

Image processing

1

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.

2

What is an image?

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

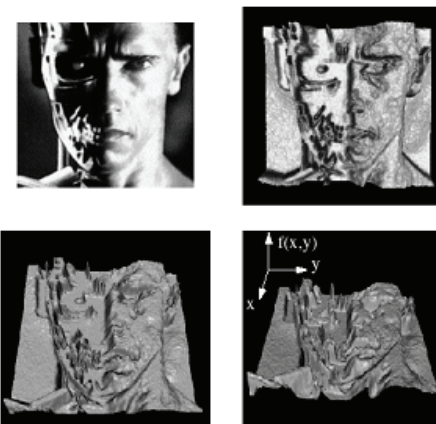
- $f(x, y)$ gives the intensity of a channel at position (x, y)
- Realistically, we expect the image only to be defined over a rectangle, with a finite range:
 - $f: [a, b] \times [c, d] \rightarrow [0, 1]$

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

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

3

Images as functions



4

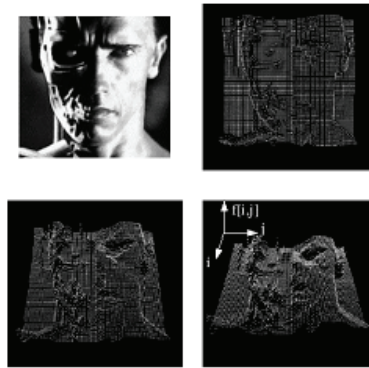
What is a digital image?

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:

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



5

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 g to grayscale is to apply the YIQ television matrix and keep the Y.

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

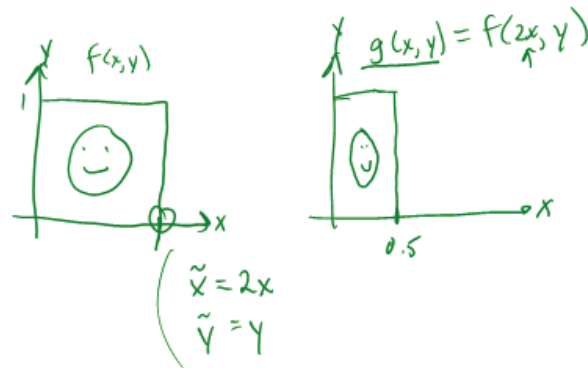
6

Pixel movement

Some operations preserve intensities, but move pixels around in the image

$$g(x, y) = f(\tilde{x}(x, y), \tilde{y}(x, y))$$

Examples: many amusing warps of images



[Show image sequence.]

7

Noise

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



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

8

Ideal noise reduction



9

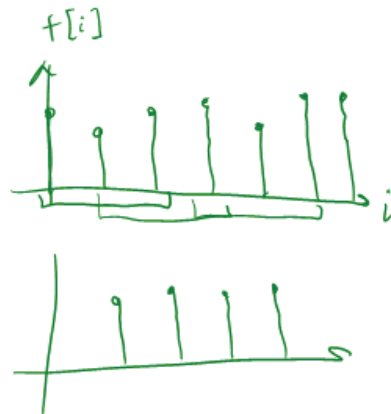
Ideal noise reduction



10

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!*

11

Convolution

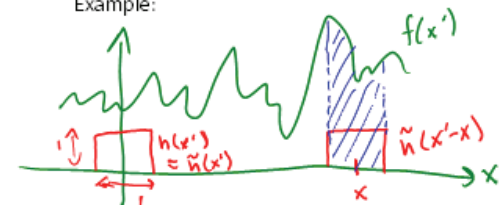
One of the most common methods for filtering an image is called **convolution**.

In 1D, convolution is defined as:

$$\begin{aligned} g(x) &= f(x) * h(x) \\ &= \int_{-\infty}^{\infty} f(x') h(x - x') dx' \\ &= \int_{-\infty}^{\infty} f(x') \tilde{h}(x' - x) dx' \end{aligned}$$

where $\tilde{h}(x) = h(-x)$.

Example:



12

Discrete convolution

For a digital signal, we define **discrete convolution** as:

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

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

Aside:

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

$$\begin{aligned} a * b &= b * a \\ (a * b) * c &= a * (b * c) \\ a * (b + c) &= a * b + a * c \end{aligned}$$

We'll make use of these properties later...

13

Convolution in 2D

In two dimensions, convolution becomes:

$$\begin{aligned} g(x, y) &= f(x, y) * h(x, y) \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x', y') h(x - x', y - y') dx' dy' \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x', y') \tilde{h}(x' - x, y' - y) dx' dy' \end{aligned}$$

where $\tilde{h}(x, y) = h(-x, -y)$.

14

Discrete convolution in 2D

Similarly, discrete convolution in 2D becomes:

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

where $\tilde{h}[i, j] = h[-i, -j]$.

Note: convolution (continuous or discrete, in any dimension) is a **linear** operation. Here is one of the consequences of that linearity:

$$\begin{aligned} g[i, j] &= f[i, j] * h[i, j] + f[i, j] * e[i, j] \\ &= f[i, j] * (h[i, j] + e[i, j]) \end{aligned}$$

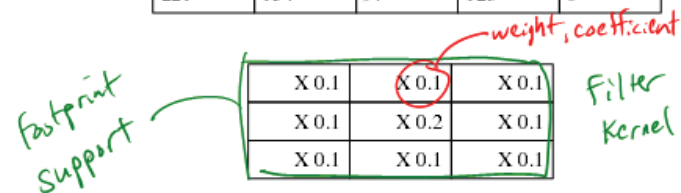
15

Convolution representation

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

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

Image



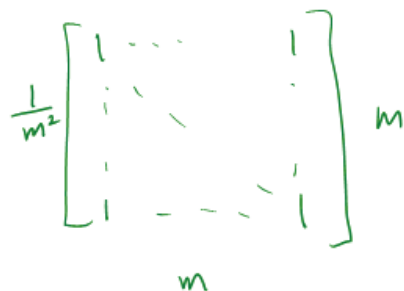
Note: This is not matrix multiplication!

Q: What happens at the edges?

16

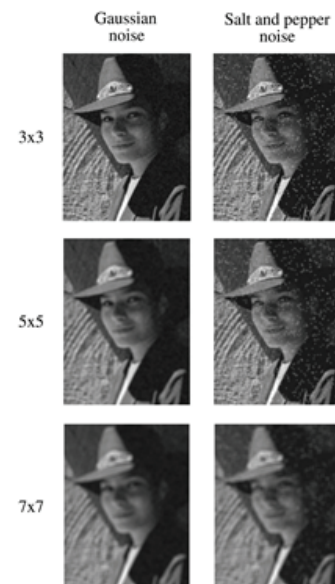
Mean filters

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



17

Effect of mean filters



18

Gaussian filters

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

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

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

What parameter controls the width of the Gaussian? σ

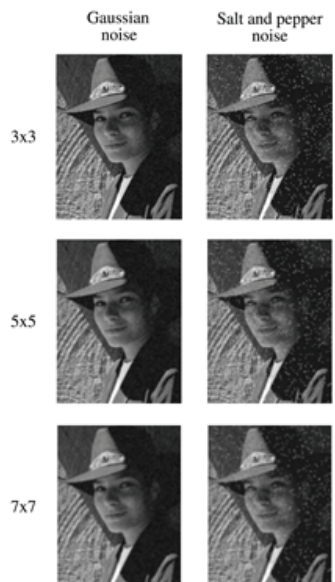
What happens to the image as the Gaussian filter kernel gets wider? *blurrer*

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

$$C = \sum_{i,j} e^{-(i^2 + j^2)/(2\sigma^2)}$$

19

Effect of Gaussian filters

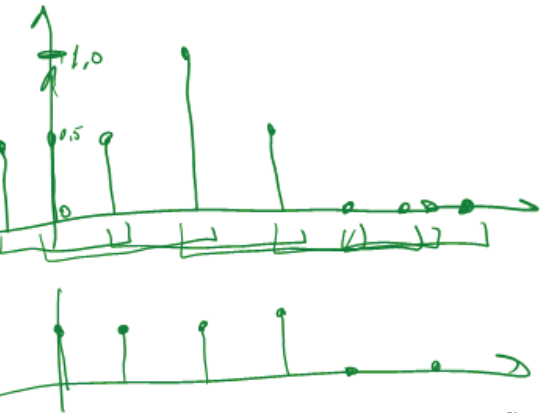


20

Median filters

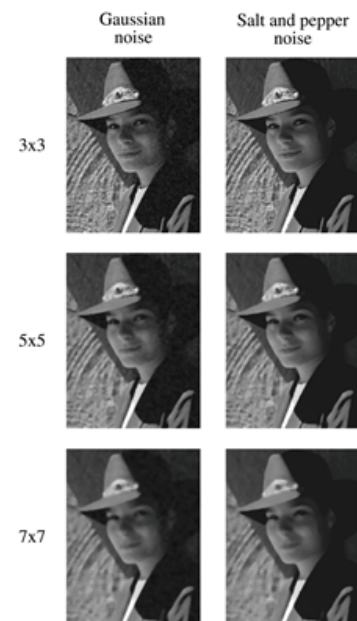
A **median filter** operates over an $m \times m$ region by selecting the median intensity in the region.

What advantage does a median filter have over a mean filter?
outlier rejection, tends not to smooth edges (as much)



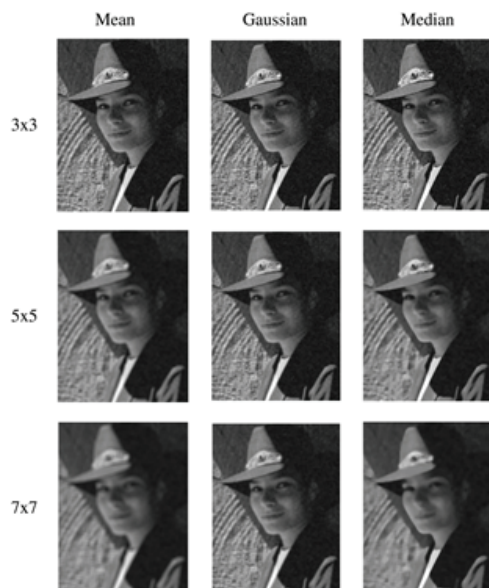
21

Effect of median filters



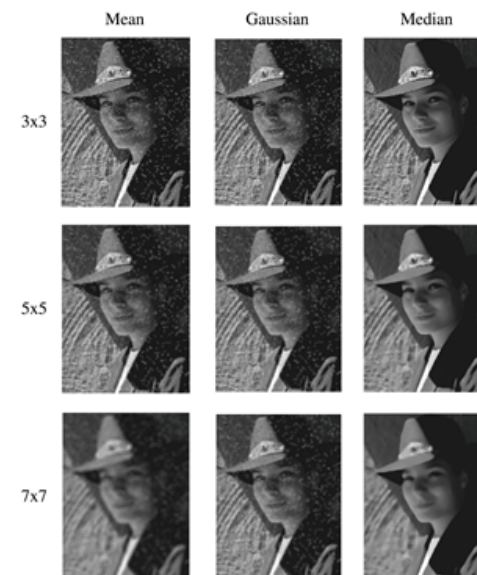
22

Comparison: Gaussian noise



23

Comparison: salt and pepper noise



24

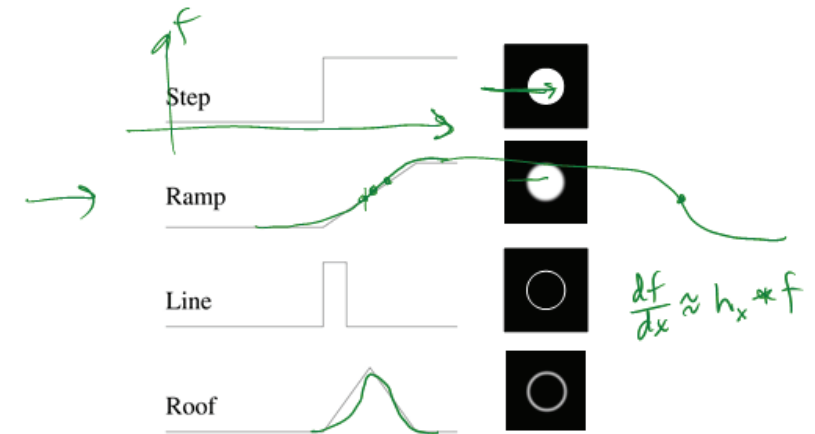
Edge detection

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

25

What is an edge?



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

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

$$g[i] = f[i+1] - f[i]$$

$$\tilde{h}_x = [0 \ 1]$$

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

26

Gradients

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

$$\tan \theta = \frac{\partial f / \partial y}{\partial f / \partial x}$$

Properties of the gradient

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

How can we approximate the gradient in a discrete image?

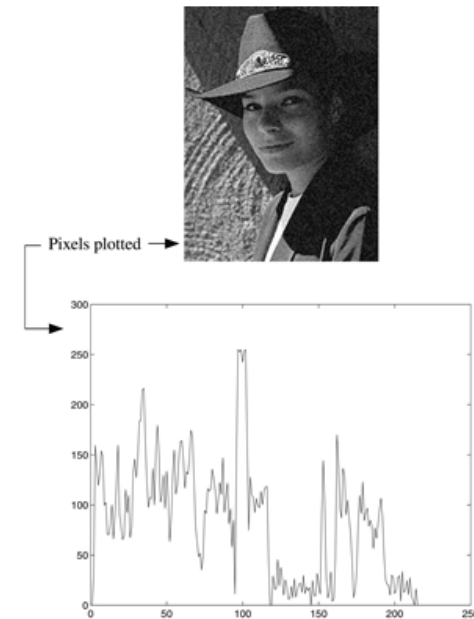
$$g_x[i] = f[i, j+1] - f[i, j]$$

$$g_y[i] = f[i, j+1] - f[i, j]$$

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

27

Less than ideal edges



28

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, which generally pass between pixels

29

Edge enhancement

A popular gradient magnitude computation is the **Sobel operator**:

$$s_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

(pre-flipped)

$$s_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

We can then compute the magnitude of the vector (s_x, s_y) .

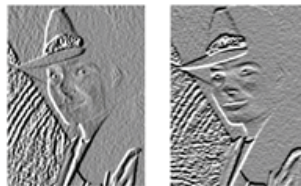
30

Results of Sobel edge detection



Original

Smoothed



Sx + 128

Sy + 128



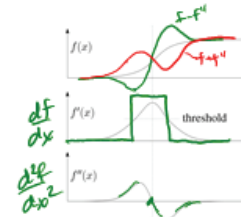
Magnitude

Threshold = 64

Threshold = 128

31

Second derivative operators



$$\frac{df}{dx} \approx h_x * f$$

$$\frac{d^2f}{dx^2} \approx h_x * (h_x * f) \approx (h_x * h_x) * f$$

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?

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

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

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

2

Localization with the Laplacian

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

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

$$\frac{\partial^2}{\partial x^2} = h_{xx} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

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

$$\frac{\partial^2}{\partial y^2} = h_{yy} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

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

$$\nabla^2 f \approx h_{xx} * f + h_{yy} * f \approx (h_{xx} + h_{yy}) * f$$

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

Zero crossings in a Laplacian filtered image correspond to positions of maximum gradient. These zero crossings can be used to localize edges.

33

Localization with the Laplacian



Original



Smoothed

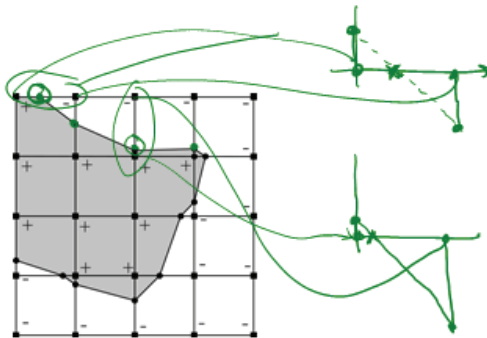


Laplacian (+128)

34

Marching squares

We can convert these signed values into edge contours using a "marching squares" technique:



35

Sharpening with the Laplacian



Original



Laplacian (+128)



Original + Laplacian



Original - Laplacian

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

$$\frac{1}{2} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 6 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$f - \lambda \Delta * f$$

$$f - \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * f$$

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

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & 1 \\ 0 & -1 & 0 \end{bmatrix} * f$$

Why does the sign make a difference?

How can you write each filter that makes each bottom image?

36

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