Image processing

Brian Curless
CSE 457
Spring 2013

Reading

Jain, Kasturi, Schunck, *Machine Vision*. McGraw-Hill, 1995. Sections 4.2-4.4, 4.5(intro), 4.5.5, 4.5.6, 5.1-5.4. [online handut]

What is an image?

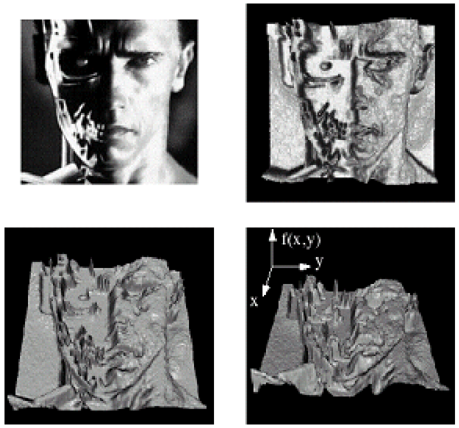
We can think of an **image** as a function, *f*, from R^2 to R:

- f (x, y) gives the intensity of a channel at position (x, y)
- Realistically, we expect the image only to be defined over a rectangle, with a finite range:
  - $f : [a, b] \times [c, d] \rightarrow [0,1]$

A color image is just three functions pasted together. We can write this as a "vector-valued" function:

$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$

Images as functions



What is a digital image?

In computer graphics, we usually operate on **digital** (**discrete**) images:

- **Sample** the space on a regular grid
- **Quantize** each sample (round to nearest integer)

If our samples are Δ apart, we can write this as:

$$f[i,j] = \text{Quantize}\{ f(i\,\Delta, j\,\Delta) \}$$

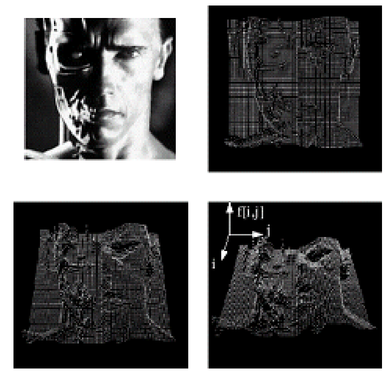


Image processing

An **image processing** operation typically defines a new image *g* in terms of an existing image *f*.

The simplest operations are those that transform each pixel in isolation. These pixel-to-pixel operations can be written:

$$g(x,y) = t(f(x,y))$$

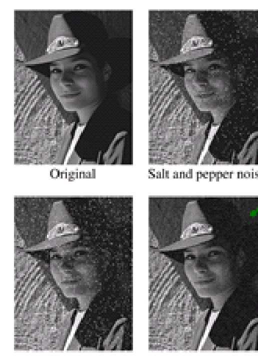
Examples: threshold, RGB $\rightarrow$ grayscale

Note: a typical choice for mapping to grayscale is to apply the YIQ television matrix and keep the Y.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Noise

Image processing is also useful for noise reduction and edge enhancement. We will focus on these applications for the remainder of the lecture…



Original — Salt and pepper noise — Impulse noise — Gaussian noise

$I(x,y) = I^*(x,y) + \eta(0,\sigma)$

Common types of noise:

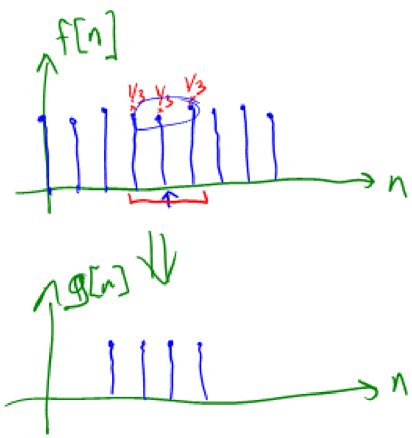
- **Salt and pepper noise**: contains random occurrences of black and white pixels
- **Impulse noise:** contains random occurrences of white pixels
- **Gaussian noise:** variations in intensity drawn from a Gaussian normal distribution

Ideal noise reduction

Ideal noise reduction

Practical noise reduction

How can we "smooth" away noise in a single image?



Is there a more abstract way to represent this sort of operation? *Of course there is!*

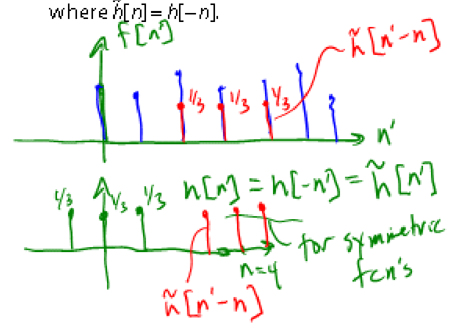
Discrete convolution

One of the most common methods for filtering an image is called **discrete convolution**. (We will just call this "convolution" from here on.)

In 1D, convolution is defined as:

$$
\begin{aligned}
g[n] &= f[n] * h[n] \\
&= \sum_{n'} f[n']h[n-n'] \\
&= \sum_{n'} f[n']\tilde{h}[n'-n]
\end{aligned}
$$

where $\tilde{h}[n] = h[-n]$.



Discrete convolution

One can show that convolution has some convenient properties. Given functions *a*, *b*, *c*:

$$a * b = b * a$$
$$(a * b) * c = a * (b * c)$$
$$a * (b + c) = a * b + a * c$$

We'll make use of these properties later…

Convolution in 2D

In two dimensions, convolution becomes:

$$\begin{aligned} g[n,m] &= f[n,m] * h[n,m] \\ &= \sum_{m'}\sum_{n'} f[n',m']\,h[n-n',m-m'] \\ &= \sum_{m'}\sum_{n'} f[n',m']\,\tilde{h}[n'-n,m'-m] \end{aligned}$$

where $\tilde{h}[n,m] = h[-n,-m]$.

Convolution representation

Since *f* and *h* are defined over finite regions, we can write them out in two-dimensional arrays:

Image f

128	54	9	78	100
145	98	240	233	86
89	177	246	228	127
67	90	255	237	95
106	111	128	167	20
221	154	97	123	0

filter h

X 0.1	X 0.1	X 0.1
X 0.1	X 0.2	X 0.1
X 0.1	X 0.1	X 0.1

Note: *This is not matrix multiplication*!

Q: What happens at the boundary of the image?

Mean filters

How can we represent our noise-reducing averaging as a convolution filter (know as a **mean filter**)?

$$\frac{1}{N^2}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} N$$

N

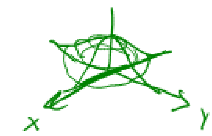
$\frac{1}{NM}$

M

Effect of mean filters

Gaussian filters

$e^{-\frac{(x^2+y^2)}{2\sigma^2}}$



Gaussian filters weigh pixels based on their distance from the center of the convolution filter. In particular:

$$h[n,m] = \frac{e^{-(n^2+m^2)/(2\sigma^2)}}{C}$$

This does a decent job of blurring noise while preserving features of the image.

What parameter controls the width of the Gaussian? σ

What happens to the image as the Gaussian filter kernel gets wider? blurrier

What is the constant C? What should we set it to?

$$C = \sum_{n,m} h[n,m]$$

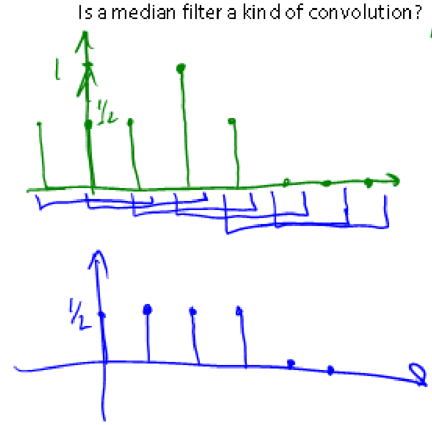
Effect of Gaussian filters

Median filters

A **median filter** operates over an mxm region by selecting the median intensity in the region.

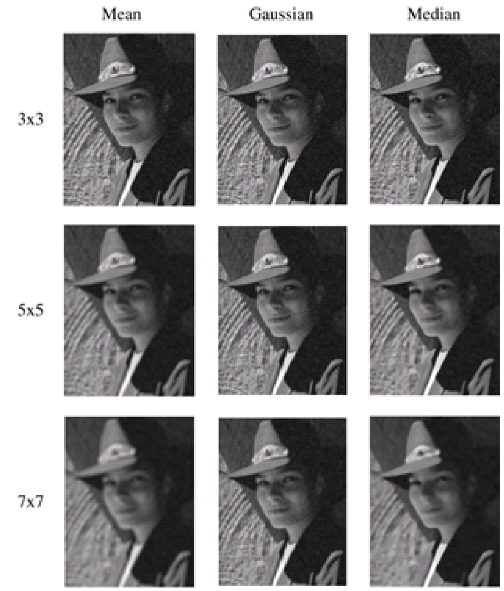
What advantage does a median filter have over a mean filter? edge preserving

Is a median filter a kind of convolution? No.

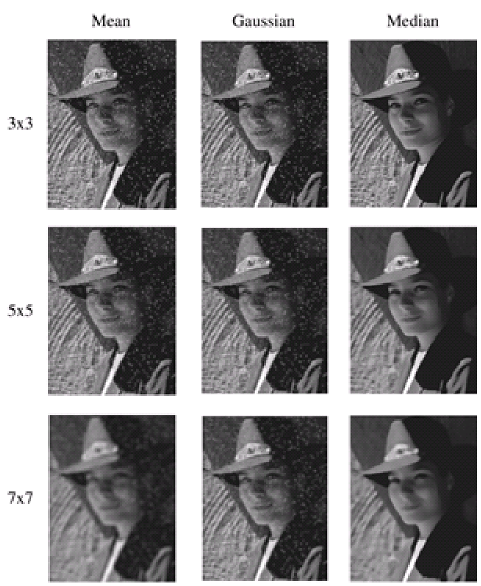


Effect of median filters

Comparison: Gaussian noise

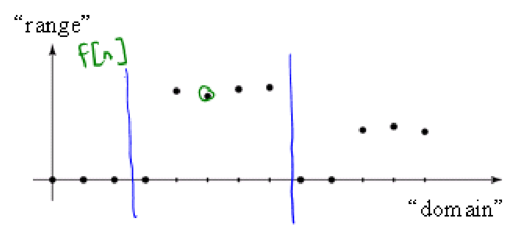


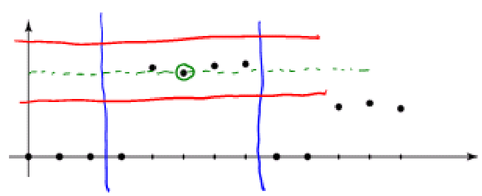
Comparison: salt and pepper noise



Bilateral filtering

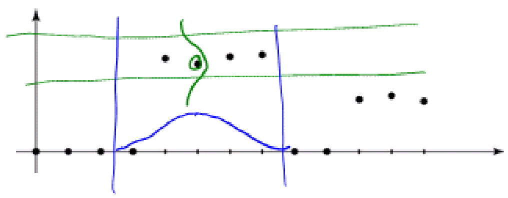
Bilateral filtering is a method to average together nearby samples only if they are similar in value.





Bilateral filtering

We can also change the filter to something "nicer" like Gaussians:



Recall that convolution looked like this:

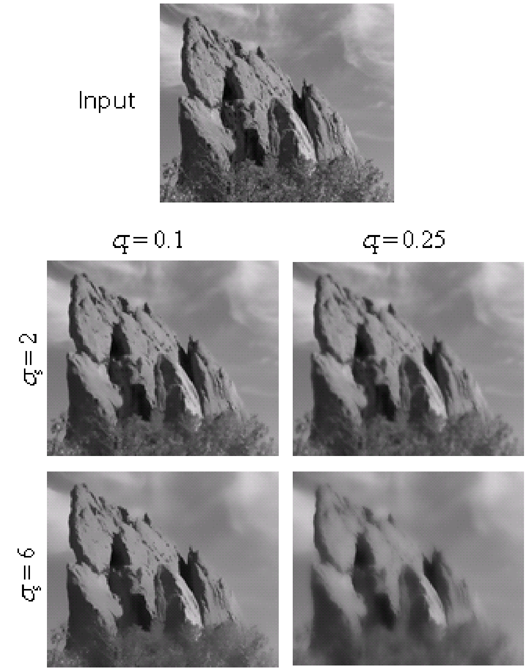
$$g[n] = \sum_{n'} f[n']h[n-n']$$

Bilateral filter is similar, but includes both range and domain filtering:

$$g[n] = 1/C \sum_{n'} f[n']h_{\sigma_s}[n-n'] \; h_{\sigma_r}(f[n]-f[n'])$$

and you have to normalize as you go:

$$C = \sum_{n'} h_{\sigma_s}[n-n'] \; h_{\sigma_r}(f[n]-f[n'])$$



Paris, et al. SIGGRAPH course notes 2007

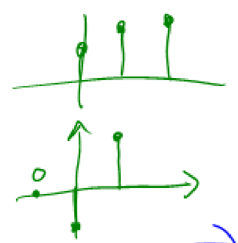
Edge detection

One of the most important uses of image processing is **edge detection:**

- Really easy for humans
- Really difficult for computers

- Fundamental in computer vision
- Important in many graphics applications

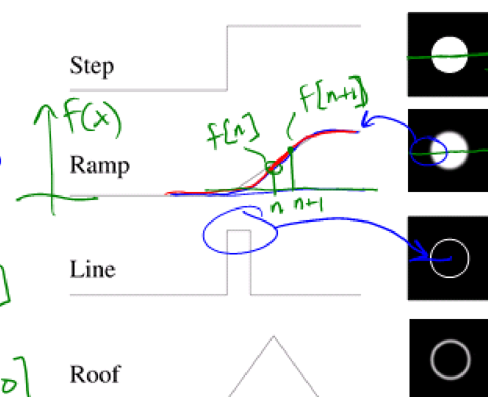
What is an edge?

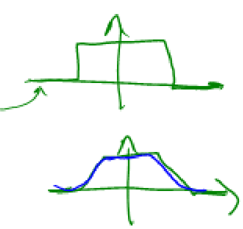


$[0 \;\; -1 \;\; 1]$

"$h[-n] = \tilde{h}[n]$"

$h[n] = [1 \;\; -1 \;\; 0]$





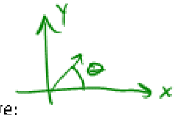
$$\frac{df}{dx} \approx f[n+1] - f[n]$$

$$f_x[n] = f[n+1] - f[n] = h[n] * f[n]$$

Q: How might you detect an edge in 1D?

$$\left|\frac{df}{dx}\right| > \text{thresh} \Rightarrow \text{edge pixel}$$

Gradients



The **gradient** is the 2D equivalent of the derivative:

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

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

Properties of the gradient

- It's a vector
- Points in the direction of maximum increase of f
- Magnitude is rate of increase

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

How can we approximate the gradient in a discrete image?

$$f_x = h_x * f$$

$$f_y = h_y * f$$

$$\tilde{h}_x = \begin{bmatrix} 0 & -1 & 1 \end{bmatrix}$$

$$\tilde{h}_y = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

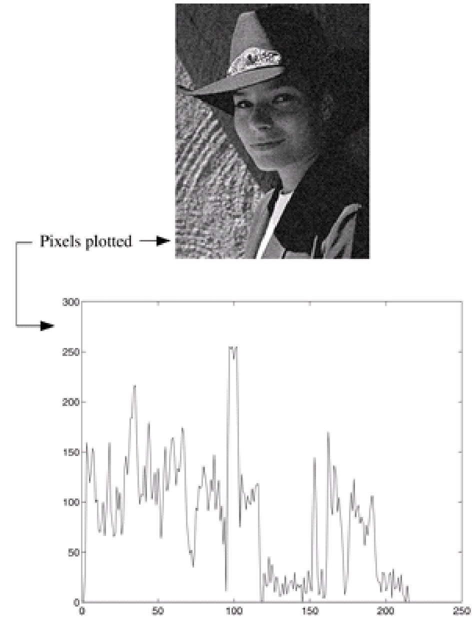
$$f_x[n,m] = f[n+1,m] - f[n,m]$$

$$f_y[n,m] = f[n,m+1] - f[n,m]$$

$$h_x = \begin{bmatrix} 1 & -1 & 0 \end{bmatrix}$$

$$h_y = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$$

Less than ideal edges



Steps in edge detection

Edge detection algorithms typically proceed in three or four steps:

- **Filtering**: cut down on noise
- **Enhancement**: amplify the difference between edges and non-edges
- **Detection**: use a threshold operation
- **Localization** (optional): estimate geometry of edges as 1D contours that can pass between pixels

Edge enhancement

A popular gradient filter is the **Sobel operator**:

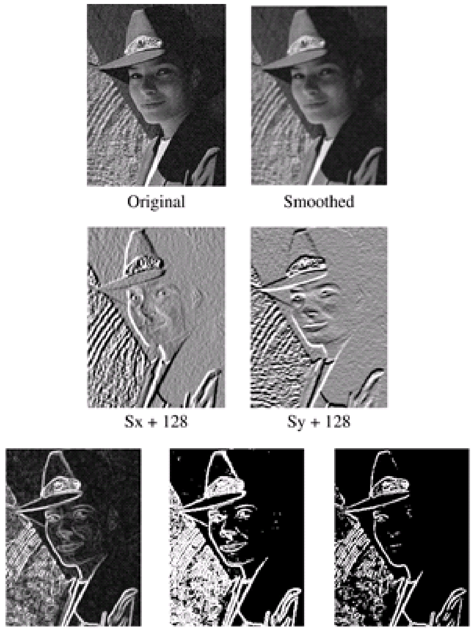
$$\tilde{s}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \tilde{h}_x = \begin{bmatrix} 0 & -1 & 1 \end{bmatrix}$$

$$\tilde{s}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad \tilde{h}_y = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

We can then compute the magnitude of the vector $(\tilde{s}_x, \tilde{s}_y)$.

Note that these operators are conveniently "pre-flipped" for convolution, so you can directly slide these across an image without flipping first.

Results of Sobel edge detection



Original | Smoothed

Sx + 128 | Sy + 128

Magnitude | Threshold = 64 | Threshold = 128

Second derivative operators

$f_x = h_x * f$

$f_{xx} = h_x * (h_x * f)$

$= (h_x * h_x) * f$

$= h_{xx} * f$

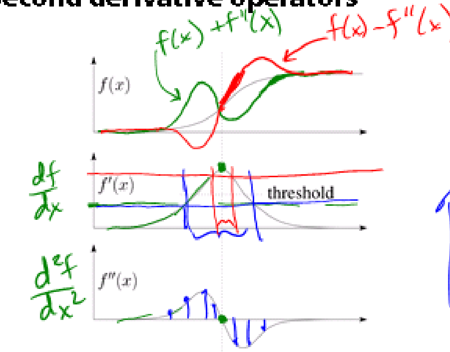
$h_x \; [1 \;\; -1 \;\; 0]$

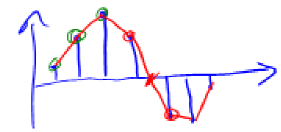
$\tilde{h}_x \; [0 \;\; -1 \;\; 1]$

$[0 \;\; -1 \;\; 1]$

$[0 \;\; -1 \;\; 1]$

$h_{xx} = [1 \;\; -2 \;\; 1]$





The Sobel operator can produce thick edges. Ideally, we're looking for infinitely thin boundaries.

An alternative approach is to look for local extrema in the first derivative: places where the change in the gradient is highest.

Q: A peak in the first derivative corresponds to what in the second derivative? 0

Q: How might we write this as a convolution filter?

Localization with the Laplacian

An equivalent measure of the second derivative in 2D is the **Laplacian**:

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

$\frac{d^2 f}{dx^2} = h_{xx} * f + h_{yy} * f$

$= (h_{xx} + h_{yy}) * f$

Δ

Using the same arguments we used to compute the gradient filters, we can derive a Laplacian filter to be:

$$\Delta = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

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

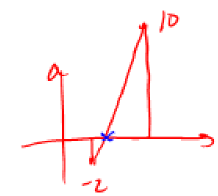
$= \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\Delta} * f$

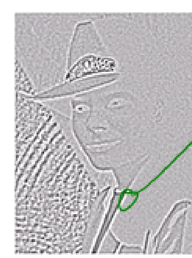
(The symbol Δ is often used to refer to the *discrete* Laplacian filter.)

Zero crossings in a Laplacian filtered image can be used to localize edges.

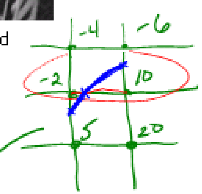
Localization with the Laplacian

Original | Smoothed





Laplacian (+128)



Marching Squares

Sharpening with the Laplacian

$f - \lambda \Delta * f$

$\Downarrow$

$$\begin{bmatrix} 0 & -\lambda & 0 \\ -\lambda & 1+4\lambda & -\lambda \\ 0 & -\lambda & 0 \end{bmatrix}$$

$\lambda = \frac{1}{2}$

$$\begin{bmatrix} 0 & -1/2 & 0 \\ -1/2 & 3 & -1/2 \\ 0 & -1/2 & 0 \end{bmatrix}$$

Original

Laplacian (+128)

Original + Laplacian

Original - Laplacian

$f - \Delta * f$

$= (1 - \Delta) * f$

$= \left([1] - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \right) * f$

$= \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \right) * f$

$= \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} * f$

Why does the sign make a difference?

How can you write the filter that makes the sharpened image?

Summary

What you should take away from this lecture:

- The meanings of all the boldfaced terms.
- How noise reduction is done
- How discrete convolution filtering works
- The effect of mean, Gaussian, and median filters
- What an image gradient is and how it can be computed
- How edge detection is done
- What the Laplacian image is and how it is used in either edge detection or image sharpening