3.1 Image Processing: Filtering

Image Processing vs Computer Vision

- What is the difference between image processing and computer vision?
- Image processing maps an image to a different version of the image.
- Computer vision maps one or more images to inferences about the visual scene.
- Image processing operations often required as pre-processing for computer vision algorithms.

Outline

- Point Operators
- Linear Filters
- Nonlinear Filters

Outline

- **Point Operators**
- Linear Filters
- Nonlinear Filters

Point Operators

- Image processing point operators transform each pixel independently of other pixels.

$$g(\boldsymbol{x}) = h(f(\boldsymbol{x}))$$

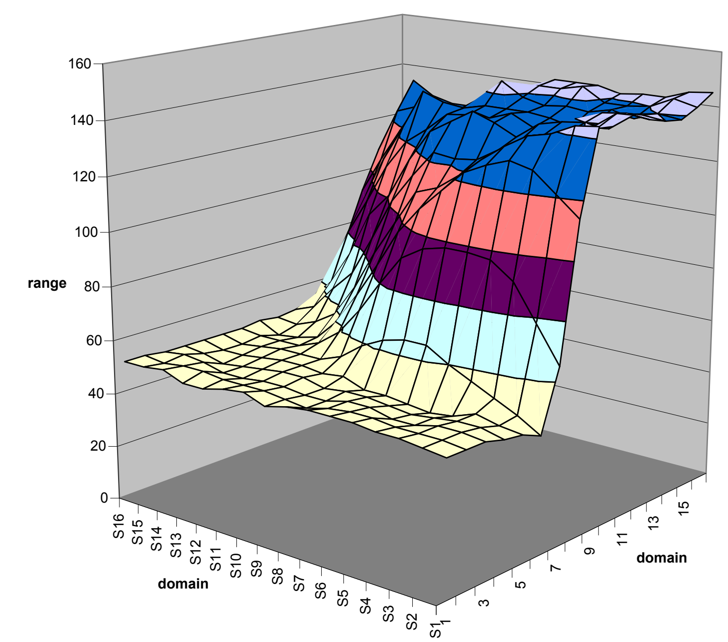
- or

col idx   row idx

$$g(i,j) = h(f(i,j))$$

output image   point operator   input image

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



Examples

- Contrast/brightness adjustment:

$$g(\boldsymbol{x}) = af(\boldsymbol{x}) + b$$

Gain (contrast) Bias (brightness)

- Inverse gamma - undo compressive gamma mapping applied in sensor so that pixel intensities are (approximately) proportional to the light irradiance at the sensor:

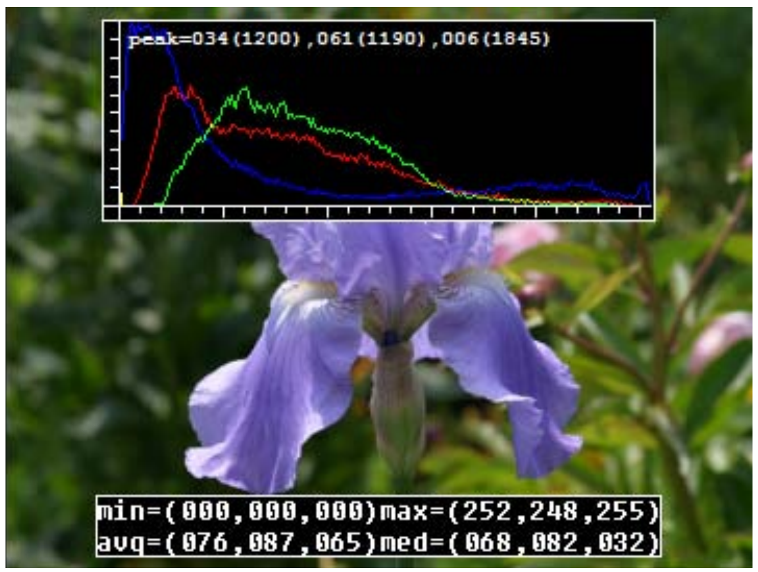
$$g(x) = x^{\gamma}$$

(note that textbook Eqn. 3.7 has this backwards)

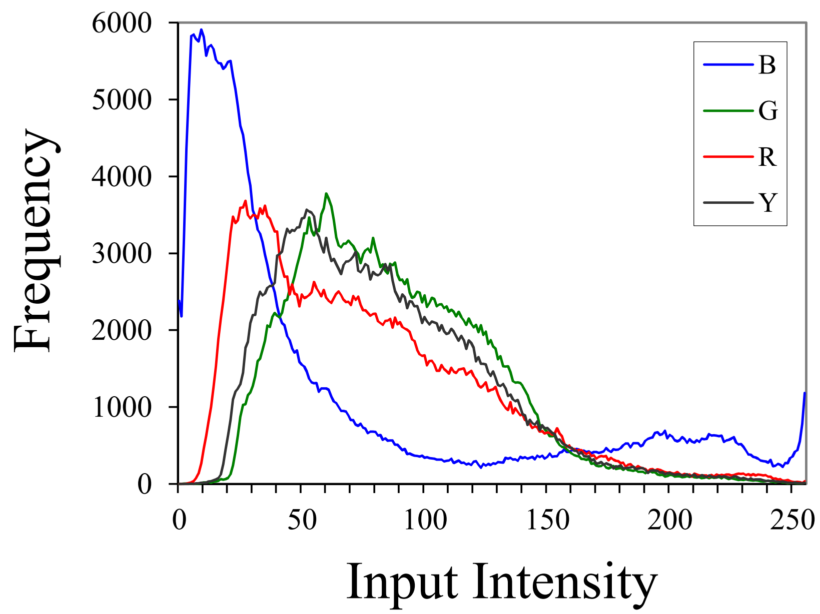
Histogram Equalization

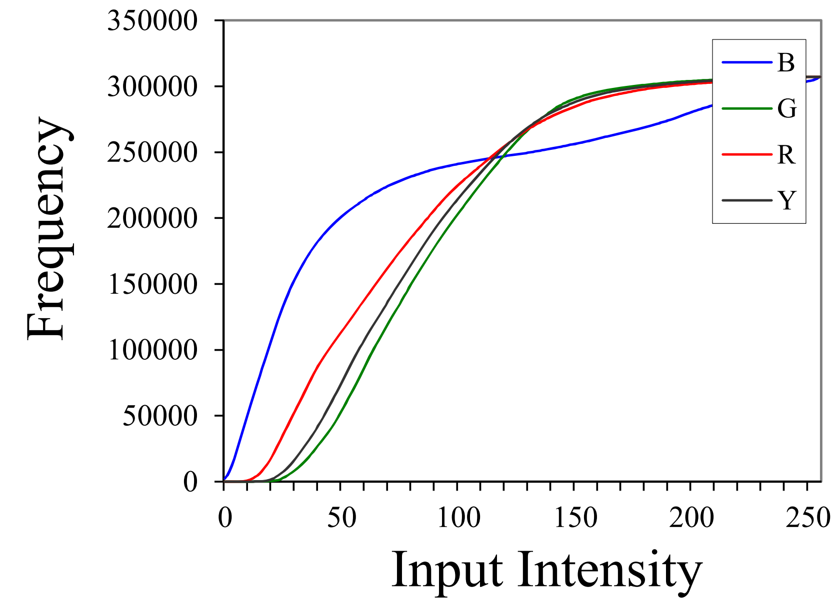
- The colours in most images are not uniformly distributed across the gamut.
- Redistribution of these colours to be uniform is called histogram equalization.

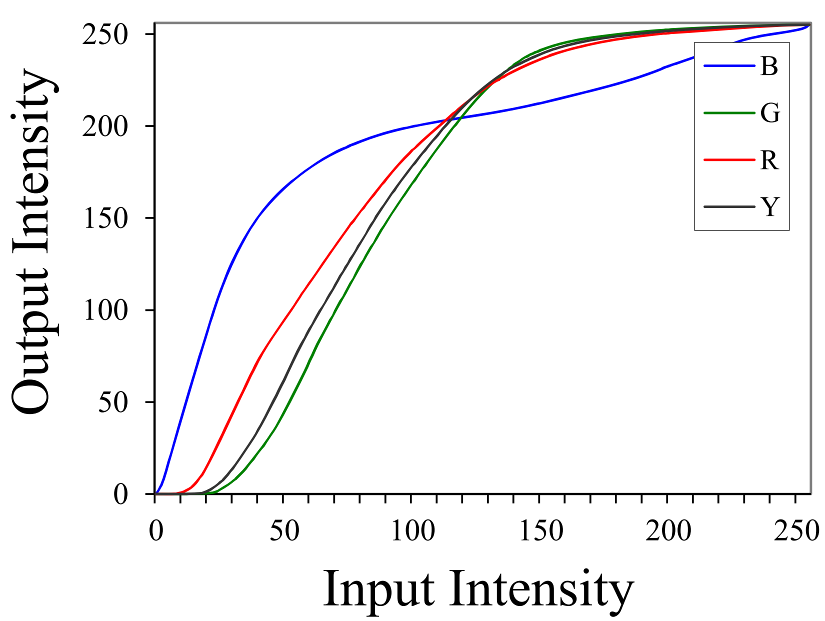
$$c(I) = \frac{1}{N}\sum_{i=0}^{I} h(i) = c(I-1) + \frac{1}{N}h(I)$$

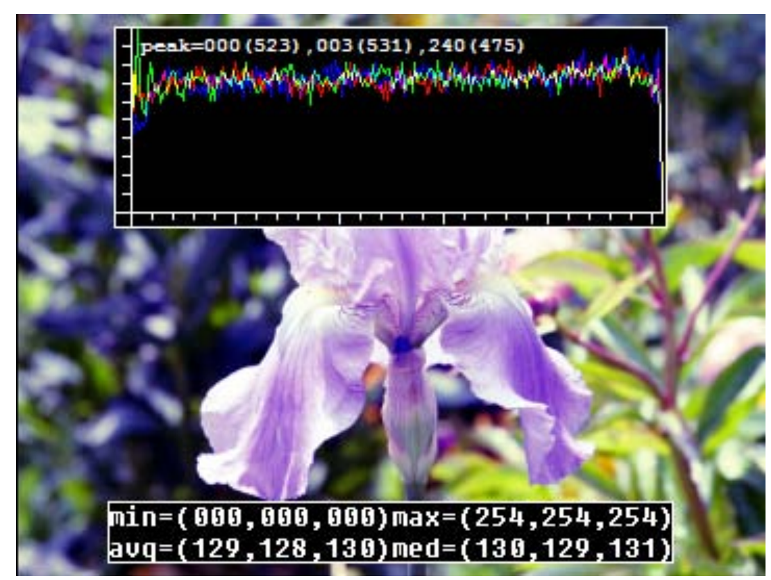


Input Image









Output Image

End of Lecture
Sept 24, 2018

Outline

- Point Operators
- **Linear Filters**
- Nonlinear Filters

Linear Filters

- Many image processing operations involve linear combinations of the pixels within a finite neighbourhood of a pixel.
- Typically, the same set of weights is applied at each pixel.
- The pattern of weights is called a *linear filter*.
- When applied at all locations in the image, this can be expressed as a correlation:

$$g(i,j) = \sum_{k,l} f(i+k, j+l)\underbrace{h(k,l)}_{\text{Linear filter}}$$

or

$$g = f \otimes h$$

MATLAB function
xcorr

- or alternatively as a convolution

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

or

$$g = f * h,$$

Impulse response function: $h * \delta = h,$

MATLAB functions
conv, conv2, convn

Linear Shift Invariant Operators

- Both correlation and convolution are linear shift invariant operators, which obey

  - Superposition

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

  - Shift invariance

    $$g(i,j) = f(i+k, j+l) \;\; \Leftrightarrow \;\; (h \circ g)(i,j) = (h \circ f)(i+k, j+l)$$

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 $\boldsymbol{f}$ and $\boldsymbol{g}$,

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

where the (sparse) $\boldsymbol{H}$ matrix contains the convolution kernels.

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

Handling Borders

- What do we do near the border of the image, where the kernel (filter) ‘falls off’ the edge?

kernel h

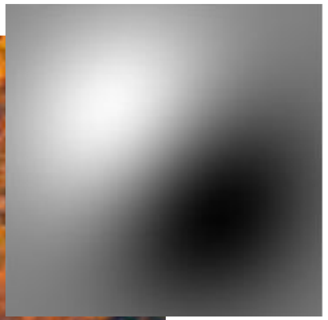


Image f

Handling Borders

- Padding options
  - Zero-padding - ignore kernel weights that fall outside image
  - Clamp - extend boundary values of image
  - Cyclic - toroidally wrap around
  - Mirror - reflect pixels across image edge
- Alternatively, we can crop the image and return only the ‘valid’ portion
  - e.g., MATLAB conv2(...,shape) returns a subsection of the two-dimensional convolution, as specified by the shape parameter:
    - ‘full' Returns the full two-dimensional convolution (default).
    - ‘same' Returns the central part of the convolution of the same size as A.
    - ‘valid' Returns only those parts of the convolution for which the kernel lies entirely within the image.

Separable Filters

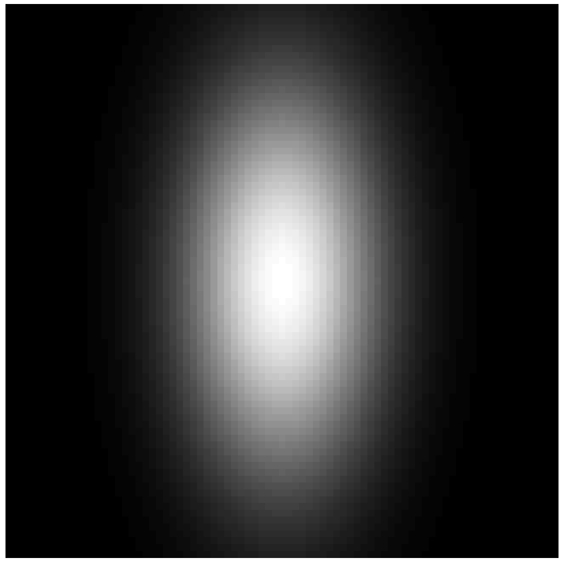
- Given a general 2D kernel of size (m, n) pixels, application at each pixel of the image involves $m*n$ multiplies.

- For an $M*N$ image, the total number of multiplies for the convolution is $M*N*m*n$.

- However, certain special 2D kernels can be decomposed into 2 1D kernels, reducing the number of multiples at a pixel to $m + n$.

- Example: 2D axis-aligned Gaussian kernel

$$h(x,y) = \frac{1}{2\pi\sigma_x\sigma_y}\exp\left(-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)\right) = \left(\frac{1}{\sqrt{2\pi}\sigma_x}\exp\left(-\frac{x^2}{2\sigma_x^2}\right)\right)\left(\frac{1}{\sqrt{2\pi}\sigma_x}\exp\left(-\frac{y^2}{2\sigma_y^2}\right)\right)$$



MATLAB function
conv2(h1, h2, A)

Example Separable Filters

$$\frac{1}{K^2}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & \cdots & 1 \\ \hline \vdots & \vdots & 1 & \vdots \\ \hline 1 & 1 & \cdots & 1 \\ \hline \end{array} \qquad \frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \qquad \frac{1}{256}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 6 & 24 & 36 & 24 & 6 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array} \qquad \frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad \frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline -2 & 4 & -2 \\ \hline 1 & -2 & 1 \\ \hline \end{array}$$

$$\frac{1}{K}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline \end{array} \qquad \frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array} \qquad \frac{1}{16}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array} \qquad \frac{1}{2}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline \end{array} \qquad \frac{1}{2}\begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline \end{array}$$





(a) box, $K = 5$ (b) bilinear (c) "Gaussian" (d) Sobel (e) corner

$\underbrace{\text{(a)–(c)}}_{\text{Smoothing}}$

Smoothing

Edge detection

Gaussian Derivatives

- Local difference filters like the Sobel filter estimate local intensity gradients.

- But the restriction to a 3x3 neighbourhood of the image makes the results noisy.

- A more general and smooth family of filters are the Gaussian derivatives, which can be derived by taking partial spatial derivatives of the 2D Gaussian function

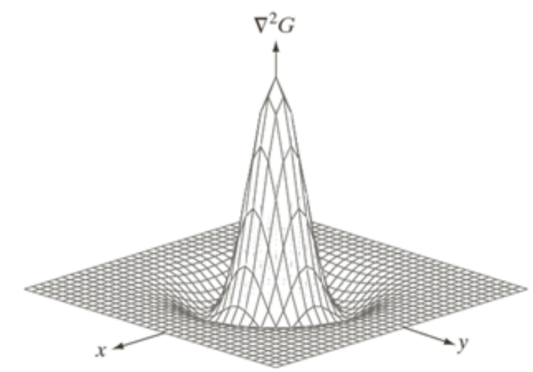
$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

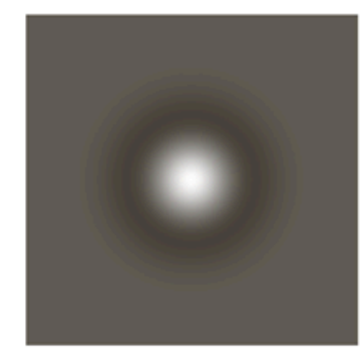
- Example: Laplacian of Gaussian (LoG):

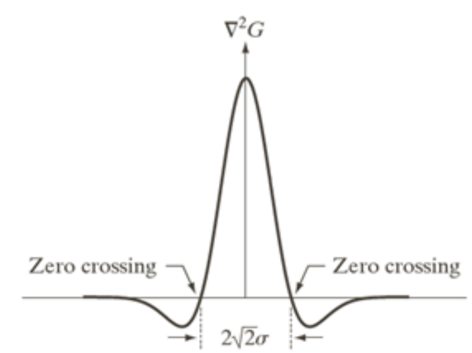
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 y}{\partial y^2}$$

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

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







0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

$\boxed{\text{MATLAB function mvnpdf}}$

Steerable Filters

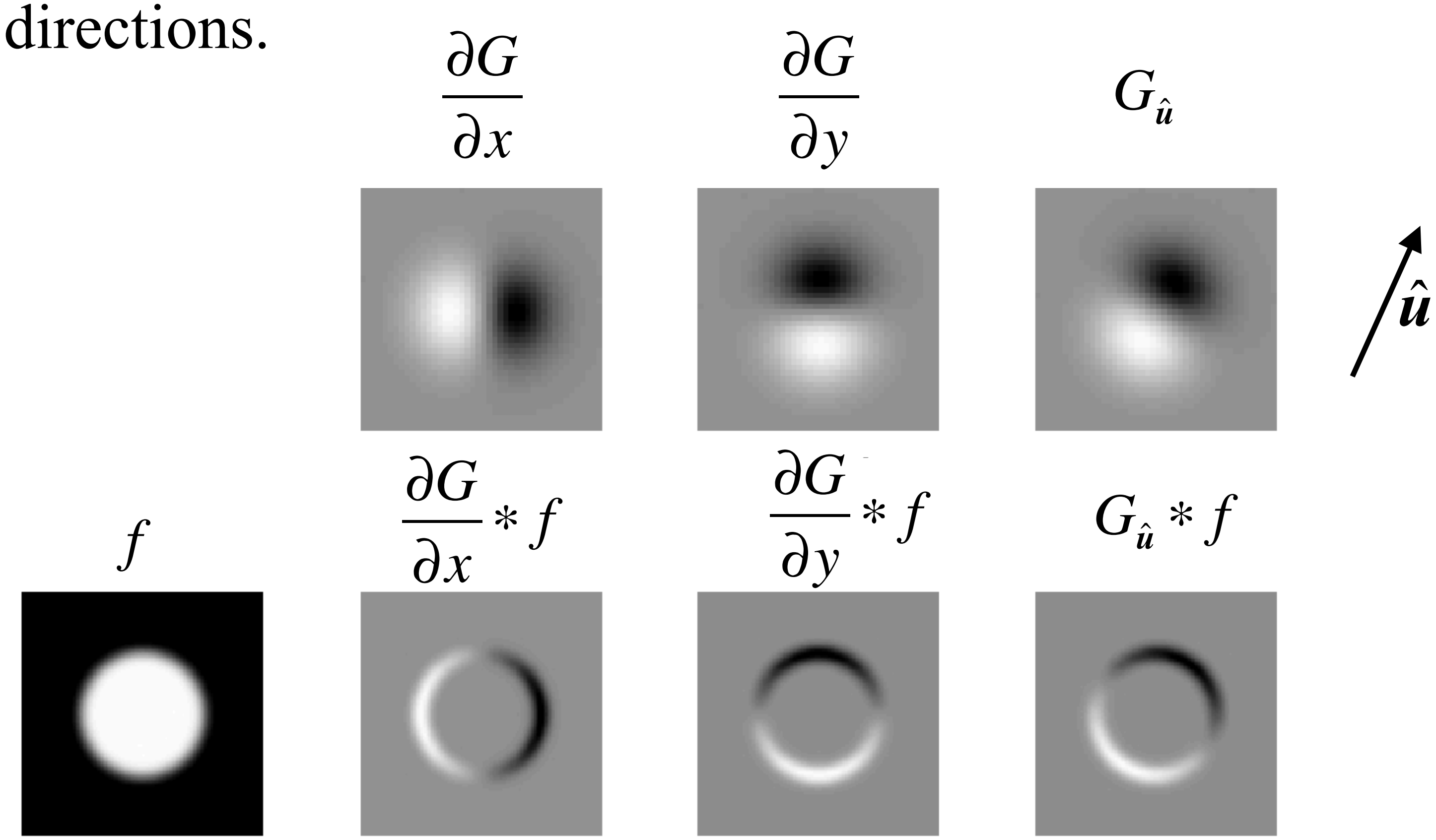
- To detect contours in the image, we typically use oriented Gaussian derivative filters, formed by taking directional derivatives of the Gaussian function:

$$\hat{\boldsymbol{u}} \cdot \nabla (G * f) = \nabla_{\hat{\boldsymbol{u}}} (G * f) = (\nabla_{\hat{\boldsymbol{u}}} G) * f.$$

- Note that

$$G_{\hat{\boldsymbol{u}}} = uG_x + vG_y = u\frac{\partial G}{\partial x} + v\frac{\partial G}{\partial y} \quad = \cos\theta \frac{\partial G}{\partial x} + \sin\theta \frac{\partial G}{\partial y} \quad \text{where } \hat{\boldsymbol{u}} = (u,v) = (\cos\theta, \sin\theta)$$

- In other words, the Gaussian derivative filter in direction $\boldsymbol{u}$ is a weighted sum of the Gaussian derivatives in x and y directions.



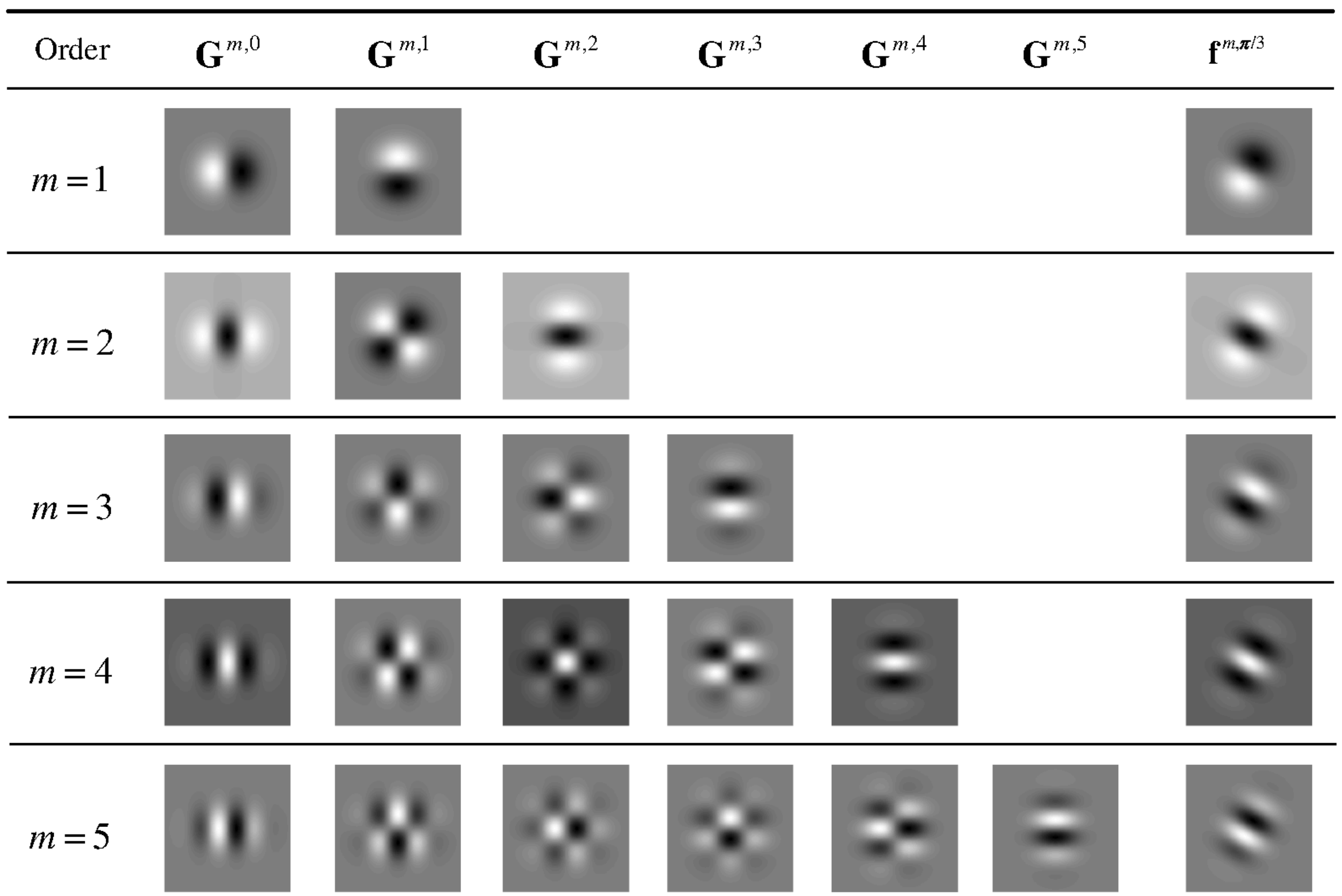
What filters are steerable?

- It turns out that Gaussian derivatives of all orders are steerable with a finite number of basis functions.

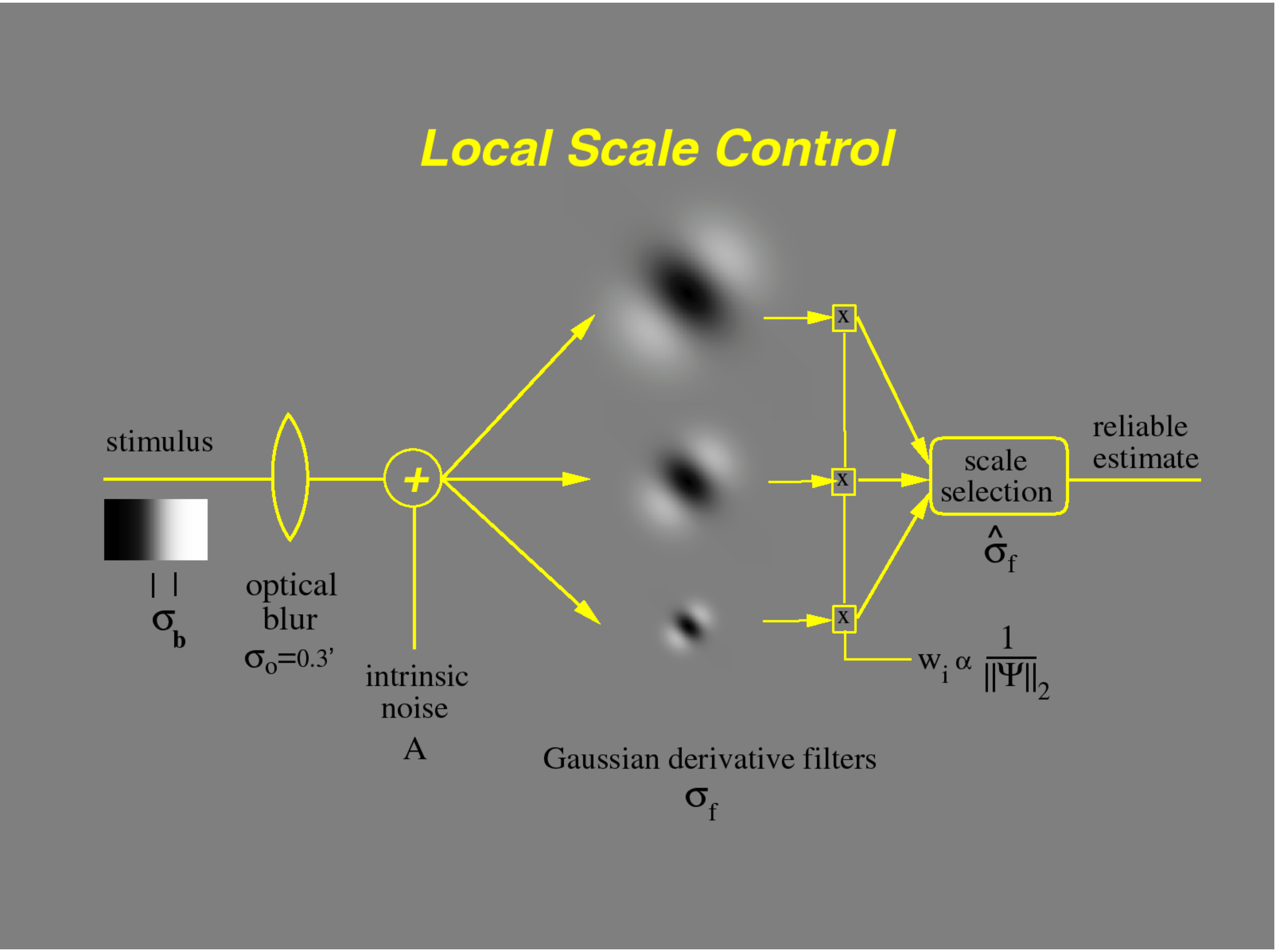
- For example, a Gaussian 2nd derivative requires 3 basis functions:

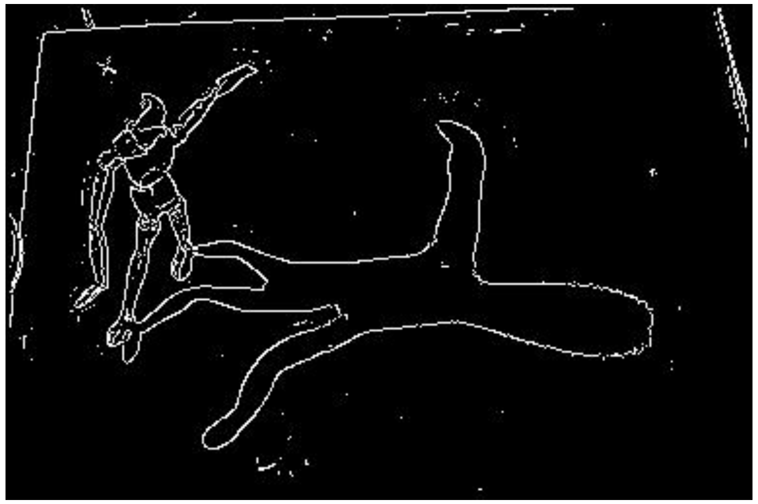
$$G_{\hat{\mathbf{u}}\hat{\mathbf{u}}} = u^2 G_{xx} + 2uvG_{xy} + v^2 G_{yy}$$

- Moreover, the basis functions are separable (or superpositions of separable functions).



Application: Edge Detection





Elder & Zucker 1998

End of Lecture
Sept 26, 2018

Integral Images

- If a diversity of box filters are to be employed, it can be very efficient to derive these from the integral image $s(i, j)$, which is the 2D analog of a 1D cumulative sum:

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

- This is efficiently computed using a raster-scan algorithm:

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

- Now, for example, a rectangular box average of arbitrary size and shape can be computed using just 4 additions/subtractions on the integral image:

$$S(i_0 \ldots i_1, j_0 \ldots j_1) = \quad s(i_1,j_1) - s(i_1,j_0-1) - s(i_0-1,j_1) + s(i_0-1,j_0-1)$$

Image f

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

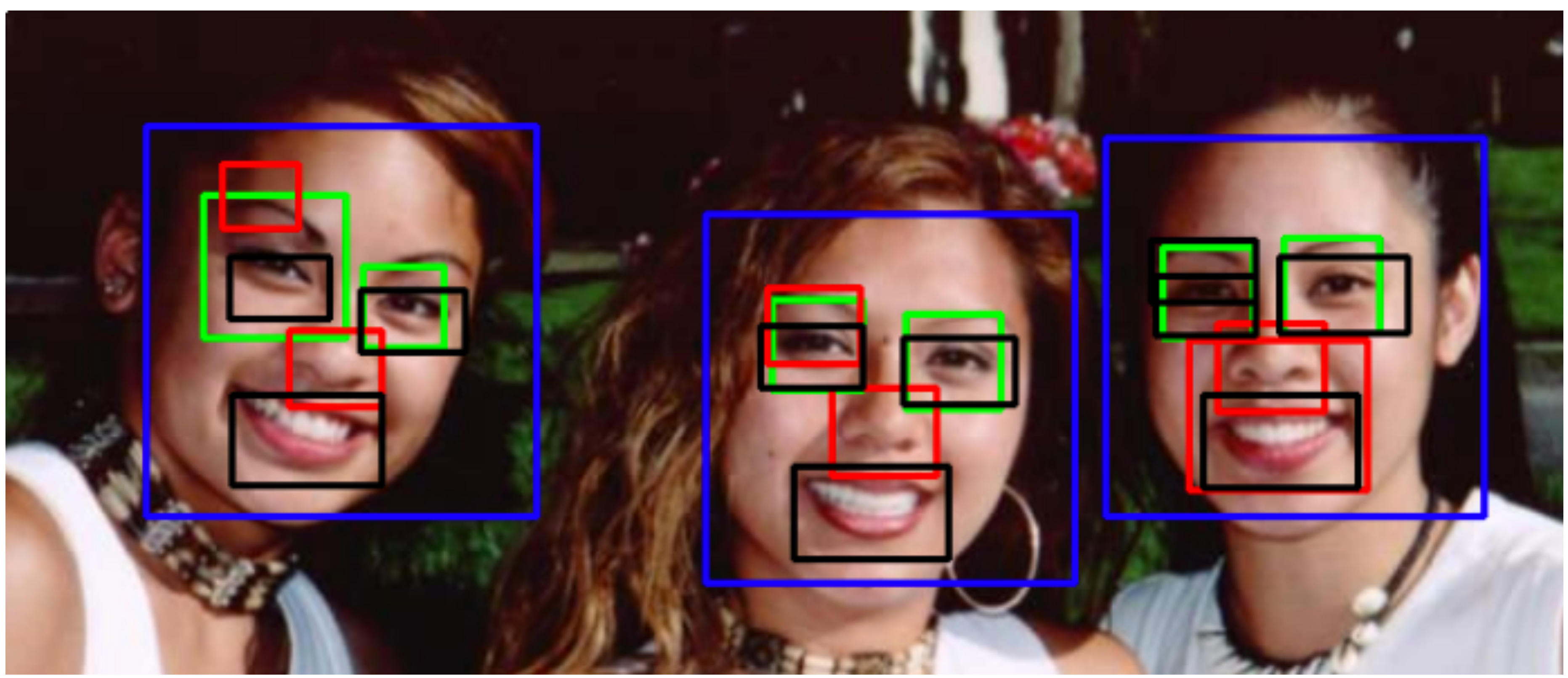
Integral image s

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

Integral image s

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

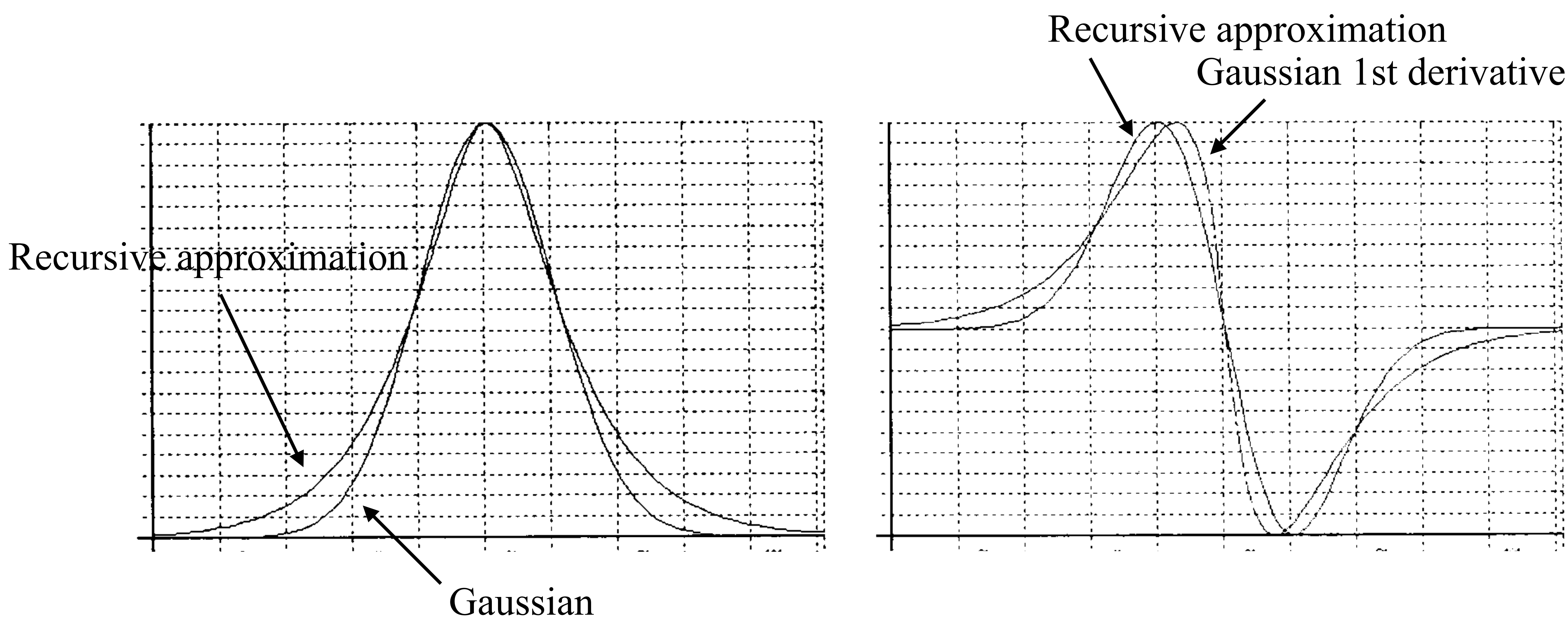
Application: Face Detection



Viola & Jones 2001

Recursive Filters

- The efficient raster-scan computation used to compute the integral image is an example of a recursive filter.
- Also known as infinite-impulse response (IIR) filters
- Unfortunately Gaussian derivatives do not have a recursive implementation.
- However, there are efficient recursive approximations



Optimal Linear Filters

- For some problems and under some conditions, it can be proven that linear filtering yields an optimal solution.

  - Example: estimation of the mean irradiance from a surface in the scene.

    Let $f(x,y) = g(x,y) + n(x,y)$ be a noisy image patch,
    where $g(x,y)$ is the true irradiance from the patch
    and $n(x,y)$ is random noise added by the sensor.

    If $n(x,y)$ is additive Gaussian, independent and identically distributed (IID), then
    $\bar{f} = \frac{1}{n}\sum_{x,y} f(x,y)$ is an optimal (unbiased and efficient) estimator of $\bar{g} = \frac{1}{n}\sum_{x,y} g(x,y)$,
    where n is the number of pixels in the patch.

  - Notes:

    This is a box filter, which can be implemented using integral images.

    $\bar{f}$ minimizes the mean squared deviation: $\bar{f} = \arg\min_{\hat{f}} \frac{1}{n}\sum_{x,y}\left(\hat{f} - f(x,y)\right)^2$

Outline

- Point Operators
- Linear Filters
- **Nonlinear Filters**

Nonlinear Filters

❖ For many problems/conditions, linear filtering is provably sub-optimal.

- Example: shot noise.

Image + shot noise

After linear filtering with a Gaussian lowpass filter

- Can we do better than this?

Median Filters

- A median filter simply replaces the pixel value with the median value in its neighbourhood.

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

MATLAB function
medfilt2

- It is a good choice for shot (heavy-tailed) noise, as the median value is not affected by extreme noise values

- Can be computed in linear time.

- Reduces blurring of edges

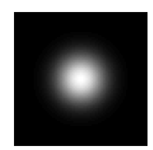


Image + shot noise

Gaussian lowpass filter

Median filter

Median Filters

- While averaging minimizes the squared deviation, median filtering minimizes the absolute (L_1) error:

$$\bar{f} = \arg\min_{\hat{f}} \frac{1}{n} \sum_{x,y} \left| \hat{f} - f(x,y) \right|$$

Bilateral Filters

- Gaussian linear filters provide a nice way of grading the weights of neighbouring pixels so that closer pixels have more influence than more distant pixels.

- Median filters provide a nice way of reducing the influence of outlier values.

- Can we somehow <u>combine these two things</u>?

Bilateral Filters

In the bilateral filter, the output pixel value depends on a weighted combination of neighboring pixel values

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

The weighting coefficient $w(i,j,k,l)$ depends on the product of a *domain kernel*

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

0.1	0.3	0.4	0.3	0.1
0.3	0.6	0.8	0.6	0.3
0.4	0.8	1.0	0.8	0.4
0.3	0.6	0.8	0.6	0.3
0.1	0.3	0.4	0.3	0.1

and a data-dependent *range kernel* (Figure 3.19d),

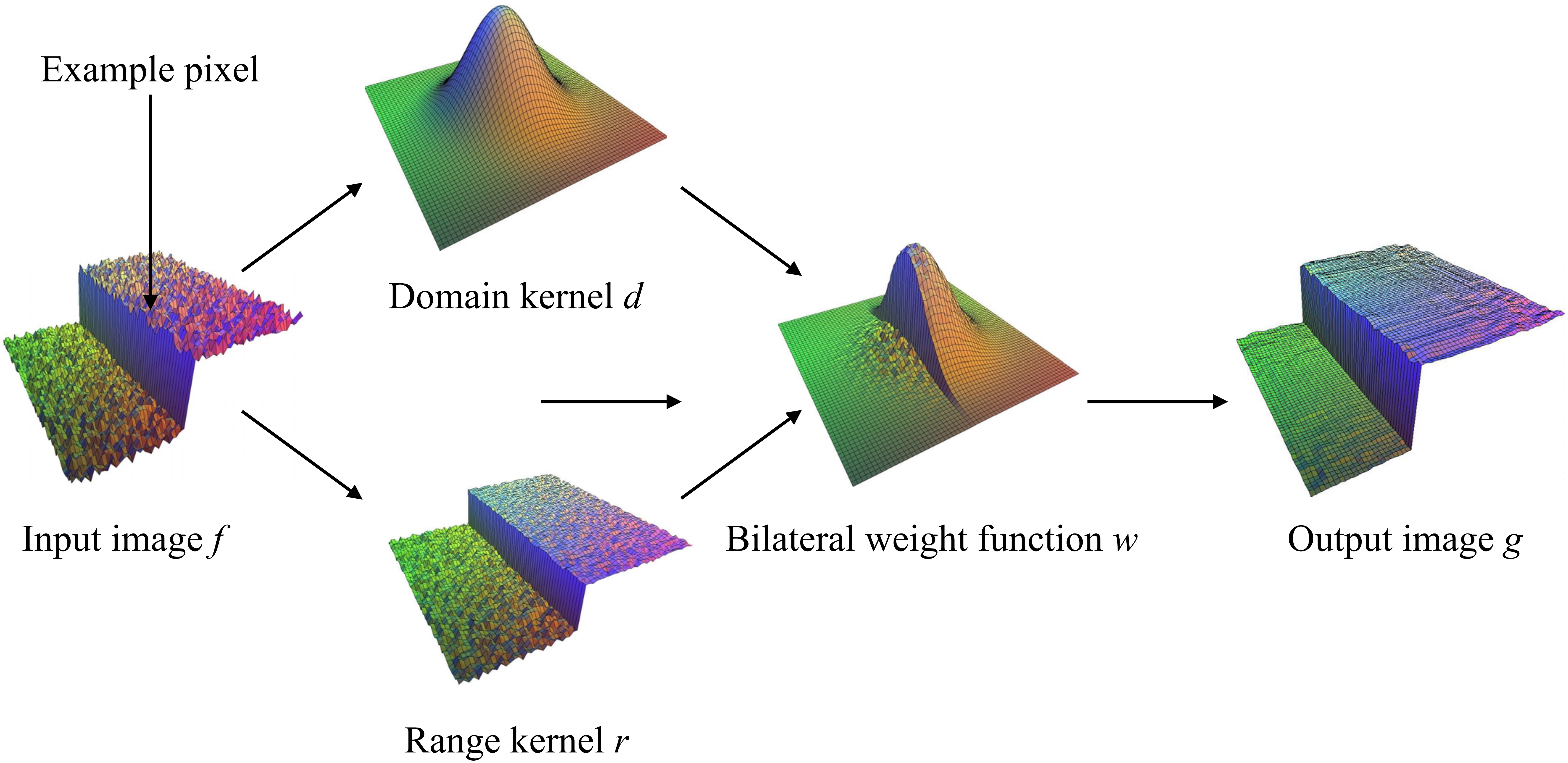
$$r(i,j,k,l) = \exp\left(-\frac{\|f(i,j) - f(k,l)\|^2}{2\sigma_r^2}\right).$$

0.0	0.0	0.0	0.0	0.2
0.0	0.0	0.0	0.4	0.8
0.0	0.0	1.0	0.8	0.4
0.0	0.2	0.8	0.8	1.0
0.2	0.4	1.0	0.8	0.4

When multiplied together, these yield the data-dependent *bilateral weight function*

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

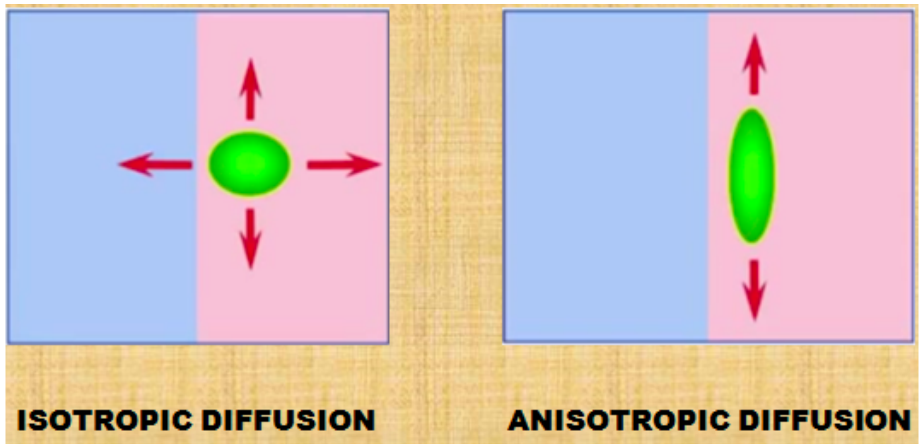
Bilateral Filters - Example



Tomasi & Manduci, 1998

Anisotropic Diffusion

- Iterative application of bilateral filtering leads to a smoothing process equivalent to a popular edge-preserving smoothing technique due to Perona & Malik called *anistropic diffusion*.



- e.g., for a 4-neighbourhood:

$$
\begin{aligned}
d(i,j,k,l) &= \exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2}\right) \\
&= \begin{cases} 1, & |k-i|+|l-j|=0, \\ \eta = e^{-1/2\sigma_d^2}, & |k-i|+|l-j|=1. \end{cases}
\end{aligned}
$$

- and so

$$
\begin{aligned}
f^{(t+1)}(i,j) &= \frac{f^{(t)}(i,j)+\eta\sum_{k,l} f^{(t)}(k,l)r(i,j,k,l)}{1+\eta\sum_{k,l} r(i,j,k,l)} \\
&= f^{(t)}(i,j)+\frac{\eta}{1+\eta R}\sum_{k,l} r(i,j,k,l)[f^{(t)}(k,l)-f^{(t)}(i,j)],
\end{aligned}
$$

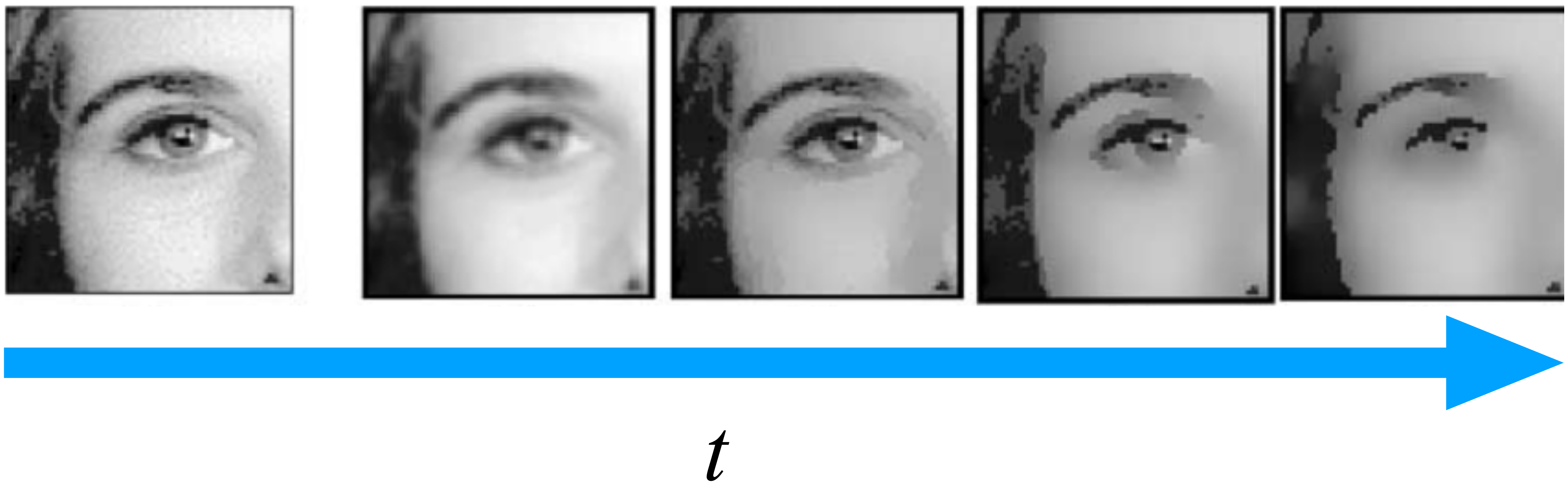
where $R = \sum_{(k,l)} r(i,j,k,l)$, (k,l) are the $\mathcal{N}_4$ neighbors of (i,j)

Perona & Malik, 1990

Anisotropic Diffusion Example

- But note that

$$\lim_{t \to \infty} f^{(t)}(i,j) = \text{constant}$$

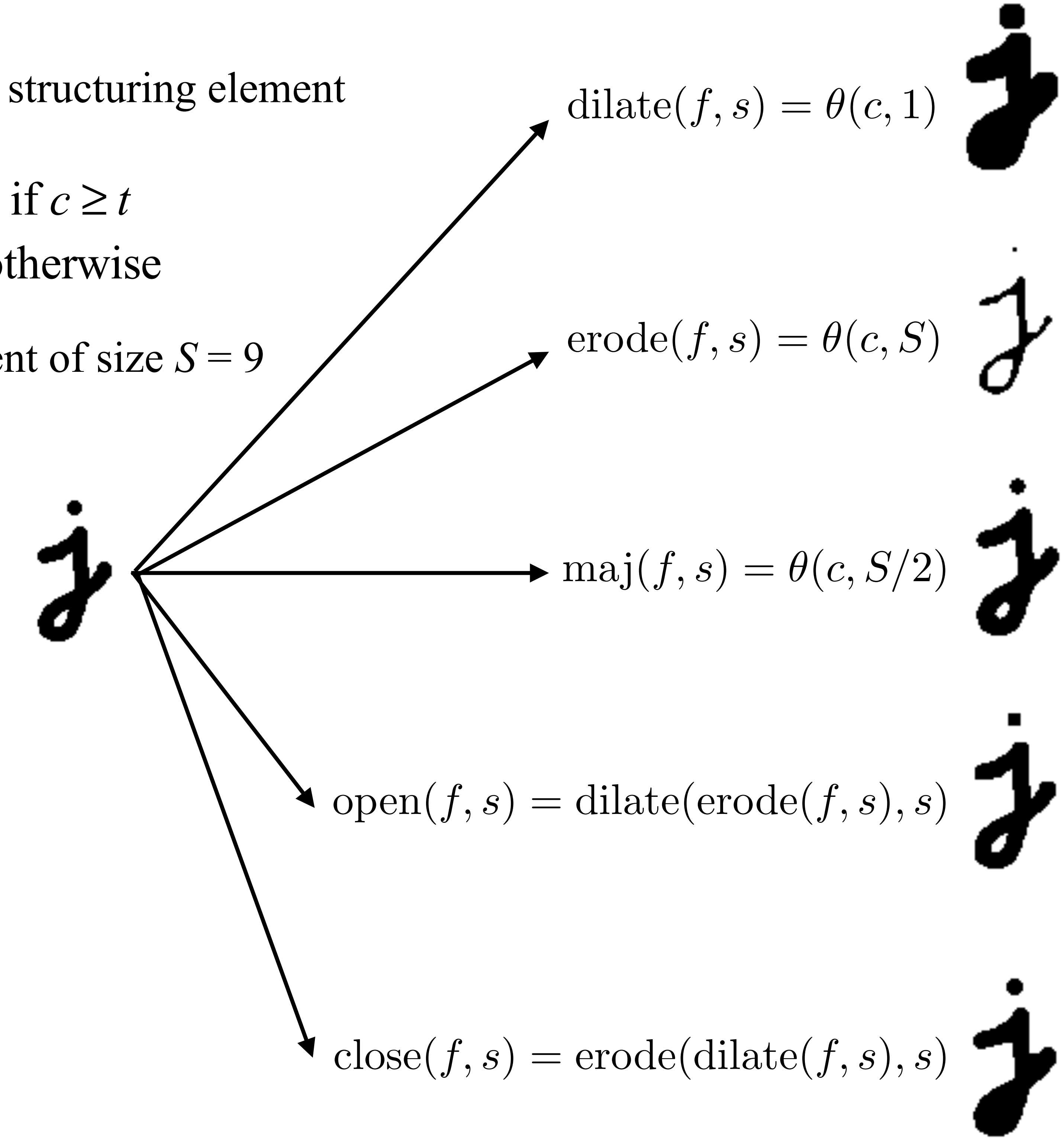


End of Lecture
Oct 1, 2018

Morphological Filters

- Binary image processing often involves morphological filtering:
  - Convolve with local filter s called a structuring element
    $c = f * s$
  - Threshold result: $\theta(c,t) = \begin{cases} 1 & \text{if } c \geq t \\ 0 & \text{otherwise} \end{cases}$
  - Example: Boxcar structuring element of size $S = 9$

$$s = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



MATLAB functions
imdilate
imerode
imopen
inclose

The Distance Transform

The distance transform $D(i,j)$ of a binary image $b(i,j)$ is defined as follows. Let $d(k,l)$ be some *distance metric* between pixel offsets. Two commonly used metrics include the *city block* or *Manhattan* distance

$$d_1(k,l) = |k| + |l|$$

and the *Euclidean* distance

$$d_2(k,l) = \sqrt{k^2 + l^2}.$$

The distance transform is then defined as

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

i.e., it is the distance to the *nearest* background pixel whose value is 0.

MATLAB function
bwdist

Computing the Distance Transform

- City block
  - Forward-backward two-pass raster scan
    - Initialize:

      $b(\text{find}(b(:))) = \infty$
    - Forward pass

```
for j = 2:n
   if b(1,j) > 0
      b(1,j) = 1 + b(1,j-1)
for i = 2:m
   if b(i,1) > 0
      b(i,1) = 1 + b(i-1,1)
   for j = 2:n
      if b(i, j) > 0
         b(i, j) = 1 + min(b(i-1, j),b(i, k-1))
```

    - Backward pass

```
for j = n-1:-1:1
   if b(m,j) > 0
      b(m,j) = 1 + min(b(m,j),b(m,j+1))
for i = m-1:-1:1
   if b(i,n) > 0
      b(i,n) = 1 + min(b(i,n),b(i+1,n))
   for j = n-1:-1:1
      if b(i, j) > 0
         b(i, j) = min(b(i,j),1+b(i+1, j),1+b(i, j+1))
```

Input image

0	0	0	0	1	0	0
0	0	1	1	1	0	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0

Forward pass

0	0	0	0	1	0	0
0	0	1	1	2	0	0
0	1	2	2	3	1	0
0	1	2	3			

Backward pass

0	0	0	0	1	0	0
0	0	1	1	2	0	0
0	1	2	2	3	1	0
0	1	2	2	1	1	0
0	1	2	1	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0

Distance transform

0	0	0	0	1	0	0
0	0	1	1	1	0	0
0	1	2	2	2	1	0
0	1	2	2	1	1	0
0	1	2	1	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0

Outline

- Point Operators
- Linear Filters
- Nonlinear Filters