Visual Recognition: Filtering and Transformations

Raquel Urtasun

TTI Chicago

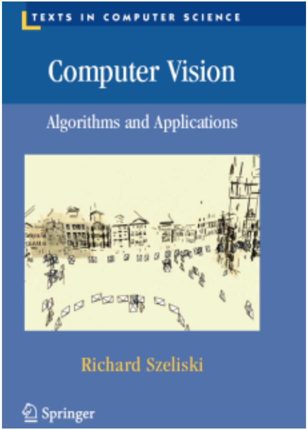
Jan 10, 2012

Today's lecture ...

- Image formation and color
- Image Filtering
- Additional transformations

Material

- Chapter 2 and 3 of Rich Szeliski book

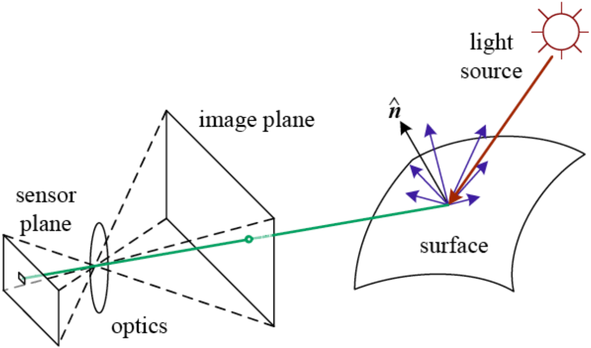


- Available online here

How is an image created?

The image formation process that produced a particular image depends on

- lighting conditions
- scene geometry,
- surface properties
- camera optics

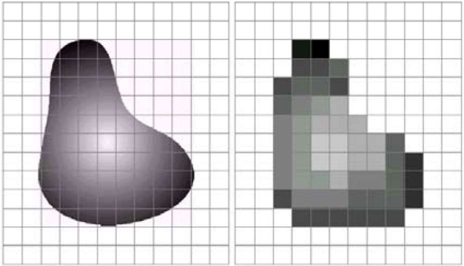


[Source: R. Szeliski]

Image formation and color

From photons to RGB values

- **Sample** the 2D space on a regular grid.
- **Quantize** each sample, i.e., the photons arriving at each active cell are integrated and then digitized.



[Source: D. Hoiem]

Problems: Aliasing

- Shannons Sampling Theorem shows that the minimum sampling

$$f_s \geq 2f_{max}$$

- If you haven't seen this... take a class on Fourier analysis... everyone should have at least one!

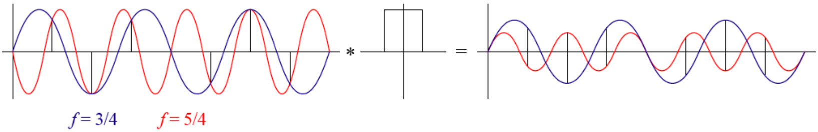


Figure: example of a 1D signal

[Source: R. Szeliski]

And in 2D...

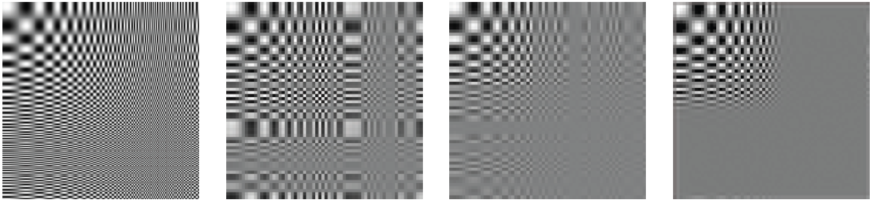


Figure: (a) Example of a 2D signal. (b–d) downsampled with different filters

[Source: R. Szeliski]

Color Cameras

- Each color camera integrates light according to the spectral response function of its red, green, and blue sensors.

$$R = \int L(\lambda) S_R(\lambda) d\lambda$$
$$G = \int L(\lambda) S_G(\lambda) d\lambda$$
$$B = \int L(\lambda) S_B(\lambda) d\lambda$$

  where λ is the incoming spectrum of light at a given pixel, and S_R, S_G, S_B, are the red, green, and blue spectral sensitivities of the corresponding sensors.

Bayer Pattern

- Color cameras use color filter arrays (CFAs), where alternating sensors are covered by different colored filters.
- More green filters as the luminance signal is mostly determined by green values and the visual system is much more sensitive to high frequency detail in luminance than in chrominance.

Bayer Pattern

- Color cameras use color filter arrays (CFAs), where alternating sensors are covered by different colored filters.
- More green filters as the luminance signal is mostly determined by green values and the visual system is much more sensitive to high frequency detail in luminance than in chrominance.
- **Demosaicing**: interpolate the missing color values to have RGB values for all pixels.

G	R	G	R
B	G	B	G
G	R	G	R
B	G	B	G

rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb
rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb

Figure: (a) Bayer Pattern. (b) interpolated RGB

[Source: R. Szeliski]

Bayer Pattern

- Color cameras use color filter arrays (CFAs), where alternating sensors are covered by different colored filters.
- More green filters as the luminance signal is mostly determined by green values and the visual system is much more sensitive to high frequency detail in luminance than in chrominance.
- **Demosaicing**: interpolate the missing color values to have RGB values for all pixels.

G	R	G	R
B	G	B	G
G	R	G	R
B	G	B	G

rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb
rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb

Figure: (a) Bayer Pattern. (b) interpolated RGB

[Source: R. Szeliski]

RGB components

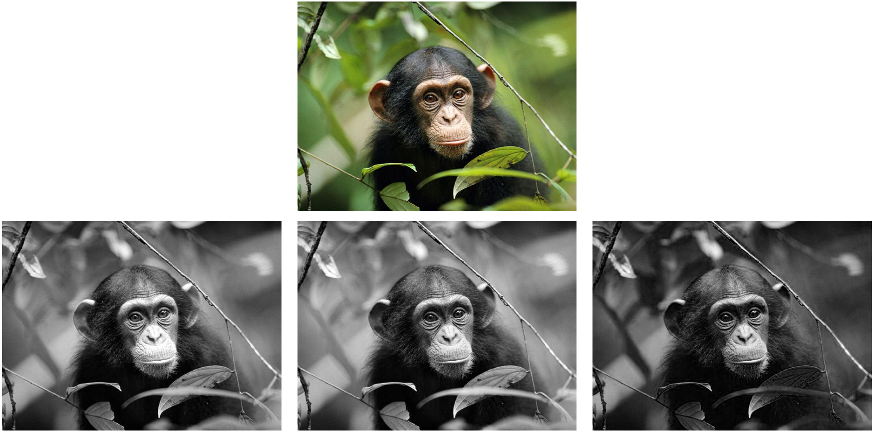
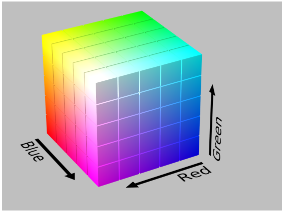
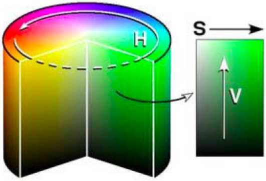


Figure: (a) Original image. (b) R component, (c) G component, (d) B component.

HSV color space



(RGB)



(HSV)

- There are other color spaces that might be better from a processing perspective: Lab, HSV, etc

HSV components

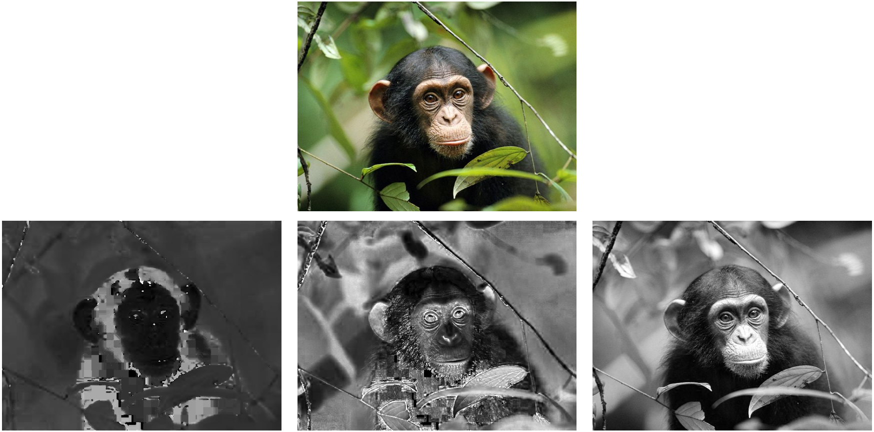


Figure: (a) Original image. (b) H component, (c) S component, (d) V component.

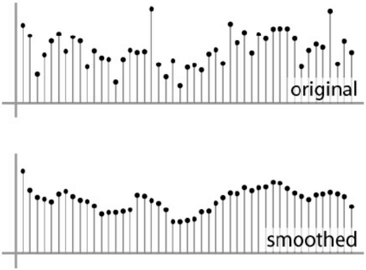
Filtering

Applications of Filtering

- Enhance an image, e.g., denoise, resize.
- Extract information, e.g., texture, edges.
- Detect patterns, e.g., template matching.

Noise reduction

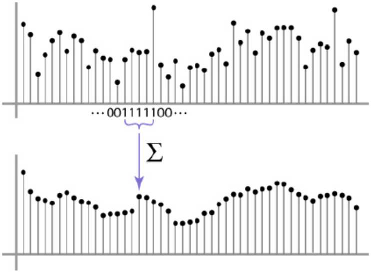
- Simplest thing: replace each pixel by the average of its neighbors.
- This assumes that neighboring pixels are similar, and the noise to be independent from pixel to pixel.



[Source: S. Marschner]

Noise reduction

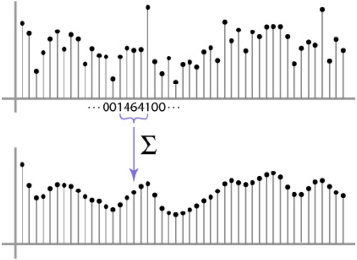
- Simpler thing: replace each pixel by the average of its neighbors
- This assumes that neighboring pixels are similar, and the noise to be independent from pixel to pixel.
- Moving average in 1D: [1, 1, 1, 1, 1]/5



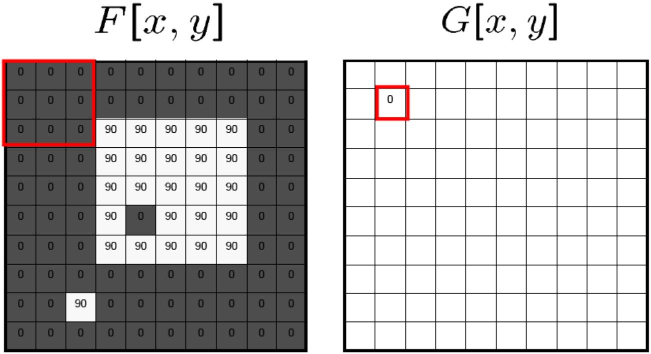
[Source: S. Marschner]

Noise reduction

- Simpler thing: replace each pixel by the average of its neighbors
- This assumes that neighboring pixels are similar, and the noise to be independent from pixel to pixel.
- Non-uniform weights [1, 4, 6, 4, 1] / 16

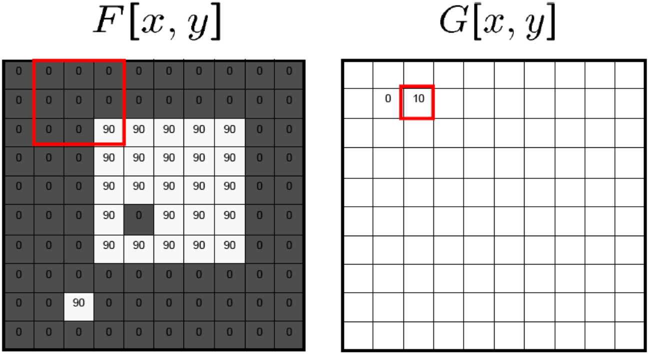


[Source: S. Marschner]



[Source: S. Seitz]

Moving Average in 2D



[Source: S. Seitz]

Moving Average in 2D

$F[x, y]$

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

$G[x, y]$

	0	10	20						

[Source: S. Seitz]

$F[x, y]$

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

$G[x, y]$

	0	10	20	30					

[Source: S. Seitz]

Moving Average in 2D

$F[x, y]$

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

$G[x, y]$

	0	10	20	30	30				

[Source: S. Seitz]

$$F[x, y]$$

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

$$G[x, y]$$

	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	

[Source: S. Seitz]

Linear Filtering: Correlation

- Involves weighted combinations of pixels in small neighborhoods.
- The output pixels value is determined as a weighted sum of input pixel values

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

Linear Filtering: Correlation

- Involves weighted combinations of pixels in small neighborhoods.
- The output pixels value is determined as a weighted sum of input pixel values

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

- The entries of the weight kernel or mask $h(k,l)$ are often called the filter coefficients.

Linear Filtering: Correlation

- Involves weighted combinations of pixels in small neighborhoods.
- The output pixels value is determined as a weighted sum of input pixel values

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

- The entries of the weight kernel or mask $h(k,l)$ are often called the filter coefficients.
- This operator is the **correlation** operator

$$g = f \otimes h$$

Linear Filtering: Correlation

- Involves weighted combinations of pixels in small neighborhoods.
- The output pixels value is determined as a weighted sum of input pixel values

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

- The entries of the weight kernel or mask $h(k,l)$ are often called the filter coefficients.
- This operator is the **correlation** operator

$$g = f \otimes h$$

Convolution Example

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

$*$

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

$=$

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

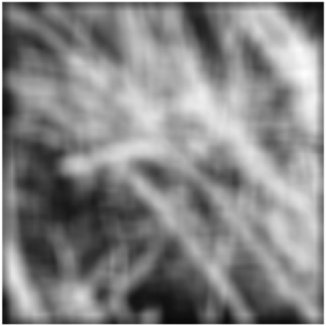
Figure: What does this filter do?

[Source: R. Szeliski]

Smoothing by averaging

depicts box filter:
white = high value, black = low value

original



filtered

- What if the filter size was 5 x 5 instead of 3 x 3?

[Source: K. Graumann]

Gaussian filter

- What if we want nearest neighboring pixels to have the most influence on the output?
- Removes high-frequency components from the image (low-pass filter).

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

$F[x, y]$

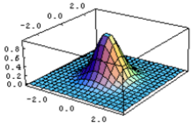
$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$H[u, v]$

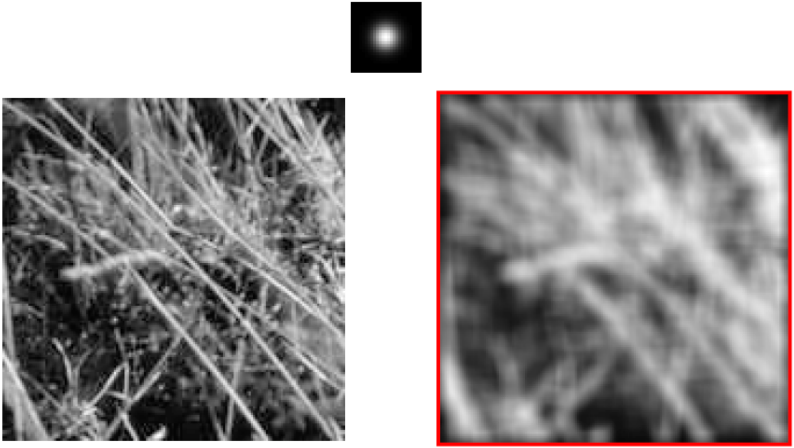
This kernel is an approximation of a 2d Gaussian function:

$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



[Source: S. Seitz]

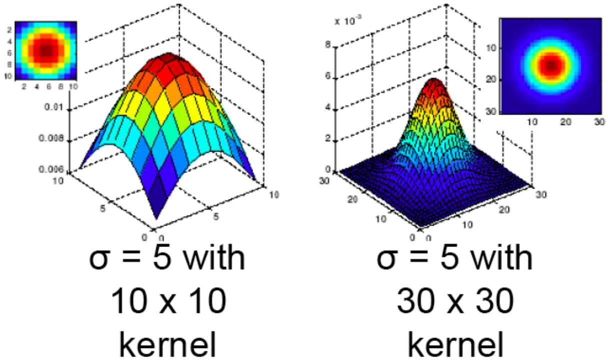
Smoothing with a Gaussian



[Source: K. Grauman]

Gaussian filter: Parameters

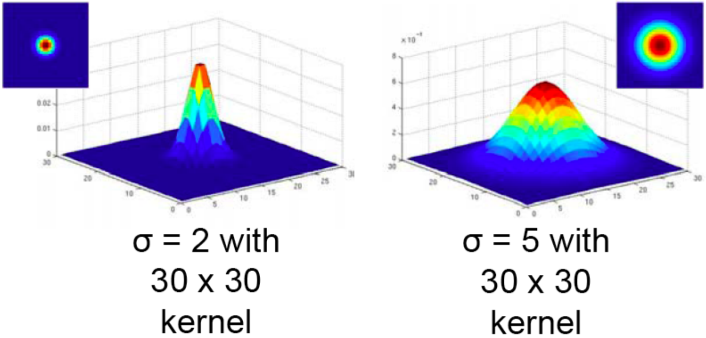
- Size of kernel or mask: Gaussian function has infinite support, but discrete filters use finite kernels.



[Source: K. Grauman]

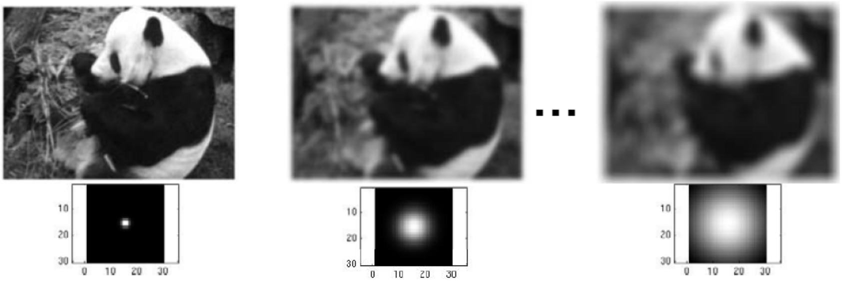
Gaussian filter: Parameters

- Variance of the Gaussian: determines extent of smoothing.



[Source: K. Grauman]

Gaussian filter: Parameters




```
for sigma=1:3:10
   h = fspecial('gaussian', fsize, sigma);
   out = imfilter(im, h);
   imshow(out);
   pause;
end
```

[Source: K. Grauman]

Properties of the Smoothing

- All values are positive.
- They all sum to 1.

Properties of the Smoothing

- All values are positive.
- They all sum to 1.
- Amount of smoothing proportional to mask size.

Properties of the Smoothing

- All values are positive.
- They all sum to 1.
- Amount of smoothing proportional to mask size.
- Remove high-frequency components; low-pass filter.

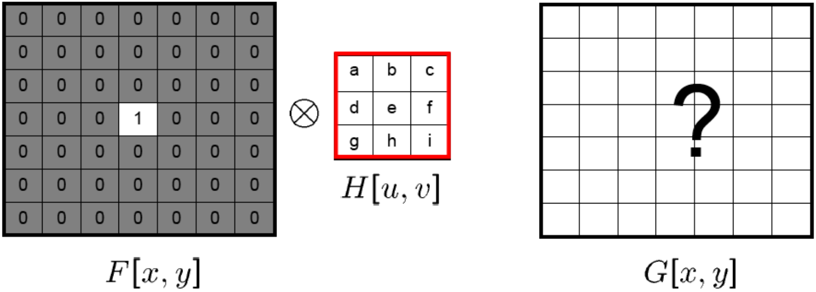
[Source: K. Grauman]

Properties of the Smoothing

- All values are positive.
- They all sum to 1.
- Amount of smoothing proportional to mask size.
- Remove high-frequency components; low-pass filter.

[Source: K. Grauman]

- What is the result of filtering the impulse signal (image) F with the arbitrary kernel H?



[Source: K. Grauman]

Convolution

- **Convolution** operator

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

  and h is then called the impulse response function.

- Equivalent to flip the filter in both dimensions (bottom to top, right to left) and apply cross-correlation.

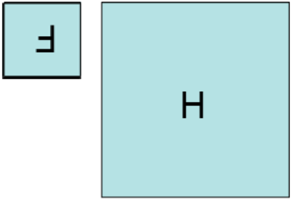
Convolution

- **Convolution** operator

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

  and h is then called the impulse response function.

- Equivalent to flip the filter in both dimensions (bottom to top, right to left) and apply cross-correlation.



Matrix form

- Correlation and convolution can both be written as a matrix-vector multiply, if we first convert the two-dimensional images $f(i,j)$ and $g(i,j)$ into raster-ordered vectors f and g

$$\mathbf{g} = \mathbf{H}\mathbf{f}$$

with **H** a sparse matrix.

$$\boxed{72}\boxed{88}\boxed{62}\boxed{52}\boxed{37} * \boxed{1/4}\boxed{1/2}\boxed{1/4} \Leftrightarrow \frac{1}{4}\begin{bmatrix} 2 & 1 & . & . & . \\ 1 & 2 & 1 & . & . \\ . & 1 & 2 & 1 & . \\ . & . & 1 & 2 & 1 \\ . & . & . & 1 & 2 \end{bmatrix}\begin{bmatrix} 72 \\ 88 \\ 62 \\ 52 \\ 37 \end{bmatrix}$$

Correlation vs Convolution

- Convolution

$$
\begin{aligned}
g(i,j) &= \sum_{k,l} f(i-k, j-l) h(k,l) \\
G &= H * F
\end{aligned}
$$

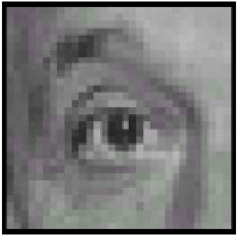
- Cross-correlation

$$
\begin{aligned}
g(i,j) &= \sum_{k,l} f(i+k, j+l) h(k,l) \\
G &= H \otimes F
\end{aligned}
$$

- For a Gaussian or box filter, how will the outputs differ?
- If the input is an impulse signal, how will the outputs differ? $h * \delta$?, and $h \otimes \delta$?

Example

- What's the result?



Original

0	0	0
0	1	0
0	0	0

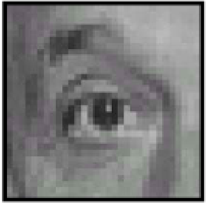
?

Example

- What's the result?

Original

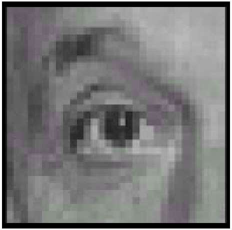
0	0	0
0	1	0
0	0	0



Filtered
(no change)

Example

- What's the result?



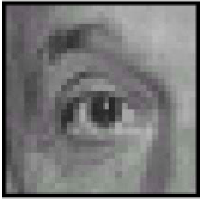
Original

0	0	0
0	0	1
0	0	0

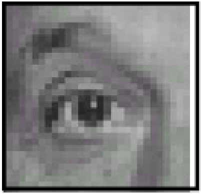
?

Example

- What's the result?



0	0	0
0	0	1
0	0	0



Correlation vs Convolution

- The convolution is both commutative and associative.
- The Fourier transform of two convolved images is the product of their individual Fourier transforms.

Correlation vs Convolution

- The convolution is both commutative and associative.
- The Fourier transform of two convolved images is the product of their individual Fourier transforms.
- Both correlation and convolution are linear shift-invariant (LSI) operators, which obey both the **superposition principle**

$$h \circ (f_0 + f_1) = h \circ f_o + h \circ f_1$$

and the **shift invariance principle**

$$\text{if} \quad g(i,j) = f(i+k, j+l) \leftrightarrow (h \circ g)(i,j) = (h \circ f)(i+k, j+l)$$

which means that shifting a signal commutes with applying the operator.

Correlation vs Convolution

- The convolution is both commutative and associative.
- The Fourier transform of two convolved images is the product of their individual Fourier transforms.
- Both correlation and convolution are linear shift-invariant (LSI) operators, which obey both the **superposition principle**

$$h \circ (f_0 + f_1) = h \circ f_o + h \circ f_1$$

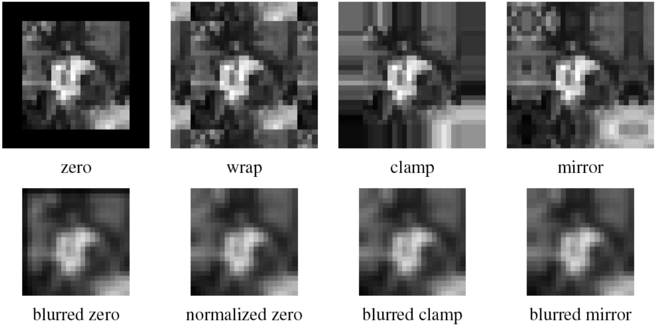
  and the **shift invariance principle**

$$\text{if} \quad g(i,j) = f(i+k, j+l) \leftrightarrow (h \circ g)(i,j) = (h \circ f)(i+k, j+l)$$

  which means that shifting a signal commutes with applying the operator.

Boundary Effects

- The results of filtering the image in this form will lead to a darkening of the corner pixels.
- The original image is effectively being padded with 0 values wherever the convolution kernel extends beyond the original image boundaries.
- A number of alternative padding or extension modes have been developed.



Separable Filters

- The process of performing a convolution requires K^2 operations per pixel, where K is the size of the convolution kernel.
- In many cases, this operation can be speed up by first performing a 1D horizontal convolution followed by a 1D vertical convolution, requiring $2K$ operations.

Separable Filters

- The process of performing a convolution requires K^2 operations per pixel, where K is the size of the convolution kernel.
- In many cases, this operation can be speed up by first performing a 1D horizontal convolution followed by a 1D vertical convolution, requiring $2K$ operations.
- If his is possible, then the convolution kernel is called **separable**.

Separable Filters

- The process of performing a convolution requires K^2 operations per pixel, where K is the size of the convolution kernel.
- In many cases, this operation can be speed up by first performing a 1D horizontal convolution followed by a 1D vertical convolution, requiring $2K$ operations.
- If his is possible, then the convolution kernel is called **separable**.
- And it is the outer product of two kernels

$$\mathbf{K} = \mathbf{v}\mathbf{h}^T$$

Separable Filters

- The process of performing a convolution requires K^2 operations per pixel, where K is the size of the convolution kernel.
- In many cases, this operation can be speed up by first performing a 1D horizontal convolution followed by a 1D vertical convolution, requiring $2K$ operations.
- If his is possible, then the convolution kernel is called **separable**.
- And it is the outer product of two kernels

$$\mathbf{K} = \mathbf{v}\mathbf{h}^T$$

Let's play a game...

Is this separable? If yes, what's the separable version?

$$\frac{1}{K^2}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & \cdots & 1 \\ \hline \vdots & \vdots & 1 & \vdots \\ \hline 1 & 1 & \cdots & 1 \\ \hline \end{array}$$

Let's play a game...

Is this separable? If yes, what's the separable version?

$$\frac{1}{K^2}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & \cdots & 1 \\ \hline \vdots & \vdots & 1 & \vdots \\ \hline 1 & 1 & \cdots & 1 \\ \hline \end{array} \qquad \frac{1}{K}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline \end{array}$$

What does this filter do?

Let's play a game...

Is this separable? If yes, what's the separable version?

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline\end{array}$$

Let's play a game...

Is this separable? If yes, what's the separable version?

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

What does this filter do?

Let's play a game...

Is this separable? If yes, what's the separable version?

$$\frac{1}{256}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 6 & 24 & 36 & 24 & 6 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}$$

Let's play a game...

Is this separable? If yes, what's the separable version?

$\frac{1}{256}$

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

$\frac{1}{16}$

1	4	6	4	1

What does this filter do?

Let's play a game...

Is this separable? If yes, what's the separable version?

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

Let's play a game...

Is this separable? If yes, what's the separable version?

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

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

What does this filter do?

Let's play a game...

Is this separable? If yes, what's the separable version?

$$\frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline -2 & 4 & -2 \\ \hline 1 & -2 & 1 \\ \hline \end{array}$$

Let's play a game...

Is this separable? If yes, what's the separable version?

$\frac{1}{4}$

1	−2	1
−2	4	−2
1	−2	1

$\frac{1}{2}$

1	−2	1

What does this filter do?

How can we tell if a given kernel K is indeed separable?

- Inspection... this is what we were doing.
- Looking at the analytic form of it.

How can we tell if a given kernel K is indeed separable?

- Inspection... this is what we were doing.
- Looking at the analytic form of it.
- Look at the singular value decomposition (SVD), and if only one singular value is non-zero, then it is separable

$$K = \mathbf{U}\Sigma\mathbf{V}^T = \sum_i \sigma_i u_i v_i^T$$

with $\Sigma = \text{diag}(\sigma_i)$.

How can we tell if a given kernel K is indeed separable?

- Inspection... this is what we were doing.
- Looking at the analytic form of it.
- Look at the singular value decomposition (SVD), and if only one singular value is non-zero, then it is separable

$$K = \mathbf{U}\Sigma\mathbf{V}^T = \sum_i \sigma_i u_i v_i^T$$

with $\Sigma = \text{diag}(\sigma_i)$.

- $\sqrt{\sigma_1}\mathbf{u}_1$ and $\sqrt{\sigma_1}\mathbf{v}_1^T$ are the vertical and horizontal kernels.

How can we tell if a given kernel K is indeed separable?

- Inspection... this is what we were doing.
- Looking at the analytic form of it.
- Look at the singular value decomposition (SVD), and if only one singular value is non-zero, then it is separable

$$K = \mathbf{U}\Sigma\mathbf{V}^T = \sum_i \sigma_i u_i v_i^T$$

with $\Sigma = \mathrm{diag}(\sigma_i)$.

- $\sqrt{\sigma_1}\mathbf{u}_1$ and $\sqrt{\sigma_1}\mathbf{v}_1^T$ are the vertical and horizontal kernels.

Filtering: Edge detection

- Map image from 2d array of pixels to a set of curves or line segments or contours.
- Look for strong gradients, post-process.

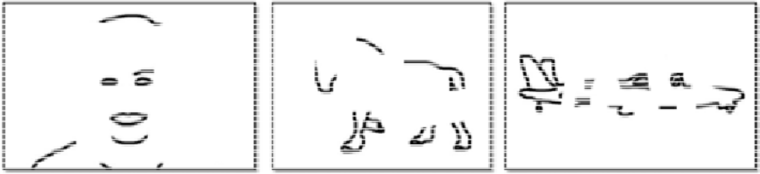


Figure: [Shotton et al. PAMI, 07]

[Source: K. Grauman]

Filtering: Edge detection

- Map image from 2d array of pixels to a set of curves or line segments or contours.
- Look for strong gradients, post-process.

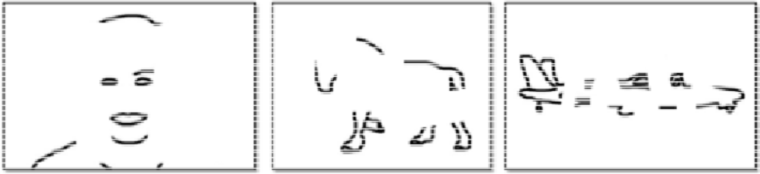
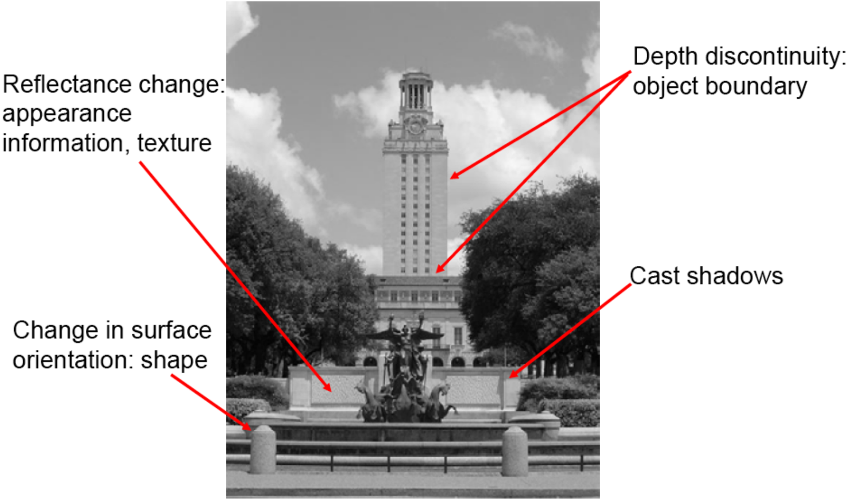


Figure: [Shotton et al. PAMI, 07]

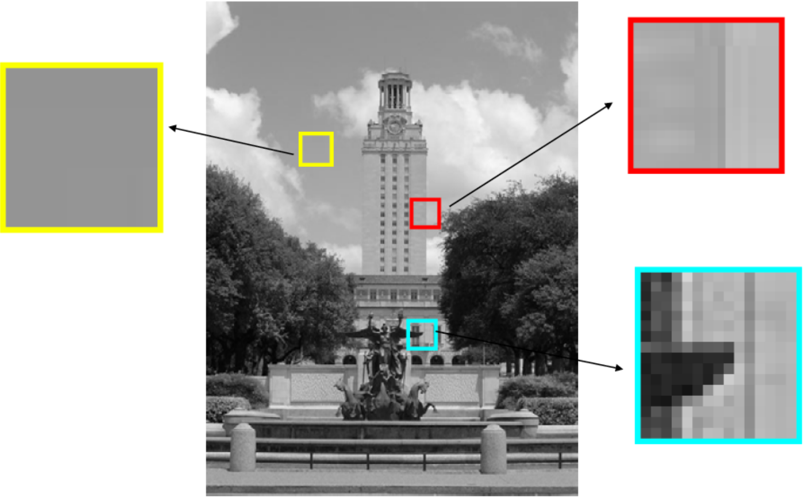
[Source: K. Grauman]

What causes an edge?



[Source: K. Grauman]

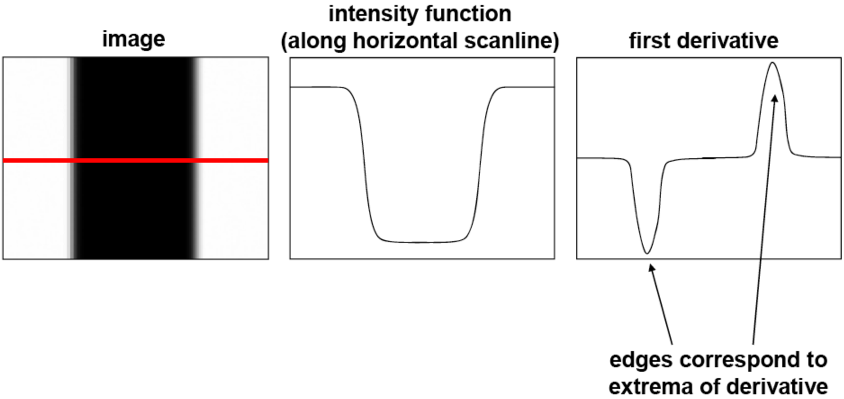
Looking more locally...



[Source: K. Grauman]

Derivatives and Edges

- An edge is a place of rapid change in the image intensity function.



[Source: S. Lazebnik]

How to Implement Derivatives with Convolution

- For 2D functions, the partial derivative is

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon, y) - f(x)}{\epsilon}$$

- We can approximate with finite differences

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x)}{1}$$

How to Implement Derivatives with Convolution

- For 2D functions, the partial derivative is

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon, y) - f(x)}{\epsilon}$$

- We can approximate with finite differences

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x)}{1}$$

- What would be the filter to implement this using convolution?

How to Implement Derivatives with Convolution

- For 2D functions, the partial derivative is

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon, y) - f(x)}{\epsilon}$$

- We can approximate with finite differences

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x)}{1}$$

- What would be the filter to implement this using convolution?

Partial derivatives of an image

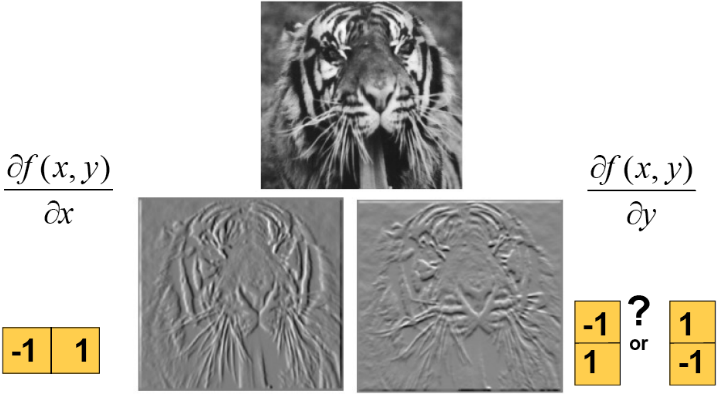


Figure: Using correlation filters

[Source: K. Grauman]

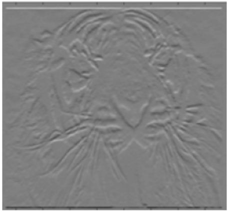
Finite Difference Filters

Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

```
>> My = fspecial('sobel');
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```



[Source: K. Grauman]

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 change in intensity

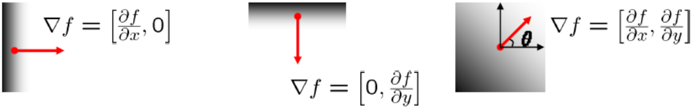
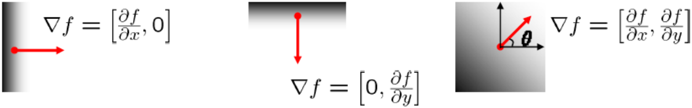


Image Gradient

- 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 change in intensity

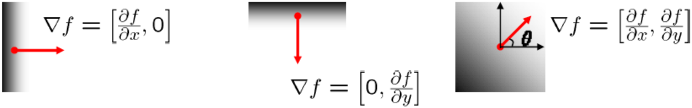


- The gradient direction (orientation of edge normal) is given by:

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

Image Gradient

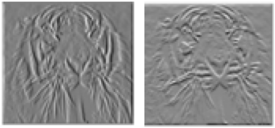
- The gradient of an image $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$
- The gradient points in the direction of most rapid change in intensity



- The gradient direction (orientation of edge normal) is given by:

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

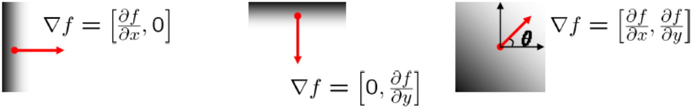
- The edge strength is given by the magnitude $||\nabla f|| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$



[Source: S. Seitz]

Image Gradient

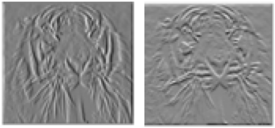
- The gradient of an image $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$
- The gradient points in the direction of most rapid change in intensity



- The gradient direction (orientation of edge normal) is given by:

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

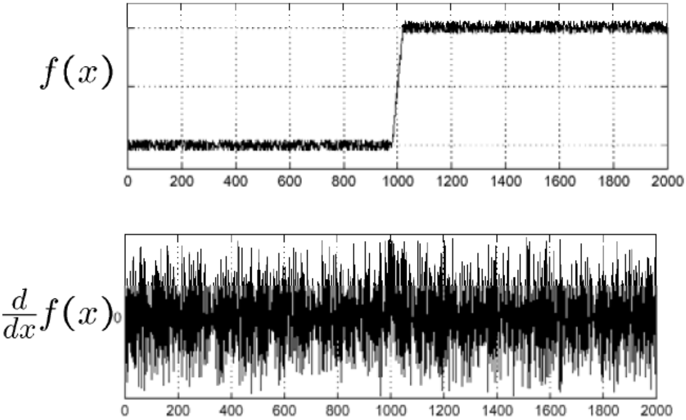
- The edge strength is given by the magnitude $||\nabla f|| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$



[Source: S. Seitz]

Effects of noise

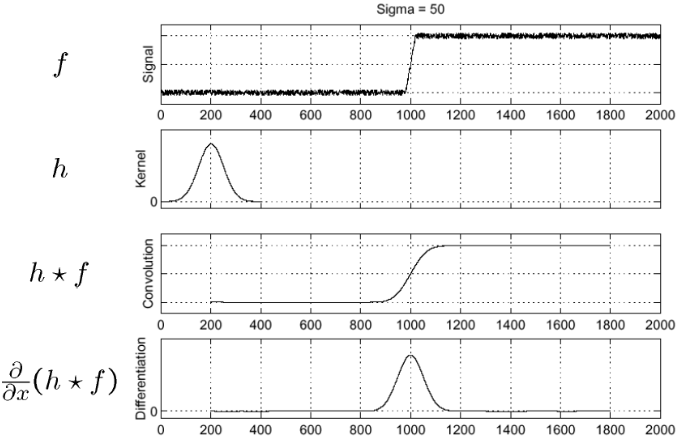
- Consider a single row or column of the image.
- Plotting intensity as a function of position gives a signal.



[Source: S. Seitz]

Effects of noise

- Smooth first, and look for picks in $\frac{\partial}{\partial x}(h * f)$.

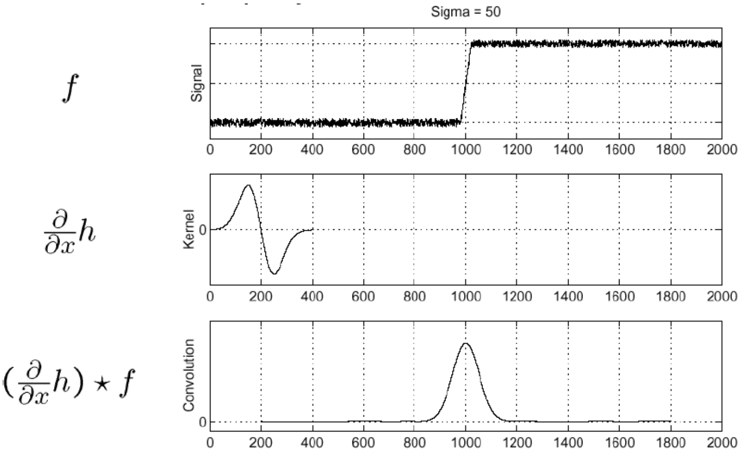


[Source: S. Seitz]

Derivative theorem of convolution

- Differentiation property of convolution

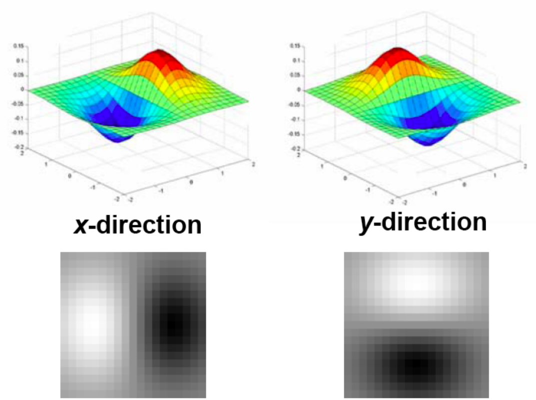
$$\frac{\partial}{\partial x}(h * f) = (\frac{\partial h}{\partial x}) * f$$



Derivative of Gaussians

- We have the following equivalence

$$(I \otimes g) \otimes h = I \otimes (g \otimes h)$$



[Source: K. Grauman]

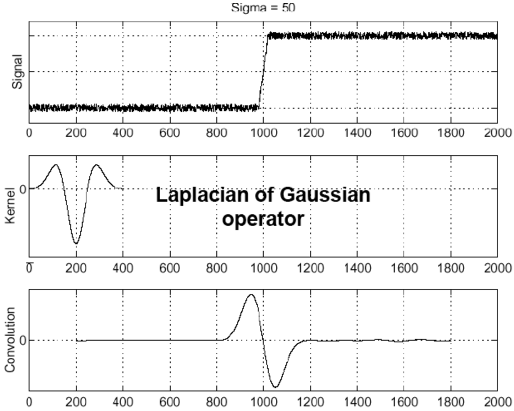
Laplacian of Gaussians

- Edge by detecting zero-crossings of bottom graph

f

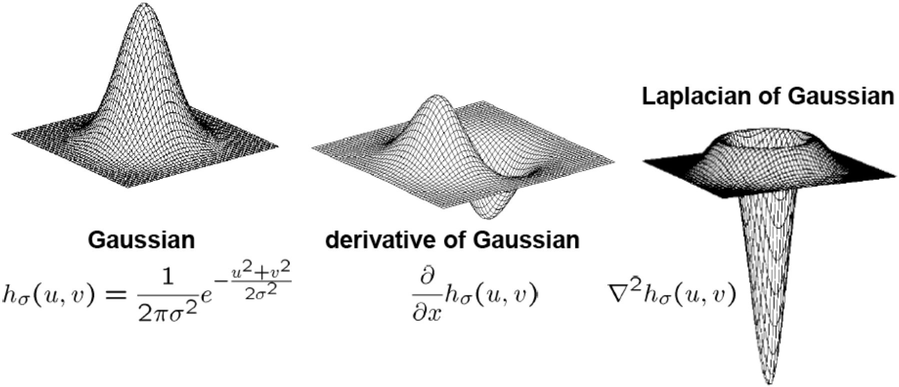
$\frac{\partial^2}{\partial x^2}h$

$(\frac{\partial^2}{\partial x^2}h) \star f$



[Source: S. Seitz]

2D Edge Filtering



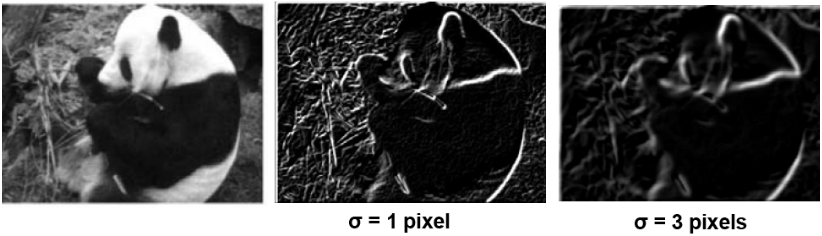
with ∇^2 the Laplacian operator $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

[Source: S. Seitz]

Effect of σ on derivatives

The detected structures differ depending on the Gaussian's scale parameter:

- Larger values: larger scale edges detected.
- Smaller values: finer features detected.



[Source: K. Grauman]

Derivatives

- Use opposite signs to get response in regions of high contrast.
- They sum to 0 so that there is no response in constant regions.
- High absolute value at points of high contrast.

[Source: K. Grauman]

Band-pass filters

- The Sobel and corner filters are band-pass and oriented filters.
- More sophisticated filters can be obtained by convolving with a Gaussian filter

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

  and taking the first or second derivatives.

Band-pass filters

- The Sobel and corner filters are band-pass and oriented filters.
- More sophisticated filters can be obtained by convolving with a Gaussian filter

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

and taking the first or second derivatives.

- These filters are **band-pass filters**: they filter low and high frequencies.

Band-pass filters

- The Sobel and corner filters are band-pass and oriented filters.
- More sophisticated filters can be obtained by convolving with a Gaussian filter
$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$
and taking the first or second derivatives.
- These filters are **band-pass filters**: they filter low and high frequencies.
- The second derivative of a two-dimensional image is the **laplacian** operator
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Band-pass filters

- The Sobel and corner filters are band-pass and oriented filters.
- More sophisticated filters can be obtained by convolving with a Gaussian filter

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

  and taking the first or second derivatives.
- These filters are **band-pass filters**: they filter low and high frequencies.
- The second derivative of a two-dimensional image is the **laplacian** operator

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

- Blurring an image with a Gaussian and then taking its Laplacian is equivalent to convolving directly with the **Laplacian of Gaussian** (LoG) filter,

$$\nabla^2 f G(x, y, \sigma) = \left(\frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2}\right) G(x, y, \sigma)$$

Band-pass filters

- The Sobel and corner filters are band-pass and oriented filters.
- More sophisticated filters can be obtained by convolving with a Gaussian filter

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

  and taking the first or second derivatives.
- These filters are **band-pass filters**: they filter low and high frequencies.
- The second derivative of a two-dimensional image is the **laplacian** operator

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

- Blurring an image with a Gaussian and then taking its Laplacian is equivalent to convolving directly with the **Laplacian of Gaussian** (LoG) filter,

$$\nabla^2 fG(x, y, \sigma) = \left(\frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2}\right) G(x, y, \sigma)$$

Band-pass filters

- The **directional or oriented filter** can obtained by smoothing with a Gaussian (or some other filter) and then taking a directional derivative $\nabla_{\mathbf{u}} = \frac{\partial}{\partial \mathbf{u}}$

$$\mathbf{u} \cdot \nabla(G * f) = \nabla_{\mathbf{u}}(G * f) = (\nabla_{\mathbf{u}} G) * f$$

  with $\mathbf{u} = (\cos\theta, \sin\theta)$.

- The Sobel operator is a simple approximation of this:

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

Steerable Filters

- Oriented filters are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.
- One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.

Steerable Filters

- Oriented filters are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.
- One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.
- More efficient is to apply a few filters corresponding to a few angles and interpolate between the responses.

Steerable Filters

- Oriented filters are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.
- One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.
- More efficient is to apply a few filters corresponding to a few angles and interpolate between the responses.
- One then needs to know how many filters are required and how to properly interpolate between the responses.

Steerable Filters

- Oriented filters are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.
- One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.
- More efficient is to apply a few filters corresponding to a few angles and interpolate between the responses.
- One then needs to know how many filters are required and how to properly interpolate between the responses.
- With the correct filter set and the correct interpolation rule, it is possible to determine the response of a filter of arbitrary orientation without explicitly applying that filter.

Steerable Filters

- Oriented filters are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.
- One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.
- More efficient is to apply a few filters corresponding to a few angles and interpolate between the responses.
- One then needs to know how many filters are required and how to properly interpolate between the responses.
- With the correct filter set and the correct interpolation rule, it is possible to determine the response of a filter of arbitrary orientation without explicitly applying that filter.
- **Steerable filters** are a class of filters in which a filter of arbitrary orientation is synthesized as a linear combination of a set of basis filters.

Steerable Filters

- Oriented filters are used in many vision and image processing tasks: texture analysis, edge detection, image data compression, motion analysis.
- One approach to finding the response of a filter at many orientations is to apply many versions of the same filter, each different from the others by some small rotation in angle.
- More efficient is to apply a few filters corresponding to a few angles and interpolate between the responses.
- One then needs to know how many filters are required and how to properly interpolate between the responses.
- With the correct filter set and the correct interpolation rule, it is possible to determine the response of a filter of arbitrary orientation without explicitly applying that filter.
- **Steerable filters** are a class of filters in which a filter of arbitrary orientation is synthesized as a linear combination of a set of basis filters.

Example of Steerable Filter

- 2D symmetric Gaussian with $\sigma = 1$ and assume constant is 1

$$G(x, y, \sigma) = \exp\left(-x^2 + y^2\right)$$

- The directional derivative operator is steerable.

Example of Steerable Filter

- 2D symmetric Gaussian with $\sigma = 1$ and assume constant is 1

$$G(x, y, \sigma) = \exp\left(-x^2 + y^2\right)$$

- The directional derivative operator is steerable.
- The first derivative

$$G_1^0 = \frac{\partial}{\partial x} \exp\left(-x^2 + y^2\right) = -2x \exp\left(-x^2 + y^2\right)$$

and the same function rotated 90 degrees is

$$G_1^{90} = \frac{\partial}{\partial y} \exp\left(-x^2 + y^2\right) = -2y \exp\left(-x^2 + y^2\right)$$

Example of Steerable Filter

- 2D symmetric Gaussian with $\sigma = 1$ and assume constant is 1

$$G(x, y, \sigma) = \exp\left(-x^2 + y^2\right)$$

- The directional derivative operator is steerable.
- The first derivative

$$G_1^0 = \frac{\partial}{\partial x} \exp\left(-x^2 + y^2\right) = -2x \exp\left(-x^2 + y^2\right)$$

and the same function rotated 90 degrees is

$$G_1^{90} = \frac{\partial}{\partial y} \exp\left(-x^2 + y^2\right) = -2y \exp\left(-x^2 + y^2\right)$$

- A filter of arbitrary orientation θ can be synthesized by taking a linear combination of G_1^0 and G_1^{90}

$$G_1^\theta = \cos\theta G_1^0 + \sin\theta G_1^{90}$$

G_1^0 and G_1^{90} are the **basis filters** and $\cos\theta$ and $\sin\theta$ are the **interpolation functions**

Example of Steerable Filter

- 2D symmetric Gaussian with $\sigma = 1$ and assume constant is 1

$$G(x, y, \sigma) = \exp\left(-x^2 + y^2\right)$$

- The directional derivative operator is steerable.
- The first derivative

$$G_1^0 = \frac{\partial}{\partial x} \exp\left(-x^2 + y^2\right) = -2x \exp\left(-x^2 + y^2\right)$$

and the same function rotated 90 degrees is

$$G_1^{90} = \frac{\partial}{\partial y} \exp\left(-x^2 + y^2\right) = -2y \exp\left(-x^2 + y^2\right)$$

- A filter of arbitrary orientation θ can be synthesized by taking a linear combination of G_1^0 and G_1^{90}

$$G_1^\theta = \cos\theta G_1^0 + \sin\theta G_1^{90}$$

G_1^0 and G_1^{90} are the **basis filters** and $\cos\theta$ and $\sin\theta$ are the **interpolation functions**

More on steerable filters

- Because convolution is a linear operation, we can synthesize an image filtered at an arbitrary orientation by taking linear combinations of the images filtered with G_1^0 and G_1^{90}

$$\text{if} \quad R_1^0 = G_1^0 * I \quad \text{and} \quad R_1^{90} = G_1^{90} * I \quad \text{then} \quad R_1^\theta = \cos\theta R_1^0 + \sin\theta R_1^{90}$$

- Check yourself that this is the case.

More on steerable filters

- Because convolution is a linear operation, we can synthesize an image filtered at an arbitrary orientation by taking linear combinations of the images filtered with G_1^0 and G_1^{90}

$$\text{if} \quad R_1^0 = G_1^0 * I \quad \text{and} \quad R_1^{90} = G_1^{90} * I \quad \text{then} \quad R_1^\theta = \cos\theta R_1^0 + \sin\theta R_1^{90}$$

- Check yourself that this is the case.
- See [Freeman & Adelson, 91] for the conditions on when a filter is steerable and how many basis are necessary.

More on steerable filters

- Because convolution is a linear operation, we can synthesize an image filtered at an arbitrary orientation by taking linear combinations of the images filtered with G_1^0 and G_1^{90}

$$\text{if} \quad R_1^0 = G_1^0 * I \quad \text{and} \quad R_1^{90} = G_1^{90} * I \quad \text{then} \quad R_1^\theta = \cos\theta R_1^0 + \sin\theta R_1^{90}$$

- Check yourself that this is the case.
- See [Freeman & Adelson, 91] for the conditions on when a filter is steerable and how many basis are necessary.

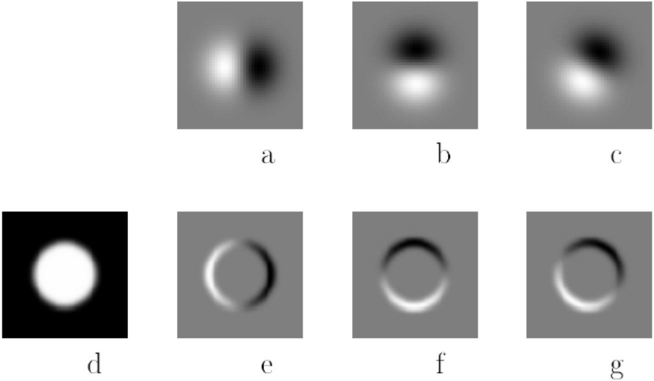
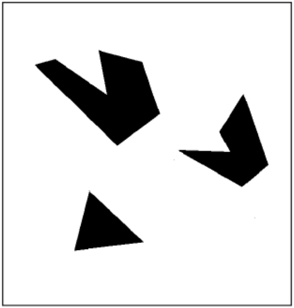


Figure 2-1: Example of steerable filters. (a) $G_1^{0°}$, first derivative with respect to x (horizontal) of a Gaussian. (b) $G_1^{90°}$, which is $G_1^{0°}$, rotated by 90°. From a linear combination of these two filters, one can create G_1^{θ}, an arbitrary rotation of the first derivative of a Gaussian. (c) $G_1^{30°}$, formed by $\frac{1}{2}G_1^{0°} + \frac{\sqrt{3}}{2}G_1^{90°}$. The same linear combinations used to synthesize G_1^{θ} from the basis filters will also synthesize the response of an image to G_1^{θ} from the responses of the image to the basis filters: (d) Image of circular disk. (e) $G_1^{0°}$ (at a smaller scale than pictured above) convolved with the disk, (d). (f) $G_1^{90°}$ convolved with (d). (g) $G_1^{30°}$ convolved with (d), obtained from $\frac{1}{2}$ [image e] $+\frac{\sqrt{3}}{2}$ [image f].

[Source: W. Freeman 91]

Template matching

- Filters as templates: filters look like the effects they are intended to find.
- Use normalized cross-correlation score to find a given pattern (template) in the image.
- Normalization needed to control for relative brightnesses.

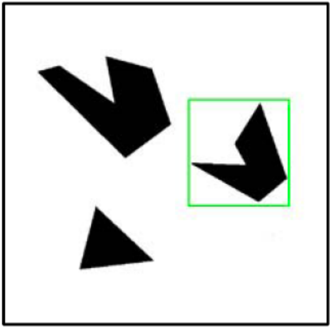


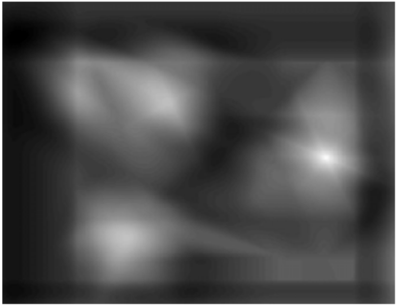
Scene

Template (mask)

[Source: K. Grauman]

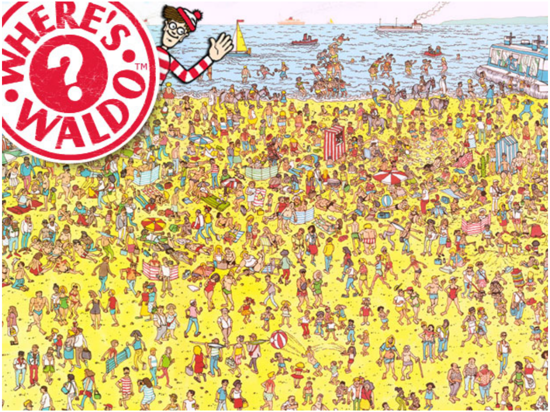
Template matching





[Source: K. Grauman]

More complex Scenes



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]

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

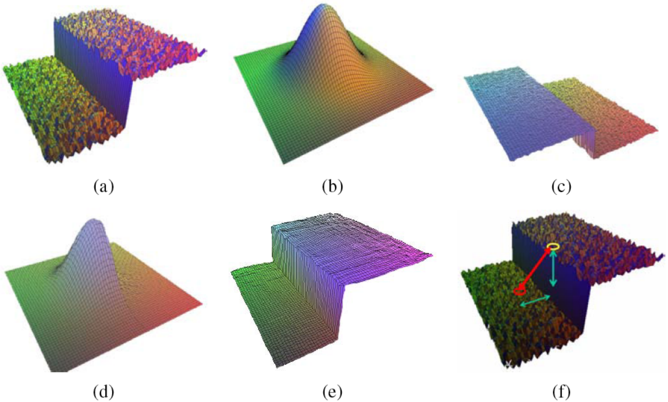


Figure: Bilateral filtering [Durand & Dorsey, 02]. (a) noisy step edge input. (b) domain filter (Gaussian). (c) range filter (similarity to center pixel value). (d) bilateral filter. (e) filtered step edge output. (f) 3D distance between pixels

[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.

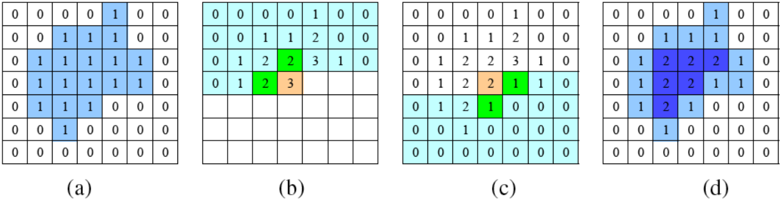


Figure: City block distance transform: (a) original binary image; (b) top to bottom (forward) raster sweep: green values are used to compute the orange value; (c) bottom to top (backward) raster sweep: green values are merged with old orange value; (d) final distance transform.

[Source: R. Szeliski]

- 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.

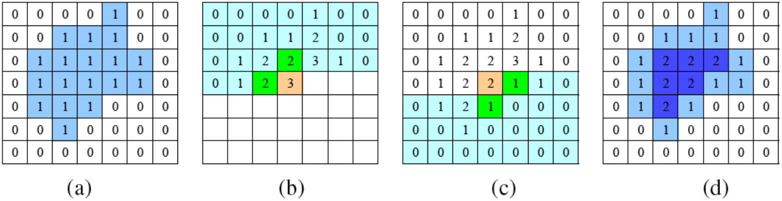
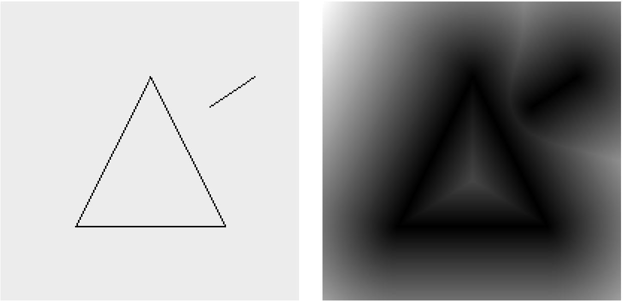


Figure: City block distance transform: (a) original binary image; (b) top to bottom (forward) raster sweep: green values are used to compute the orange value; (c) bottom to top (backward) raster sweep: green values are merged with old orange value; (d) final distance transform.

[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

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

  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

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

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**

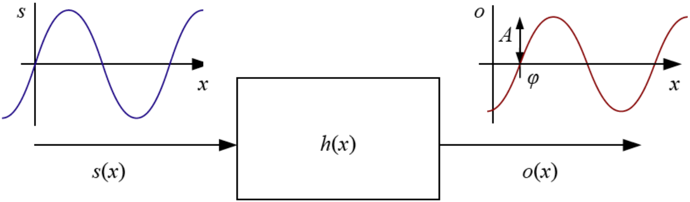


Figure 3.24 The Fourier Transform as the response of a filter $h(x)$ to an input sinusoid $s(x) = e^{j\omega x}$ yielding an output sinusoid $o(x) = h(x) * s(x) = Ae^{j\omega x+\phi}$.

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

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

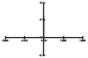
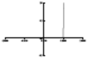
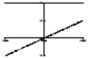
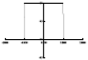
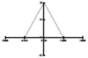
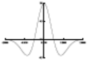
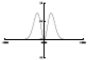
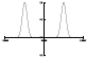
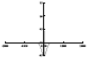
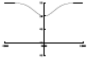
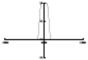
  where N is the length of the signal.

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

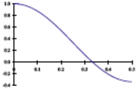
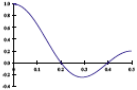
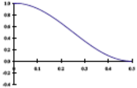
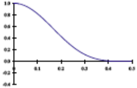
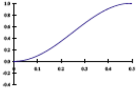
Properties Fourier Transform

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		$\text{box}(x/a)$	$\Leftrightarrow$	$a\text{sinc}(a\omega)$	
tent		$\text{tent}(x/a)$	$\Leftrightarrow$	$a\text{sinc}^2(a\omega)$	
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		$\text{rcos}(x/(aW))$ $\text{sinc}(x/a)$	$\Leftrightarrow$	(see Figure 3.29)	

[Source: R. Szeliski]

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

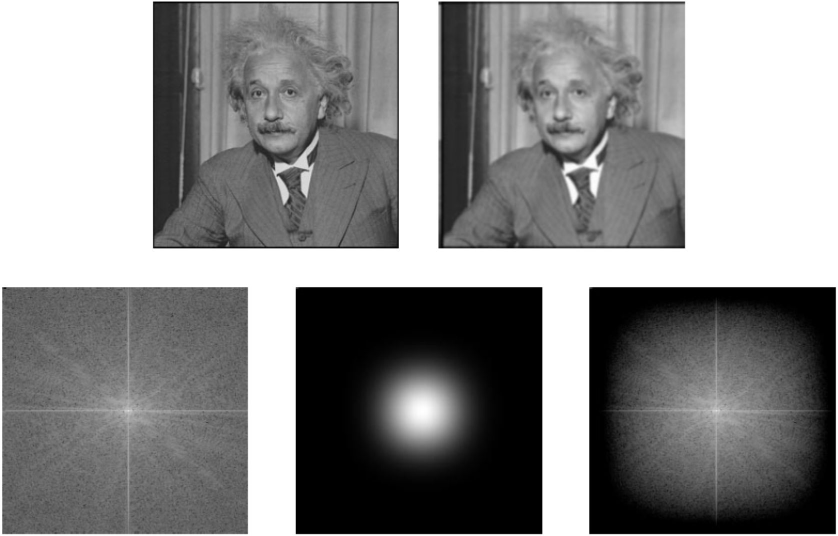
$$H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j\omega_x x + \omega_y y} dxdy$$

  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.

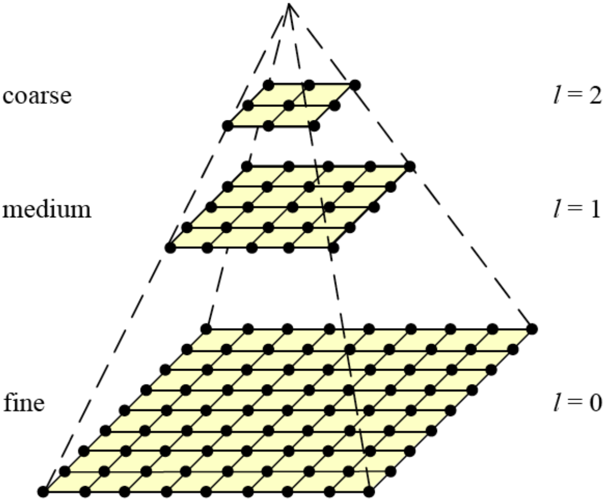
Pyramids

- 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 and Decimation

- 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.

Interpolation and Decimation

- 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.

Interpolation and Decimation

- 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.
- **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.

Interpolation and Decimation

- 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.
- **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.

Interpolation and Decimation

- 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.
- **Decimation**: reduces resolution

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

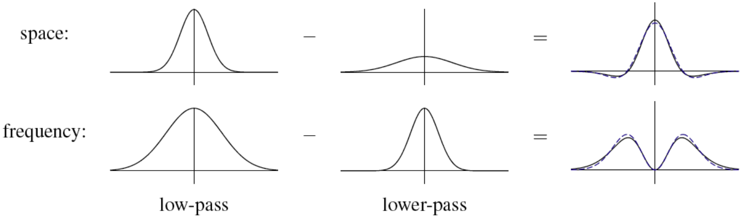
with r the down-sampling rate.

- 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.
- Wavelets are alternative pyramids. We will not see them here.



[Source: R. Szeliski]

Next class ... some image features