} dx$$

Complex notation

- The sinusoid is express as $s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x$ and the filter sinusoid as
$$o(x) = h(x) * s(x) = Ae^{j\omega x + \phi}$$
- The Fourier transform pair is
$$h(x) \longleftrightarrow H(\omega)$$
- The Fourier transform in continuous domain
$$H(\omega) = \int_{-\infty}^{\infty} h(x) e^{-j\omega x} dx$$
- The Fourier transform in discrete domain
$$H(k) = \frac{1}{N} \sum_{x=0}^{N-1} h(x) e^{-j\frac{2\pi kx}{N}}$$
where N is the length of the signal.

Complex notation

- The sinusoid is express as $s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x$ and the filter sinusoid as
$$o(x) = h(x) * s(x) = Ae^{j\omega x + \phi}$$
- The Fourier transform pair is
$$h(x) \longleftrightarrow H(\omega)$$
- The Fourier transform in continuous domain
$$H(\omega) = \int_{-\infty}^{\infty} h(x) e^{-j\omega x} dx$$
- The Fourier transform in discrete domain
$$H(k) = \frac{1}{N} \sum_{x=0}^{N-1} h(x) e^{-j \frac{2\pi kx}{N}}$$
where N is the length of the signal.
- The discrete form is known as the Discrete Fourier Transform (DFT).

Complex notation

- The sinusoid is express as $s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x$ and the filter sinusoid as

$$o(x) = h(x) * s(x) = Ae^{j\omega x + \phi}$$

- The Fourier transform pair is

$$h(x) \longleftrightarrow H(\omega)$$

- The Fourier transform in continuous domain

$$H(\omega) = \int_{-\infty}^{\infty} h(x) e^{-j\omega x} dx$$

- The Fourier transform in discrete domain

$$H(k) = \frac{1}{N} \sum_{x=0}^{N-1} h(x) e^{-j \frac{2\pi kx}{N}}$$

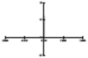
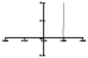
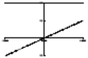
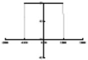
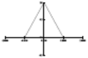
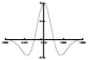
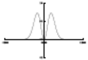
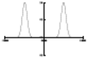
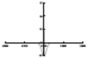
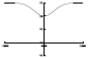
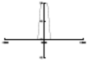
  where N is the length of the signal.

- The discrete form is known as the Discrete Fourier Transform (DFT).

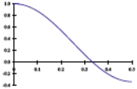
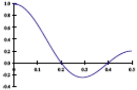
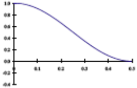
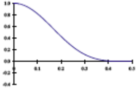
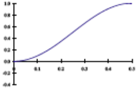
Properties Fourier Transform

Property	Signal		Transform
superposition	$f_1(x) + f_2(x)$		$F_1(\omega) + F_2(\omega)$
shift	$f(x - x_0)$		$F(\omega)e^{-j\omega x_0}$
reversal	$f(-x)$		$F^*(\omega)$
convolution	$f(x) * h(x)$		$F(\omega)H(\omega)$
correlation	$f(x) \otimes h(x)$		$F(\omega)H^*(\omega)$
multiplication	$f(x)h(x)$		$F(\omega) * H(\omega)$
differentiation	$f'(x)$		$j\omega F(\omega)$
domain scaling	$f(ax)$		$1/aF(\omega/a)$
real images	$f(x) = f^*(x)$	$\Leftrightarrow$	$F(\omega) = F(-\omega)$
Parseval's Theorem	$\sum_x [f(x)]^2$	$=$	$\sum_\omega [F(\omega)]^2$

[Source: R. Szeliski]

Name	Signal			Transform	
impulse		$\delta(x)$	$\Leftrightarrow$	1	
shifted impulse		$\delta(x-u)$	$\Leftrightarrow$	$e^{-j\omega u}$	
box filter		$\mathrm{box}(x/a)$	$\Leftrightarrow$	$a\,\mathrm{sinc}(a\omega)$	
tent		$\mathrm{tent}(x/a)$	$\Leftrightarrow$	$a\,\mathrm{sinc}^2(a\omega)$	
Gaussian		$G(x;\sigma)$	$\Leftrightarrow$	$\frac{\sqrt{2\pi}}{\sigma}G(\omega;\sigma^{-1})$	
Laplacian of Gaussian		$(\frac{x^2}{\sigma^4}-\frac{1}{\sigma^2})G(x;\sigma)$	$\Leftrightarrow$	$-\frac{\sqrt{2\pi}}{\sigma}\omega^2 G(\omega;\sigma^{-1})$	
Gabor		$\cos(\omega_0 x)G(x;\sigma)$	$\Leftrightarrow$	$\frac{\sqrt{2\pi}}{\sigma}G(\omega \pm \omega_0;\sigma^{-1})$	
unsharp mask		$(1+\gamma)\delta(x)$ $-\gamma G(x;\sigma)$	$\Leftrightarrow$	$(1+\gamma)-$ $\frac{\sqrt{2\pi}\gamma}{\sigma}G(\omega;\sigma^{-1})$	
windowed sinc		$\mathrm{rcos}(x/(aW))$ $\mathrm{sinc}(x/a)$	$\Leftrightarrow$	(see Figure 3.29)	

[Source: R. Szeliski]

Name	Kernel	Transform	Plot
box-3	$\frac{1}{3}\boxed{1}\boxed{1}\boxed{1}$	$\frac{1}{3}(1 + 2\cos\omega)$	
box-5	$\frac{1}{5}\boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{1}$	$\frac{1}{5}(1 + 2\cos\omega + 2\cos 2\omega)$	
linear	$\frac{1}{4}\boxed{1}\boxed{2}\boxed{1}$	$\frac{1}{2}(1 + \cos\omega)$	
binomial	$\frac{1}{16}\boxed{1}\boxed{4}\boxed{6}\boxed{4}\boxed{1}$	$\frac{1}{4}(1 + \cos\omega)^2$	
Sobel	$\frac{1}{2}\boxed{-1}\boxed{0}\boxed{1}$	$\sin\omega$	
corner	$\frac{1}{2}\boxed{-1}\boxed{2}\boxed{-1}$	$\frac{1}{2}(1 - \cos\omega)$	

[Source: R. Szeliski]

2D Fourier Transform

- Same as 1D, but in 2D. Now the sinusoid is

$$s(x, y) = \sin(\omega_x x + \omega_y y)$$

- The 2D Fourier in continuous domain is then

$$H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j\omega_x x + \omega_y y} dxdy$$

and in the discrete domain

$$H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-2\pi j \frac{k_x x + k_y y}{MN}}$$

where M and N are the width and height of the image.

2D Fourier Transform

- Same as 1D, but in 2D. Now the sinusoid is

$$s(x, y) = \sin(\omega_x x + \omega_y y)$$

- The 2D Fourier in continuous domain is then

$$H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j\omega_x x + \omega_y y} dxdy$$

  and in the discrete domain

$$H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-2\pi j \frac{k_x x + k_y y}{MN}}$$

  where M and N are the width and height of the image.

- All the properties carry over to 2D.

2D Fourier Transform

- Same as 1D, but in 2D. Now the sinusoid is

$$s(x, y) = \sin(\omega_x x + \omega_y y)$$

- The 2D Fourier in continuous domain is then

$$H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j\omega_x x + \omega_y y} dxdy$$

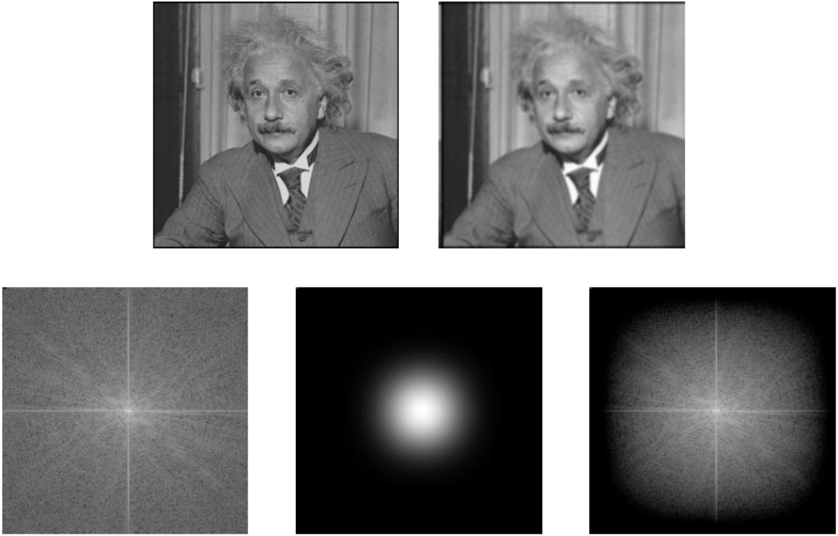
  and in the discrete domain

$$H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-2\pi j \frac{k_x x + k_y y}{MN}}$$

  where M and N are the width and height of the image.

- All the properties carry over to 2D.

Example of 2D Fourier Transform



[Source: A. Jepson]

Pyramids

- We might want to change resolution of an image before processing.
- We might not know which scale we want, e.g., when searching for a face in an image.

Pyramids

- We might want to change resolution of an image before processing.
- We might not know which scale we want, e.g., when searching for a face in an image.
- In this case, we will generate a full pyramid of different image sizes.

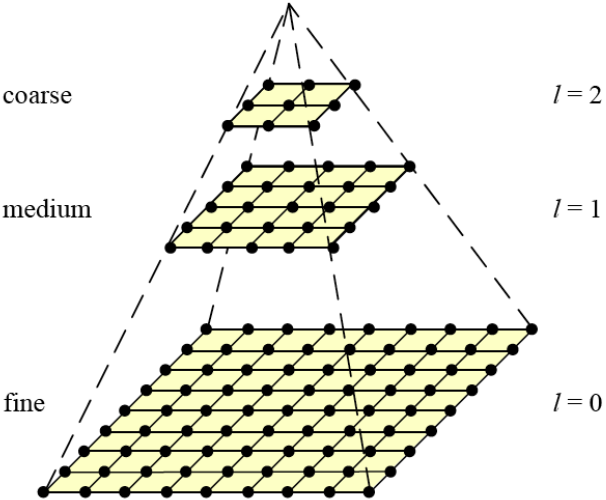
Pyramids

- We might want to change resolution of an image before processing.
- We might not know which scale we want, e.g., when searching for a face in an image.
- In this case, we will generate a full pyramid of different image sizes.
- Can also be used to accelerate the search, by first finding at the coarser level of the pyramid and then at the full resolution.

Pyramids

- We might want to change resolution of an image before processing.
- We might not know which scale we want, e.g., when searching for a face in an image.
- In this case, we will generate a full pyramid of different image sizes.
- Can also be used to accelerate the search, by first finding at the coarser level of the pyramid and then at the full resolution.

Image Pyramid



[Source: R. Szeliski]

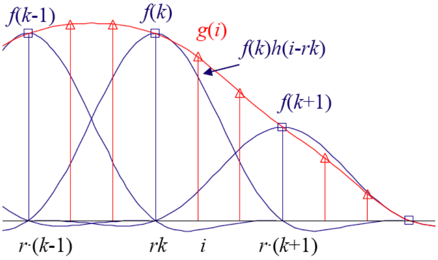
Interpolation

- To **interpolate** (or upsample) an image to a higher resolution, we need to select an interpolation kernel with which to convolve the image

$$g(i,j) = \sum_{k,l} f(k,l)h(i - rk, j - rl)$$

with r the up-sampling rate.

- The linear interpolator (corresponding to the tent kernel) produces interpolating piecewise linear curves.
- More complex kernels, e.g., B-splines.



[Source: R. Szeliski]

Decimation

- **Decimation**: reduces resolution

$$g(i,j) = \sum_{k,l} f(k,l)h(i - k/r, j - l/r)$$

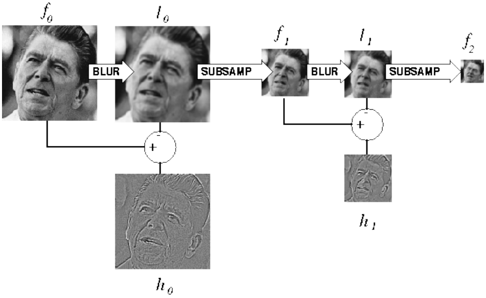
  with r the down-sampling rate.

- Different filters exist as well.

Multi-Resolution Representations

The most used one is the Laplacian pyramid:

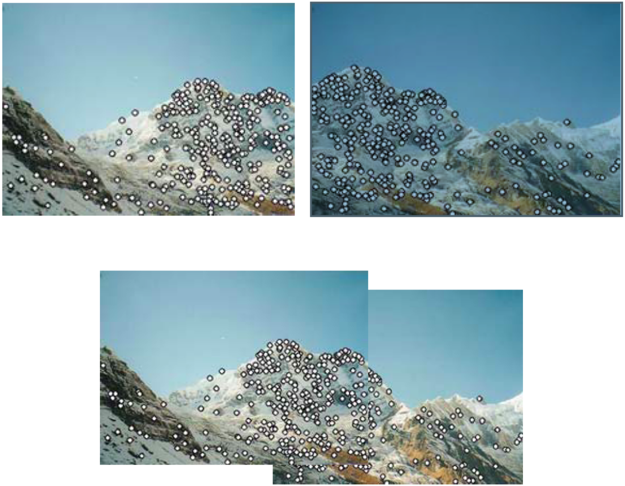
- We first blur and subsample the original image by a factor of two and store this in the next level of the pyramid.
- They then subtract this low-pass version from the original to yield the band-pass Laplacian image.
- The pyramid has perfect reconstruction: the Laplacian images plus the base-level Gaussian are sufficient to exactly reconstruct the original image.



- How do we reconstruct back?

Local features for **instance-level** recognition

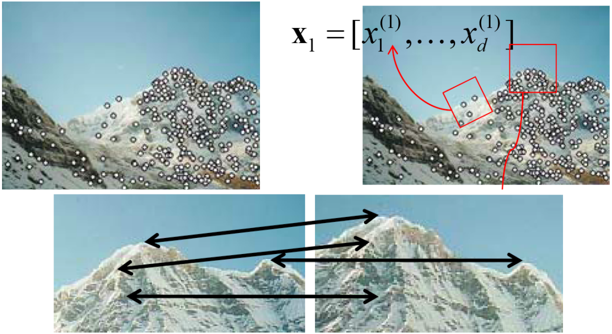
Application Example: Image stitching



[Source: K. Grauman]

Local features

- **Detection**: Identify the interest points.
- **Description**: Extract vector feature descriptor around each interest point.
- **Matching**: Determine correspondence between descriptors in two views.
- **Tracking**: alternative to matching that only searches a small neighborhood around each detected feature.



[Source: K. Grauman]

Goal: interest operator repeatability

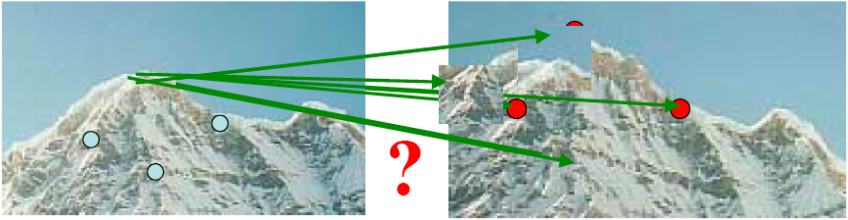
- We want to detect (at least some of) the same points in both images.
- We have to be able to run the detection procedure independently per image.

Figure: No chance to find the true matches

[Source: K. Grauman]

Goal: descriptor distinctiveness

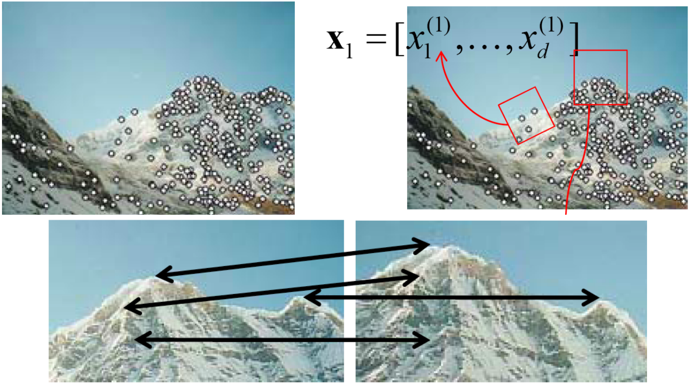
- We want to be able to reliably determine which point goes with which.
- Must provide some invariance to geometric and photometric differences between the two views.



[Source: K. Grauman]

Local features

- **Detection**: Identify the interest points.
- **Description**: Extract vector feature descriptor around each interest point.
- **Matching**: Determine correspondence between descriptors in two views.



[Source: K. Grauman]

What points to choose?

[Source: K. Grauman]

What points to choose?

- Textureless patches are nearly impossible to localize.
- Patches with large contrast changes (gradients) are easier to localize.

What points to choose?

- Textureless patches are nearly impossible to localize.
- Patches with large contrast changes (gradients) are easier to localize.
- But straight line segments at a single orientation suffer from the aperture problem, i.e., it is only possible to align the patches along the direction normal to the edge direction.

What points to choose?

- Textureless patches are nearly impossible to localize.
- Patches with large contrast changes (gradients) are easier to localize.
- But straight line segments at a single orientation suffer from the aperture problem, i.e., it is only possible to align the patches along the direction normal to the edge direction.
- Gradients in at least two (significantly) different orientations are the easiest, e.g., corners.

What points to choose?

- Textureless patches are nearly impossible to localize.
- Patches with large contrast changes (gradients) are easier to localize.
- But straight line segments at a single orientation suffer from the aperture problem, i.e., it is only possible to align the patches along the direction normal to the edge direction.
- Gradients in at least two (significantly) different orientations are the easiest, e.g., corners.

Corners as distinctive interest points

- We should easily recognize the point by looking through a small window.
- Shifting a window in any direction should give a large change in intensity.

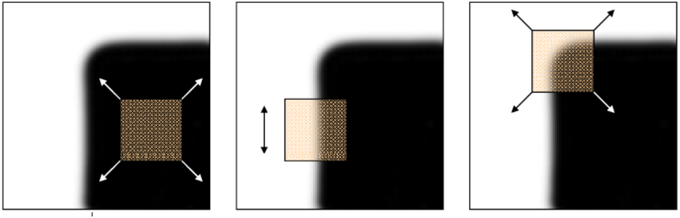


Figure: (left) flat region: no change in all directions, (center) edge: no change along the edge direction, (right) corner: significant change in all directions

[Source: Alyosha Efros, Darya Frolova, Denis Simakov]

A Simple Matching Criteria

- Compare two image patches using (weighted) summed square difference

$$E_{WSSD}(\mathbf{u}) = \sum_i w(\mathbf{p}_i)[I_1(\mathbf{p}_i + \mathbf{u}) - I_0(\mathbf{p}_i)]^2$$

  with I_0 and I_1 two images being compared, $\mathbf{u}(u_x, u_y)$ a displacement vector, $w(\mathbf{p})$ a spatially varying weighting function, and the summation i is over all the pixels in the patch.

- We do not know which other image locations the feature will end up being matched against.

A Simple Matching Criteria

- Compare two image patches using (weighted) summed square difference

$$E_{WSSD}(\mathbf{u}) = \sum_i w(\mathbf{p}_i)[I_1(\mathbf{p}_i + \mathbf{u}) - I_0(\mathbf{p}_i)]^2$$

  with I_0 and I_1 two images being compared, $\mathbf{u}(u_x, u_y)$ a displacement vector, $w(\mathbf{p})$ a spatially varying weighting function, and the summation i is over all the pixels in the patch.

- We do not know which other image locations the feature will end up being matched against.

- We can only compute how stable this metric is with respect to small variations in position u by comparing an image patch against itself.

A Simple Matching Criteria

- Compare two image patches using (weighted) summed square difference

$$E_{WSSD}(\mathbf{u}) = \sum_i w(\mathbf{p}_i)[I_1(\mathbf{p}_i + \mathbf{u}) - I_0(\mathbf{p}_i)]^2$$

  with I_0 and I_1 two images being compared, $\mathbf{u}(u_x, u_y)$ a displacement vector, $w(\mathbf{p})$ a spatially varying weighting function, and the summation i is over all the pixels in the patch.

- We do not know which other image locations the feature will end up being matched against.
- We can only compute how stable this metric is with respect to small variations in position u by comparing an image patch against itself.
- This is the **auto-correlation function**

$$E_{AC}(\Delta\mathbf{u}) = \sum_i w(\mathbf{p}_i)[I_0(\mathbf{p}_i + \Delta u) - I_0(\mathbf{p}_i)]^2$$

A Simple Matching Criteria

- Compare two image patches using (weighted) summed square difference

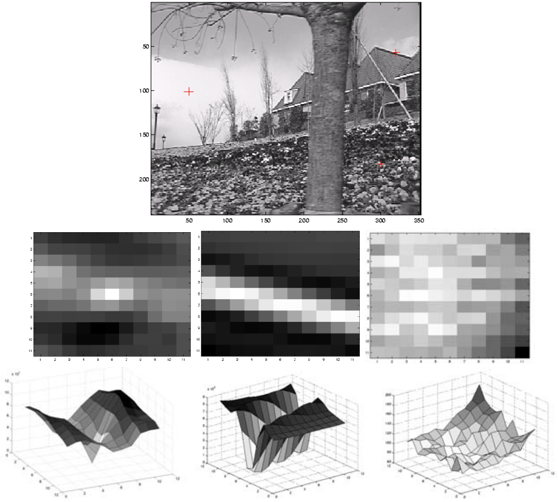
$$E_{WSSD}(\mathbf{u}) = \sum_i w(\mathbf{p}_i)[I_1(\mathbf{p}_i + \mathbf{u}) - I_0(\mathbf{p}_i)]^2$$

  with I_0 and I_1 two images being compared, $\mathbf{u}(u_x, u_y)$ a displacement vector, $w(\mathbf{p})$ a spatially varying weighting function, and the summation i is over all the pixels in the patch.

- We do not know which other image locations the feature will end up being matched against.
- We can only compute how stable this metric is with respect to small variations in position u by comparing an image patch against itself.
- This is the **auto-correlation function**

$$E_{AC}(\Delta\mathbf{u}) = \sum_i w(\mathbf{p}_i)[I_0(\mathbf{p}_i + \Delta u) - I_0(\mathbf{p}_i)]^2$$

Which one is better?



[Source: R. Szeliski]

How to select?

- Using a Taylor Series expansion $I_0(\mathbf{p}_i + \Delta\mathbf{u}) \approx I_0(\mathbf{p}_i) + \nabla I_0(\mathbf{p}_i)$ we can approximate the autocorrelation as

$$
\begin{aligned}
E_{AC}(\Delta\mathbf{u}) &= \sum_i w(\mathbf{p}_i)[I_0(\mathbf{p}_i + \Delta\mathbf{u}) - I_0(\mathbf{p}_i)]^2 \\
&\approx \sum_i w(\mathbf{p}_i)[I_0(\mathbf{p}_i) + \nabla I_0(\mathbf{p}_i)\Delta\mathbf{u} - I_0(\mathbf{p}_i)]^2 \\
&= \sum_i w(\mathbf{p}_i)[\nabla I_0(\mathbf{p}_i)\Delta\mathbf{u}]^2 \\
&= \Delta\mathbf{u}^T \mathbf{A} \Delta\mathbf{u}
\end{aligned}
$$

  with

$$
\nabla I_0(\mathbf{p}_i) = \left(\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y}\right)(\mathbf{p}_i)
$$

  the image gradient.

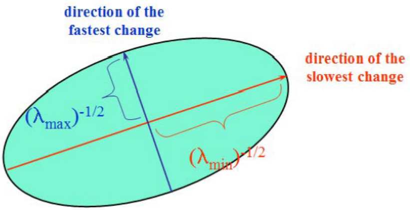
- Gradient can be computed with the filtering techniques we saw, e.g., derivatives of Gaussians.

More on selection

- The autocorrelation is $E_{AC}(\Delta\mathbf{u}) = \Delta\mathbf{u}^T \mathbf{A} \Delta\mathbf{u}$, with

$$\mathbf{A} = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$$

  where we have replaced the weighted summations with discrete convolutions with the weighting kernel w.
- **A** can be interpreted as a tensor where the outer products of the gradients are convolved with a weighting function.
- Eigenvalues a notion of uncertainty



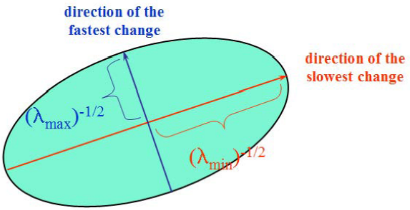
[Source: R. Szeliski]

More on selection

- The autocorrelation is $E_{AC}(\Delta\mathbf{u}) = \Delta\mathbf{u}^T\mathbf{A}\Delta\mathbf{u}$, with

$$\mathbf{A} = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$$

  where we have replaced the weighted summations with discrete convolutions with the weighting kernel w.
- $\mathbf{A}$ can be interpreted as a tensor where the outer products of the gradients are convolved with a weighting function.
- Eigenvalues a notion of uncertainty



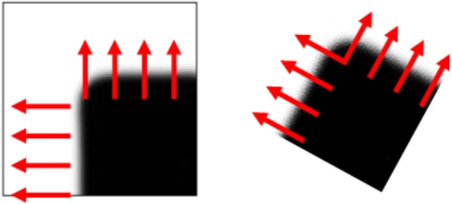
[Source: R. Szeliski]

Eigenvalues a notion of uncertainty

- $\mathbf{A}$ is symmetric

$$\mathbf{A} = \mathbf{U} \begin{bmatrix} \lambda_0 & 0 \\ 0 & \lambda_1 \end{bmatrix} \mathbf{U}^T \quad \text{with} \quad \mathbf{A}\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

- The eigenvalues of $\mathbf{A}$ reveal the amount of intensity change in the two principal orthogonal gradient directions in the window.
- How is this matrix for



[Source: R. Szeliski]

Local Feature Selection Criteria

- Shi and Tomasi, 94 proposed the smallest eigenvalue of **A**, i.e., $\lambda_0^{-1/2}$.
- Harris and Stephens, 88 is rotationally invariant and downweights edge-like features where $\lambda_1 \gg \lambda_0$

$$\det(\mathbf{A}) - \alpha \mathrm{trace}(\mathbf{A})^2 = \lambda_0 \lambda_1 - \alpha(\lambda_0 + \lambda_1)^2$$

Local Feature Selection Criteria

- Shi and Tomasi, 94 proposed the smallest eigenvalue of $\mathbf{A}$, i.e., $\lambda_0^{-1/2}$.
- Harris and Stephens, 88 is rotationally invariant and downweights edge-like features where $\lambda_1 \gg \lambda_0$

$$\det(\mathbf{A}) - \alpha \text{trace}(\mathbf{A})^2 = \lambda_0\lambda_1 - \alpha(\lambda_0 + \lambda_1)^2$$

- Triggs, 04 suggested

$$\lambda_0 - \alpha\lambda_1$$

also reduces the response at 1D edges, where aliasing errors sometimes inflate the smaller eigenvalue.

Local Feature Selection Criteria

- Shi and Tomasi, 94 proposed the smallest eigenvalue of $\mathbf{A}$, i.e., $\lambda_0^{-1/2}$.
- Harris and Stephens, 88 is rotationally invariant and downweights edge-like features where $\lambda_1 \gg \lambda_0$

$$\det(\mathbf{A}) - \alpha \mathrm{trace}(\mathbf{A})^2 = \lambda_0\lambda_1 - \alpha(\lambda_0 + \lambda_1)^2$$

- Triggs, 04 suggested

$$\lambda_0 - \alpha\lambda_1$$

  also reduces the response at 1D edges, where aliasing errors sometimes inflate the smaller eigenvalue.
- Brown et al, 05 use the harmonic mean

$$\frac{\det(\mathbf{A})}{\mathrm{trace}(\mathbf{A})} = \frac{\lambda_0\lambda_1}{\lambda_0 + \lambda_1}$$

Local Feature Selection Criteria

- Shi and Tomasi, 94 proposed the smallest eigenvalue of $\mathbf{A}$, i.e., $\lambda_0^{-1/2}$.
- Harris and Stephens, 88 is rotationally invariant and downweights edge-like features where $\lambda_1 \gg \lambda_0$

$$\det(\mathbf{A}) - \alpha \text{trace}(\mathbf{A})^2 = \lambda_0\lambda_1 - \alpha(\lambda_0 + \lambda_1)^2$$

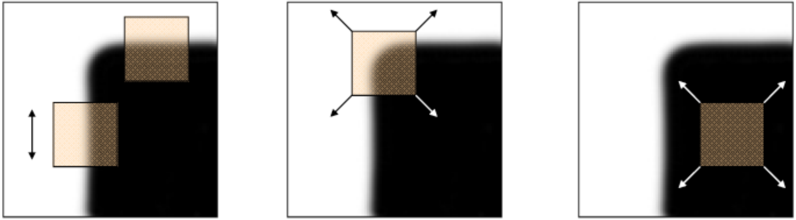
- Triggs, 04 suggested

$$\lambda_0 - \alpha\lambda_1$$

  also reduces the response at 1D edges, where aliasing errors sometimes inflate the smaller eigenvalue.
- Brown et al, 05 use the harmonic mean

$$\frac{\det(\mathbf{A})}{\text{trace}(\mathbf{A})} = \frac{\lambda_0\lambda_1}{\lambda_0 + \lambda_1}$$

Type of responses



"edge":
$\lambda_1 >> \lambda_2$
$\lambda_2 >> \lambda_1$

"corner":
λ_1 and λ_2 are large,
$\lambda_1 \sim \lambda_2$;

"flat" region
λ_1 and λ_2 are small;

[Source: K. Grauman]

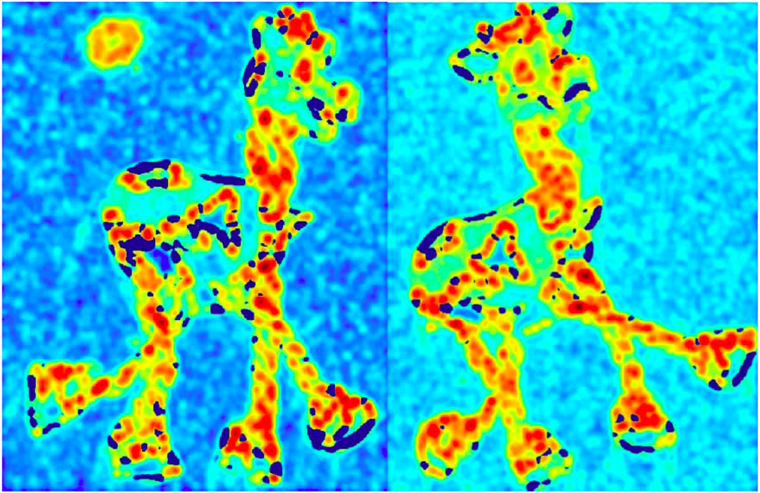
Harris Corner detector

1. Compute **A** for each image window to get their cornerness scores.
2. Find points whose surrounding window gave large corner response ($f >$ threshold).
3. Take the points of local maxima, i.e., perform non-maximum suppression.

Example

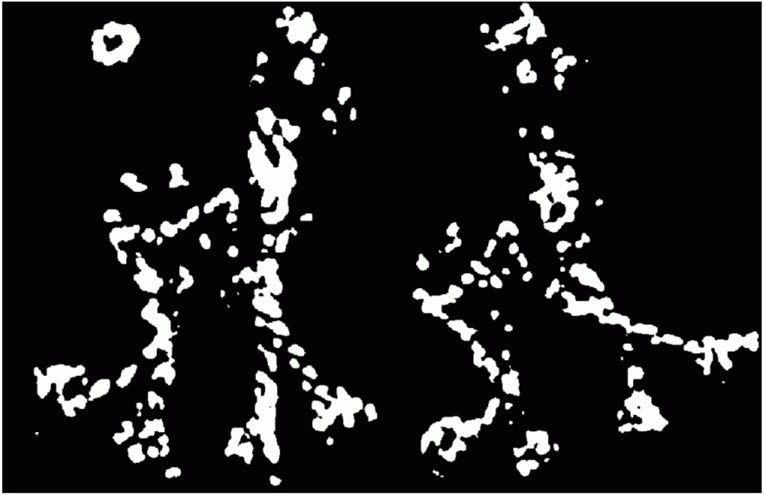
[Source: K. Grauman]

1) Compute Cornerness



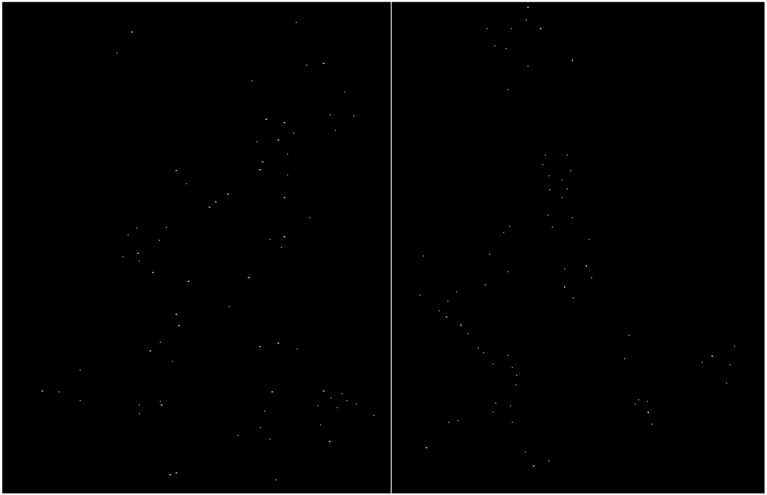
[Source: K. Grauman]

2) Find High Response



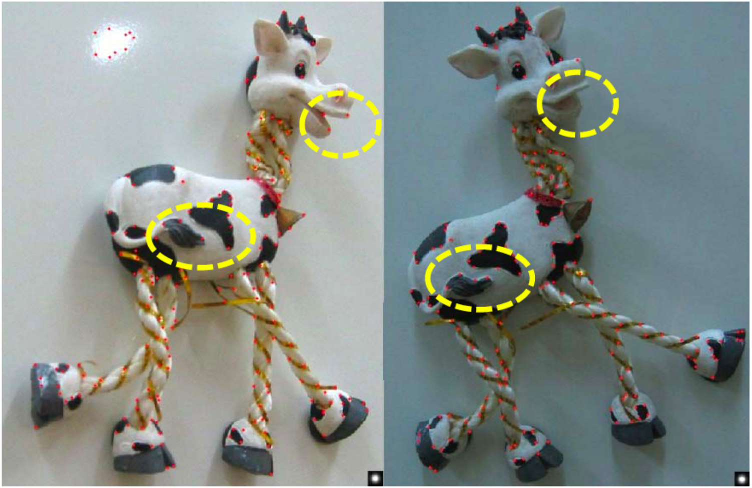
[Source: K. Grauman]

3) Non-maxima Suppresion



[Source: K. Grauman]

Results

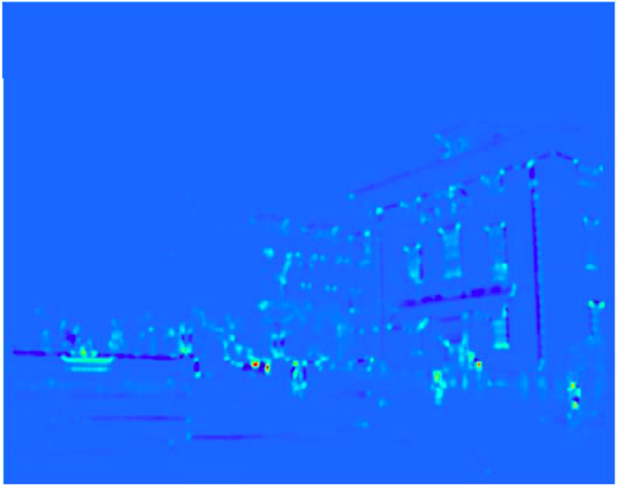


[Source: K. Grauman]

Another Example

[Source: K. Grauman]

Cornerness



[Source: K. Grauman]

Interest Points

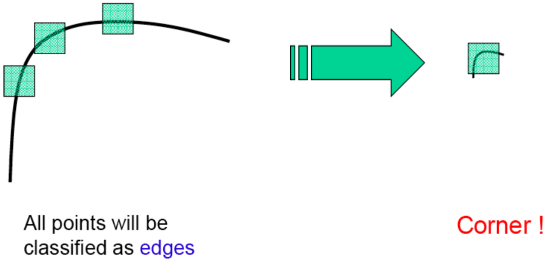
[Source: K. Grauman]

Properties of Harris Corner Detector

- Rotation invariant?

$$\mathbf{A} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} = \mathbf{U} \begin{bmatrix} \lambda_0 & 0 \\ 0 & \lambda_1 \end{bmatrix} \mathbf{U}^T \quad \text{with} \quad \mathbf{A}\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

- Scale Invariant?



[Source: K. Grauman]

Scale invariant interest points

How can we independently select interest points in each image, such that the detections are repeatable across different scales?

- Extract features at a variety of scales, e.g., by using multiple resolutions in a pyramid, and then matching features at the same level.

Scale invariant interest points

How can we independently select interest points in each image, such that the detections are repeatable across different scales?

- Extract features at a variety of scales, e.g., by using multiple resolutions in a pyramid, and then matching features at the same level.
- When does this work?

Scale invariant interest points

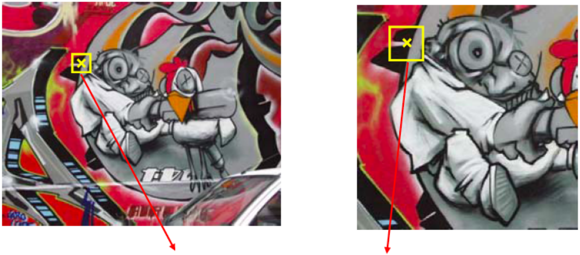
How can we independently select interest points in each image, such that the detections are repeatable across different scales?

- Extract features at a variety of scales, e.g., by using multiple resolutions in a pyramid, and then matching features at the same level.
- When does this work?
- More efficient to extract features that are stable in both location and scale.

Scale invariant interest points

How can we independently select interest points in each image, such that the detections are repeatable across different scales?

- Extract features at a variety of scales, e.g., by using multiple resolutions in a pyramid, and then matching features at the same level.
- When does this work?
- More efficient to extract features that are stable in both location and scale.
- Find scale that gives local maxima of a function f in both position and scale.

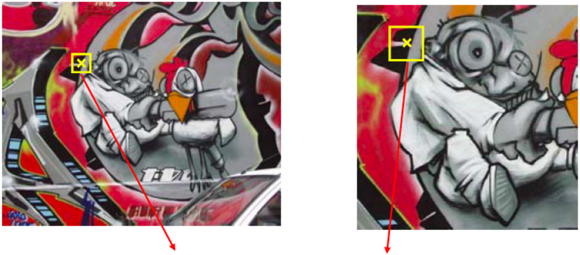


$$f(I_{i_1 \ldots i_m}(x, \sigma)) = f(I_{i_1 \ldots i_m}(x', \sigma'))$$

Scale invariant interest points

How can we independently select interest points in each image, such that the detections are repeatable across different scales?

- Extract features at a variety of scales, e.g., by using multiple resolutions in a pyramid, and then matching features at the same level.
- When does this work?
- More efficient to extract features that are stable in both location and scale.
- Find scale that gives local maxima of a function f in both position and scale.



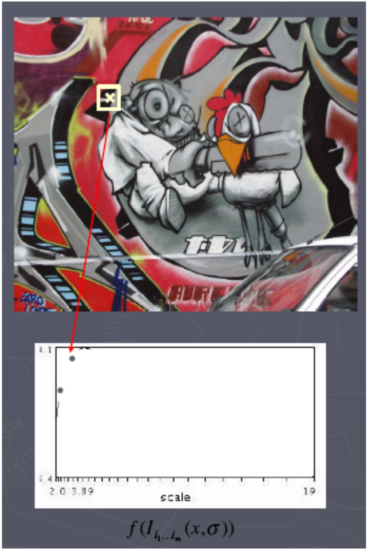
$$f(I_{i_1 \ldots i_m}(x, \sigma)) = f(I_{i_1 \ldots i_m}(x', \sigma'))$$

Automatic Scale Selection

Function responses for increasing scale (scale signature).

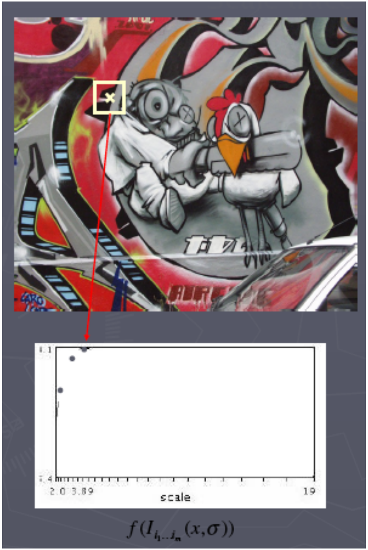
Automatic Scale Selection

Function responses for increasing scale (scale signature).



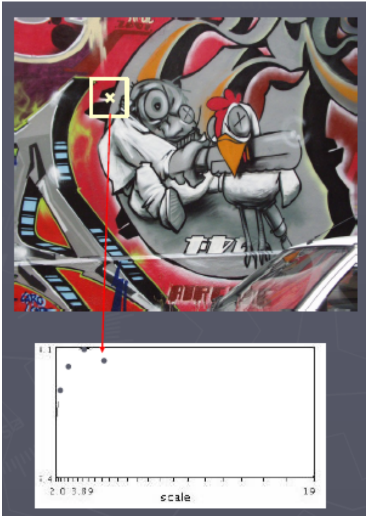
Automatic Scale Selection

Function responses for increasing scale (scale signature).



Automatic Scale Selection

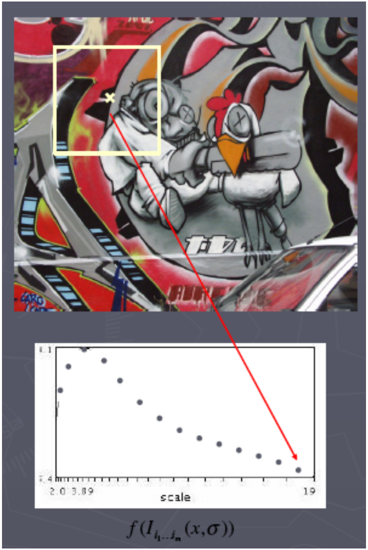
Function responses for increasing scale (scale signature).



$f(I_{i_1 \ldots i_m}(x, \sigma))$

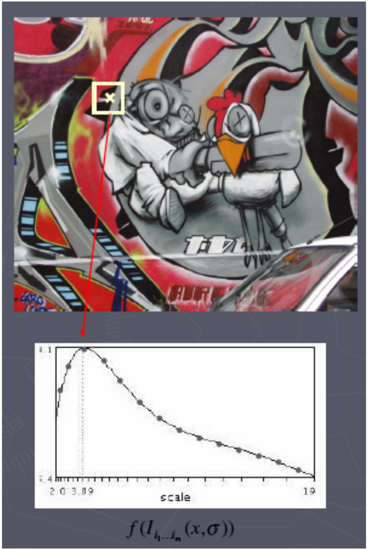
Automatic Scale Selection

Function responses for increasing scale (scale signature).



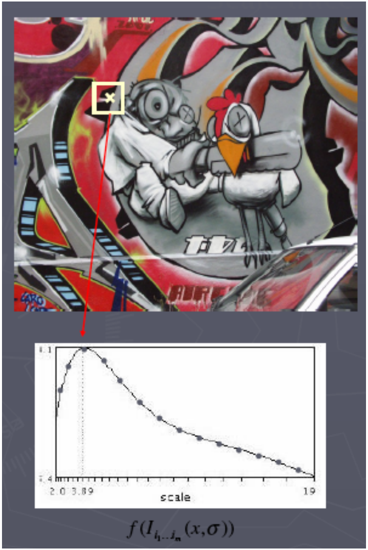
Automatic Scale Selection

Function responses for increasing scale (scale signature).



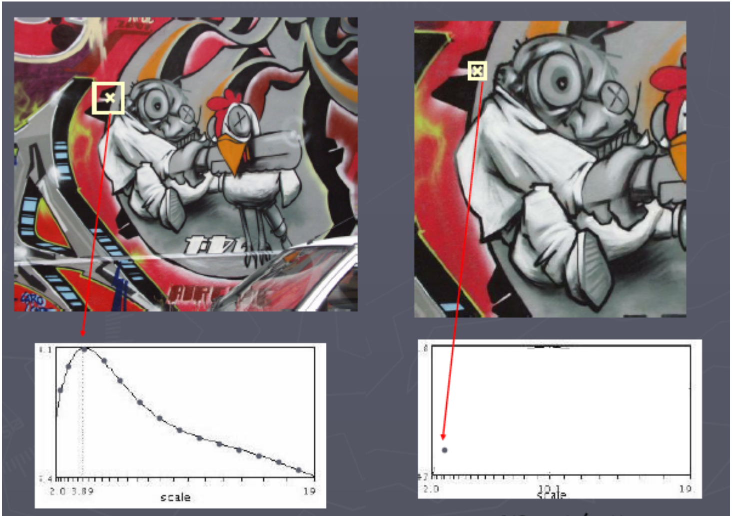
Automatic Scale Selection

Function responses for increasing scale (scale signature).



Automatic Scale Selection

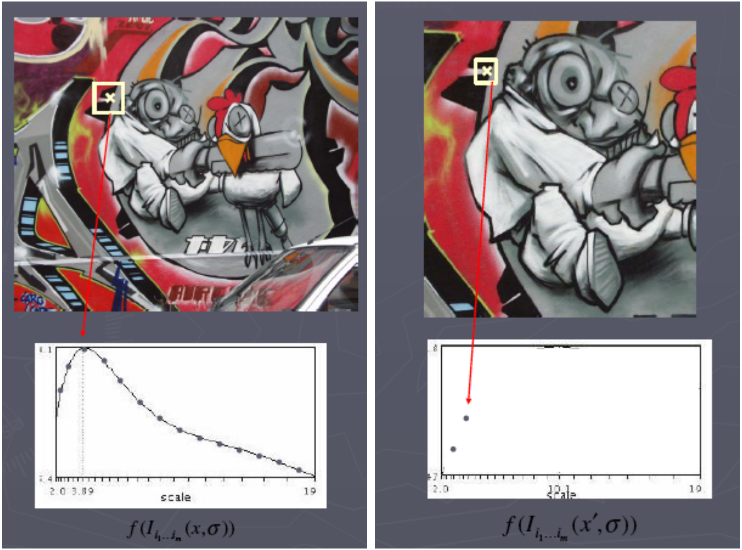
Function responses for increasing scale (scale signature).



$f(I_{i_1 \ldots i_m}(x, \sigma))$ $f(I_{i_1 \ldots i_m}(x', \sigma))$

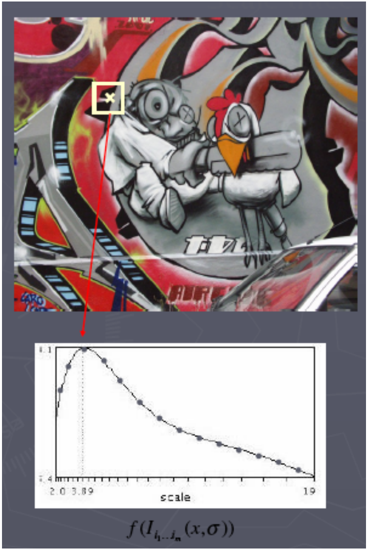
Automatic Scale Selection

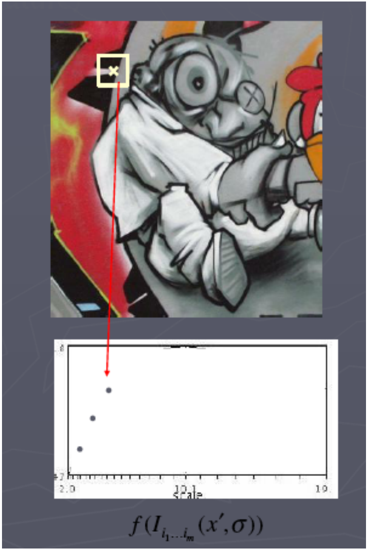
Function responses for increasing scale (scale signature).



Automatic Scale Selection

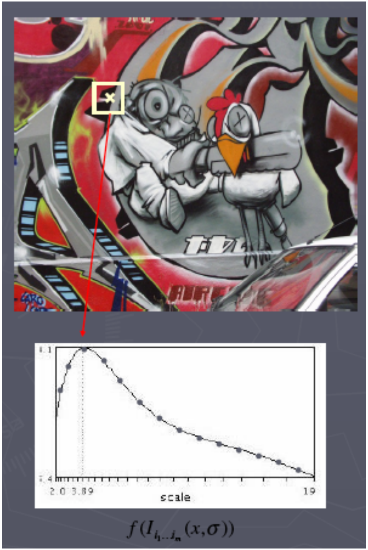
Function responses for increasing scale (scale signature).





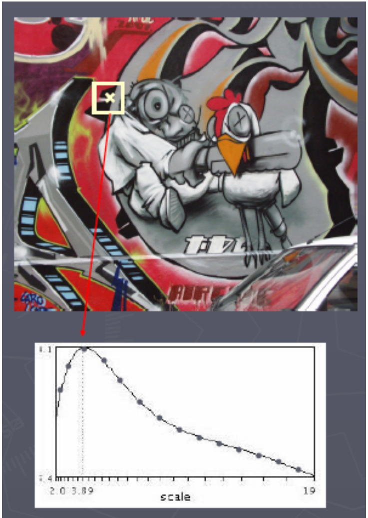
Automatic Scale Selection

Function responses for increasing scale (scale signature).



Automatic Scale Selection

Function responses for increasing scale (scale signature).

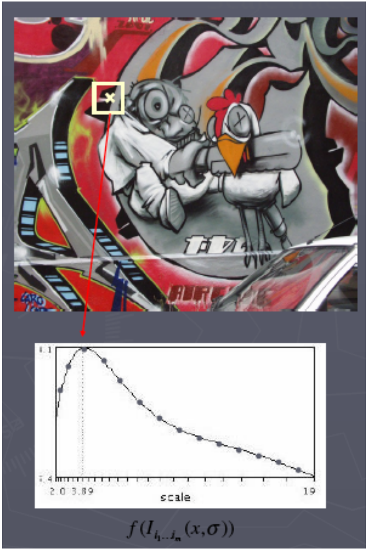


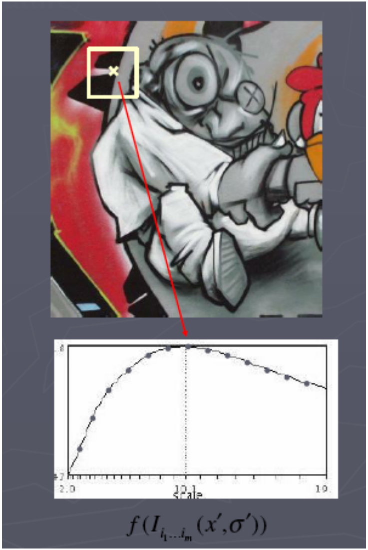
$f(I_{i_1 \ldots i_m}(x, \sigma))$

$f(I_{i_1 \ldots i_m}(x', \sigma))$

Automatic Scale Selection

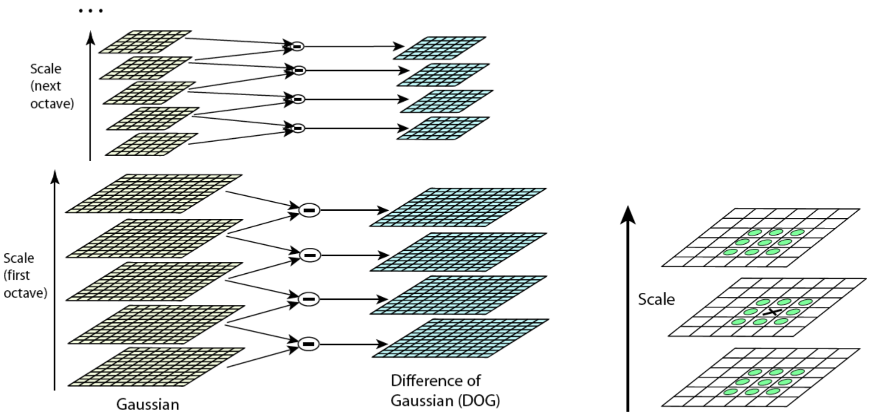
Function responses for increasing scale (scale signature).





What can the signature function be?

- Lindeberg (1998): extrema in the Laplacian of Gaussian (LoG).
- Lowe (2004) proposed computing a set of sub-octave Difference of Gaussian filters looking for 3D (space+scale) maxima in the resulting structure.

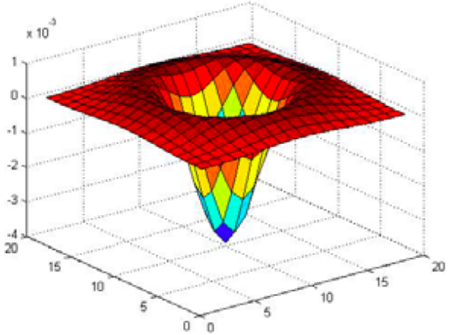


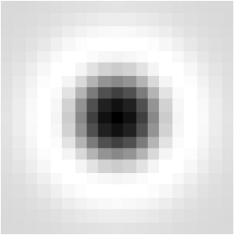
[Source: R. Szeliski]

Blob detection

- Laplacian of Gaussian: Circularly symmetric operator for blob detection in 2D

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

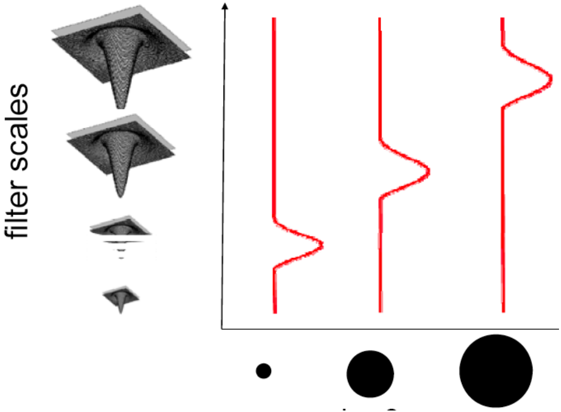




[Source: K. Grauman]

Blob detection in 2D: scale selection

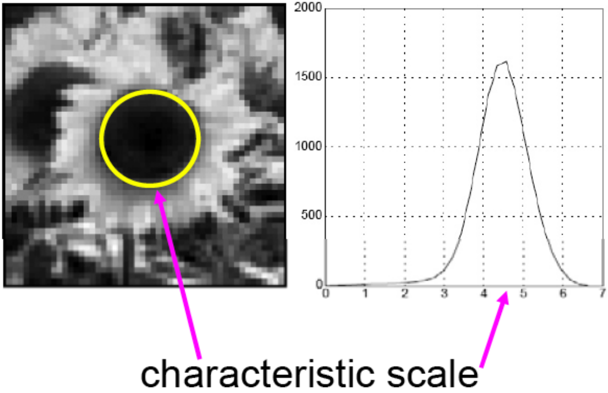
Laplacian-of-Gaussian = blob detector



[Source: B. Leibe]

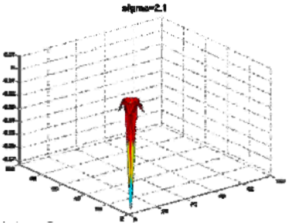
Characteristic Scale

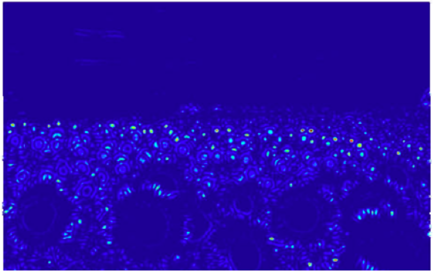
- We define the **characteristic scale** as the scale that produces peak of Laplacian response



[Source: S. Lazebnik]

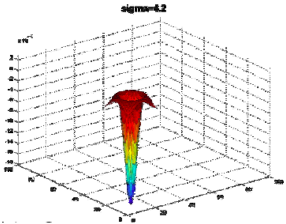
Example

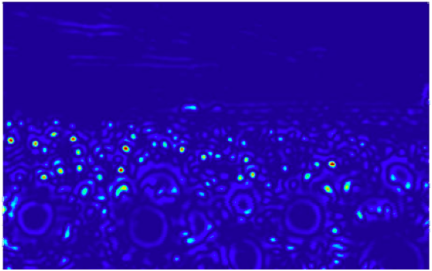




[Source: K. Grauman]

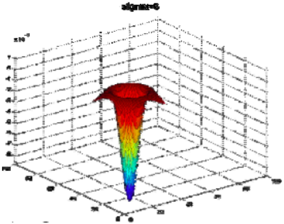
Example

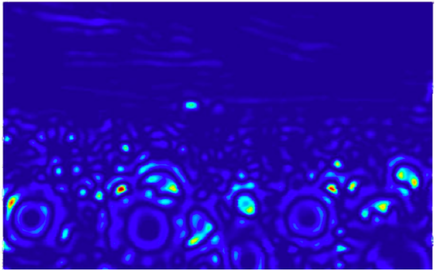




[Source: K. Grauman]

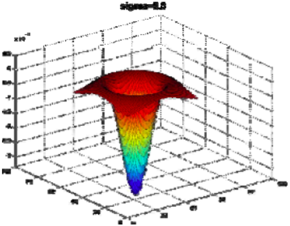
Example

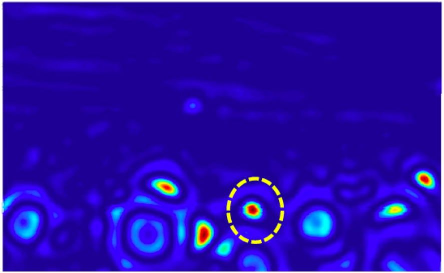




[Source: K. Grauman]

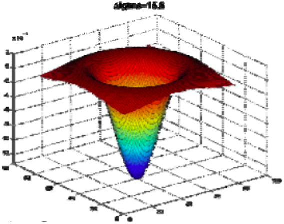
Example

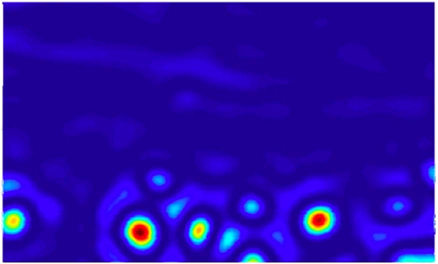




[Source: K. Grauman]

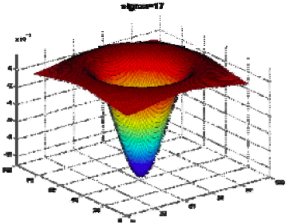
Example

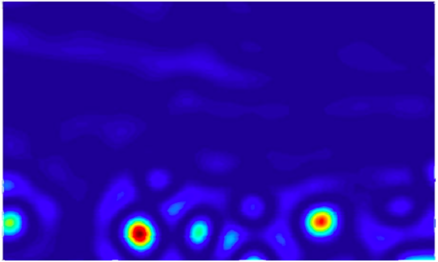




[Source: K. Grauman]

Example

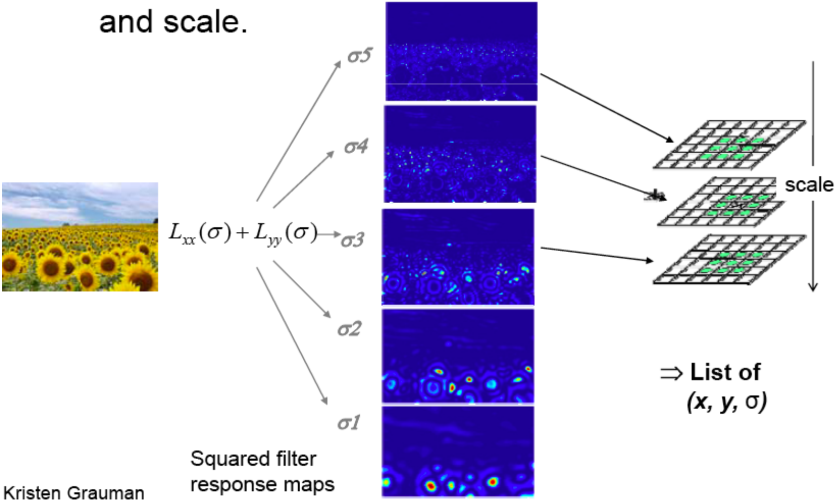




[Source: K. Grauman]

Scale invariant interest points

Interest points are local maxima in both position and scale.



Example

[Source: S. Lazebnik]

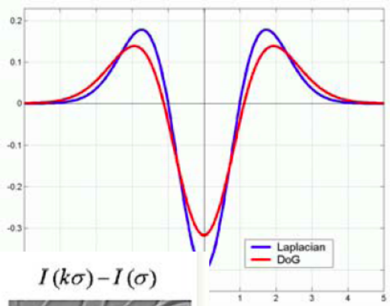
Fast approximation

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

(Laplacian)

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

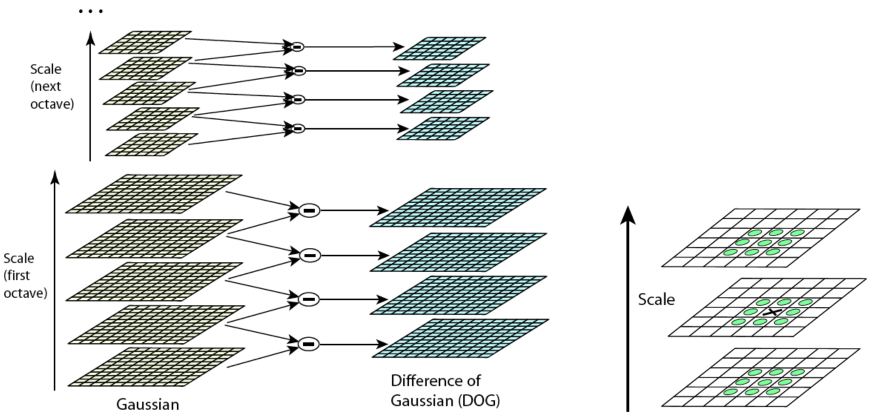
(Difference of Gaussians)



[Source: K. Grauman]

Lowe's DoG

- Lowe (2004) proposed computing a set of sub-octave Difference of Gaussian filters looking for 3D (space+scale) maxima in the resulting structure



[Source: R. Szeliski]

Laplacian vs Hessian

- Laplacian of Gaussians is scale invariant.
- Simple and efficient.
- But fires more on edges than determinant of hessian

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.
- **Distinctive**: individual features can be matched to a large database of objects.

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.
- **Distinctive**: individual features can be matched to a large database of objects.
- **Quantity**: many features can be generated for even small objects.

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.
- **Distinctive**: individual features can be matched to a large database of objects.
- **Quantity**: many features can be generated for even small objects.
- **Accurate**: precise localization.

Properties of the ideal feature

- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.
- **Distinctive**: individual features can be matched to a large database of objects.
- **Quantity**: many features can be generated for even small objects.
- **Accurate**: precise localization.
- **Efficient**: close to real-time performance.

Properties of the ideal feature

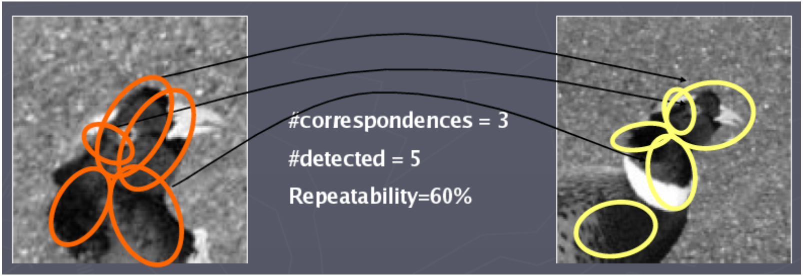
- **Local**: features are local, so robust to occlusion and clutter (no prior segmentation).
- **Invariant**: to certain transformations, e.g, scale, rotation.
- **Robust**: noise, blur, discretization, compression, etc. do not have a big impact on the feature.
- **Distinctive**: individual features can be matched to a large database of objects.
- **Quantity**: many features can be generated for even small objects.
- **Accurate**: precise localization.
- **Efficient**: close to real-time performance.

A lot of other interest point detectors

- Hessian
- Lowe: DoG
- Lindeberg: scale selection
- Miikolajczyk & Schmid: Hessian/Harris-Laplacian/Affine
- Tuyttelaars & Van Gool: EBR and IBR
- Matas: MSER
- Kadir & Brrady: Salient Regions
- Speeded–Up Robust Features (SURF) of Bay et al.

Evaluation criteria: repeatability

- Repeatability rate: percentage of detected that have correct corresponding points
- What's the problem of this?



[Source: T. Tuyttellaars]

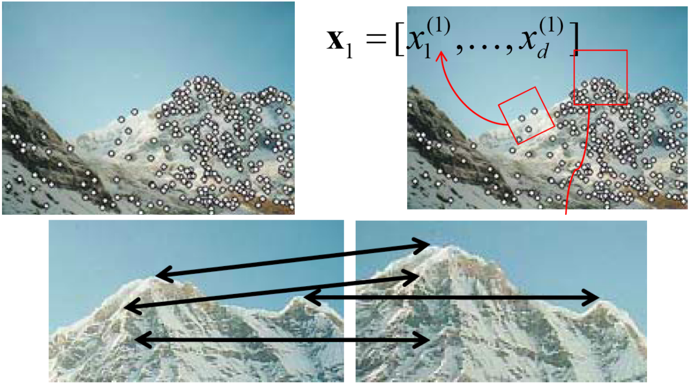
Evaluation criteria: repeatability

- Two points are in correspondence if the intersection over union is bigger than a certain threshold.

[Source: T. Tuyttellaars]

Local features

- **Detection**: Identify the interest points.
- **Description**: Extract vector feature descriptor around each interest point.
- **Matching**: Determine correspondence between descriptors in two views.



[Source: K. Grauman]

The ideal feature descriptor

- Repeatable (invariant/robust)
- Distinctive
- Compact
- Efficient

Invariances

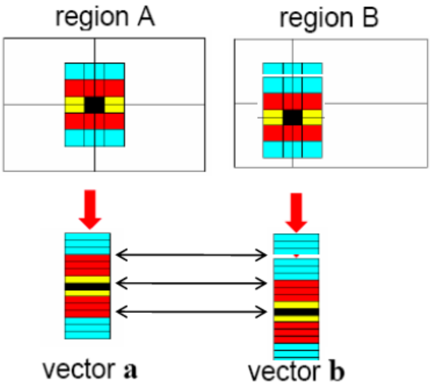
[Source: T. Tuytelaars]

Invariances

[Source: T. Tuytelaars]

Raw Pixels as Local Descriptrs

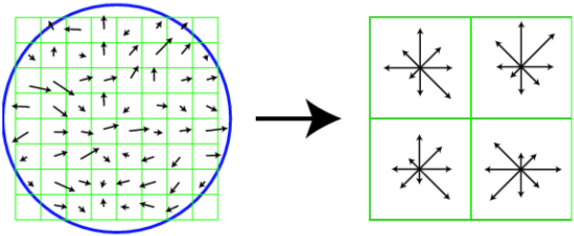
- The simplest way is to write down the list of intensities to form a feature vector, and normalize them (i.e., mean 0, variance 1).
- But this is very sensitive to even small shifts, rotations.



[Source: K. Grauman]

SIFT descriptor [Lowe 2004]

- Compute the gradient at each pixel in a 16×16 window around the detected keypoint, using the appropriate level of the Gaussian pyramid at which the keypoint was detected.
- Doweight gradients by a Gaussian fall-off function (blue circle) to reduce the influence of gradients far from the center.
- In each 4×4 quadrant, compute a gradient orientation histogram using 8 orientation histogram bins.



(a) image gradients (b) keypoint descriptor

[Source: R. Szeliski]

SIFT descriptor [Lowe 2004]

- To reduce the effects of location and dominant orientation misestimation, each of the original 256 weighted gradient magnitudes is softly added to $2 \times 2 \times 2$ histogram bins using trilinear interpolation.
- The resulting 128 non-negative values form a raw version of the SIFT descriptor vector.
- To reduce the effects of contrast or gain (additive variations are already removed by the gradient), the 128-D vector is normalized to unit length.
- To further make the descriptor robust to other photometric variations, values are clipped to 0.2 and the resulting vector is once again renormalized to unit length.
- Great engineering effort!
- Why subpatches?
- Why does SIFT have some illumination invariance?

SIFT descriptor [Lowe 2004]

Extraordinarily robust matching technique

- Changes in viewpoint: up to about 60 degree out of plane rotation
- Changes in illumination: sometimes even day vs. night
- Fast and efficientcan run in real time
- Lots of code available

[Source: S. Seitz]

Example

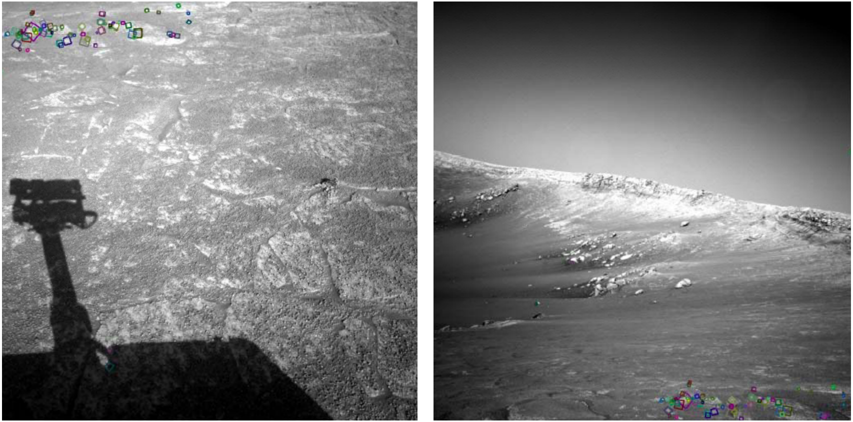


Figure: NASA Mars Rover images with SIFT feature matches

[Source: N. Snavely]

SIFT properties

Invariant to

- Scale
- Rotation

Partially invariant to

- Illumination changes
- Camera viewpoint
- Occlusion, clutter

Making descriptor rotation invariant

- Rotate patch according to its dominant gradient orientation
- This puts the patches into a canonical orientation

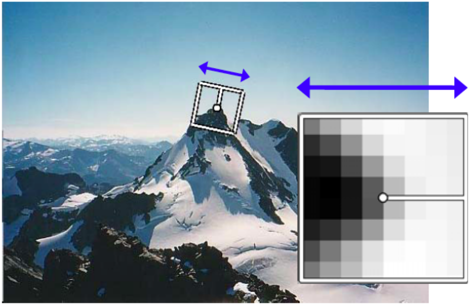
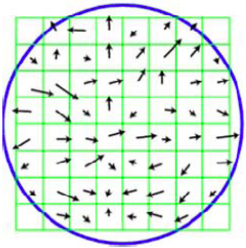


Figure: Figure from M. Brown

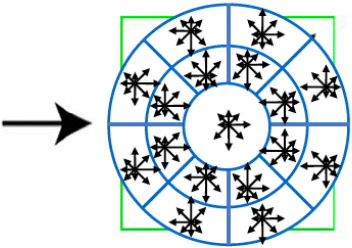
[Source: K. Grauman]

Gradient location-orientation histogram (GLOH)

- Developed by Mikolajczyk and Schmid (2005): variant on SIFT that uses a log-polar binning structure instead of the four quadrants.
- The spatial bins are 11, and 15, with eight angular bins (except for the central region), for a total of 17 spatial bins and 16 orientation bins.
- The 272D histogram is then projected onto a 128D descriptor using PCA trained on a large database.



(a) image gradients



(b) keypoint descriptor

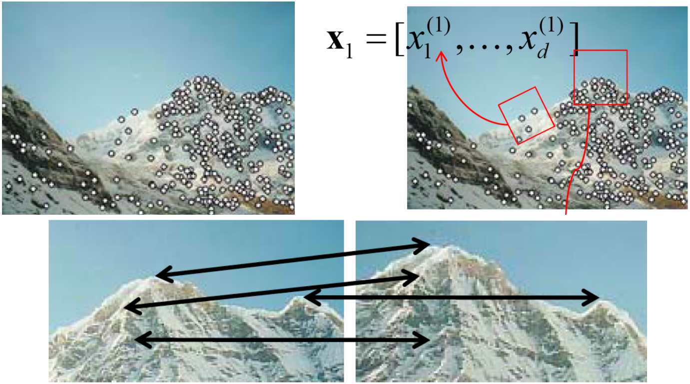
[Source: R. Szeliski]

Other Descriptors

- Steerable filters
- moment invariants,
- complex filters
- shape contexts,,
- PCA-SIFT,
- HOG,
- SURF
- DAISY

Local features

- **Detection**: Identify the interest points.
- **Description**: Extract vector feature descriptor around each interest point.
- **Matching**: Determine correspondence between descriptors in two views.



[Source: K. Grauman]

Matching local features

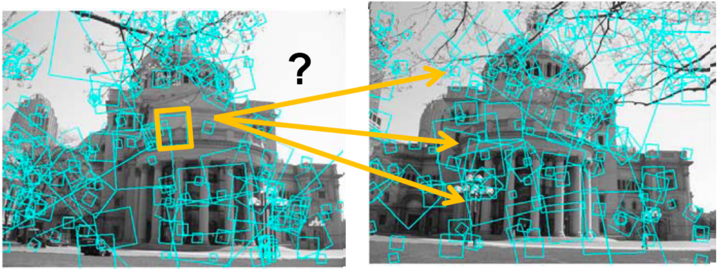
Once we have extracted features and their descriptors, we need to match the features between these images.

- Matching strategy: which correspondences are passed on to the next stage
- Devise efficient data structures and algorithms to perform this matching

Figure: Images from K. Grauman

Matching local features

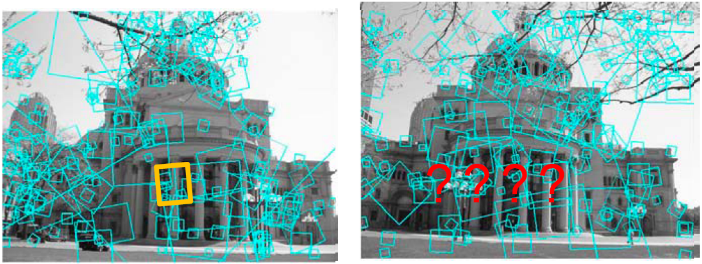
- To generate candidate matches, find patches that have the most similar appearance (e.g., lowest SSD)
- Simplest approach: compare them all, take the closest (or closest k, or within a thresholded distance)



[Source: K. Grauman]

Ambiguous matches

- At what SSD value do we have a good match?
- To add robustness, consider ratio of distance to best match to distance to second best match
  - If low, first match looks good.
  - If high, could be ambiguous match.



[Source: K. Grauman]

Matching SIFT Descriptors

- Nearest neighbor (Euclidean distance)
- Threshold ratio of nearest to 2nd nearest descriptor

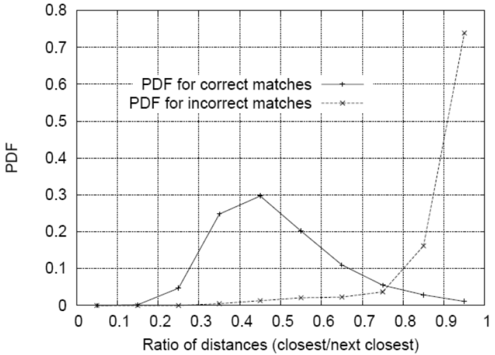


Figure: Images from D. Lowe

[Source: K. Grauman]