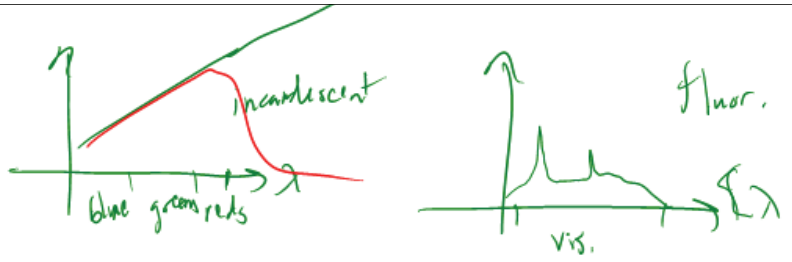


Image processing

Reading

Shirley, Ch. 9.

Jain, Kasturi, Schunck, *Machine Vision*. McGraw-Hill, 1995. Sections 5.1-5.4. [Handout]

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 → grayscale

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

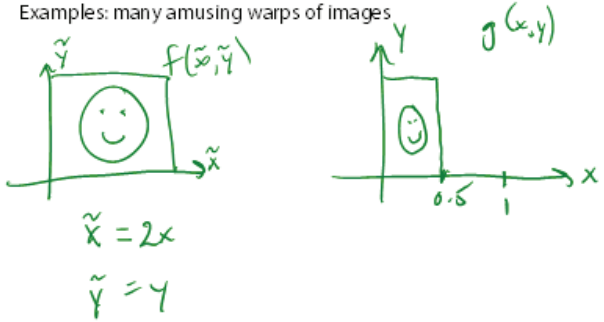
$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

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.]

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

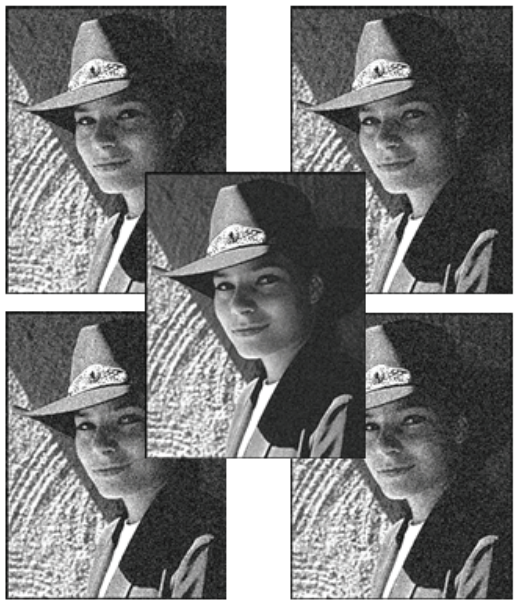
Impulse noise | Gaussian noise

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

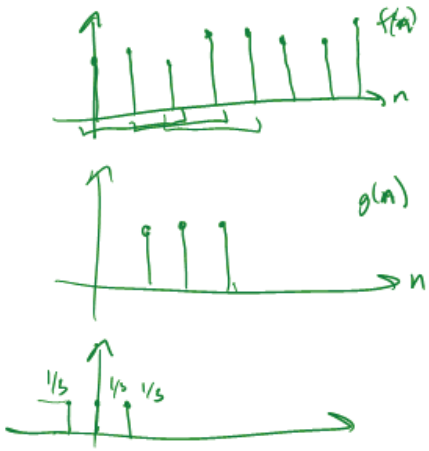
Ideal noise reduction

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

For a digital signal, we define **discrete convolution** 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 $\tilde{h}[n] = h[-n]$.

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…

Discrete convolution in 2D

Similarly, discrete convolution in 2D is:

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

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:

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

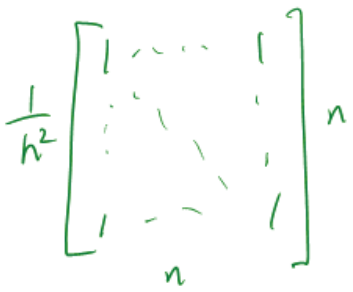
X 0.1	X 0.1	X 0.1
X 0.1	X 0.2	X 0.1
X 0.1	X 0.1	X 0.1

Note: *This is not matrix multiplication*!

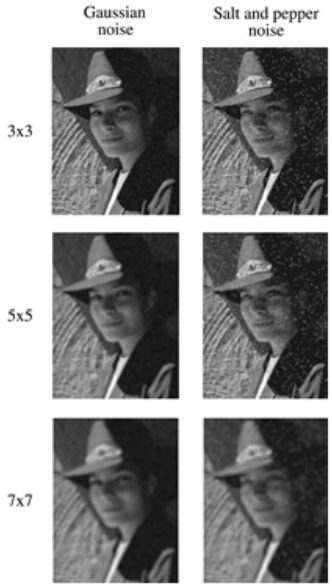
Q: What happens at the edges?

Mean filters

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



Effect of mean filters



Gaussian filters

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

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

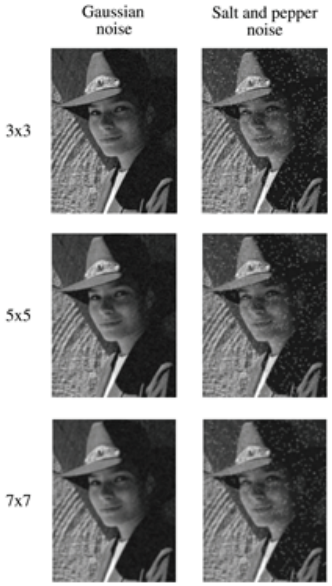
This does a decent job of blurring noise while preserving features of the image.

What parameter controls the width of the Gaussian?

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

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

Effect of Gaussian filters

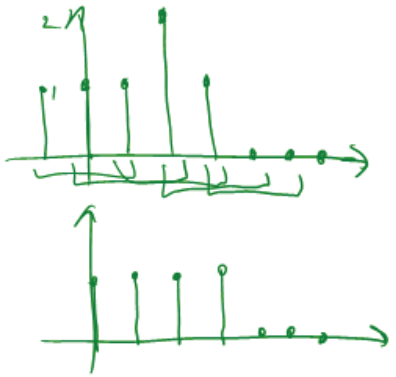


Median filters

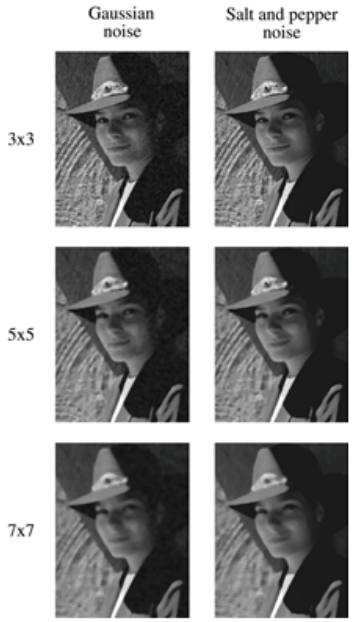
A **median filter** operates over an mxm region by selecting the median intensity in the region.

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

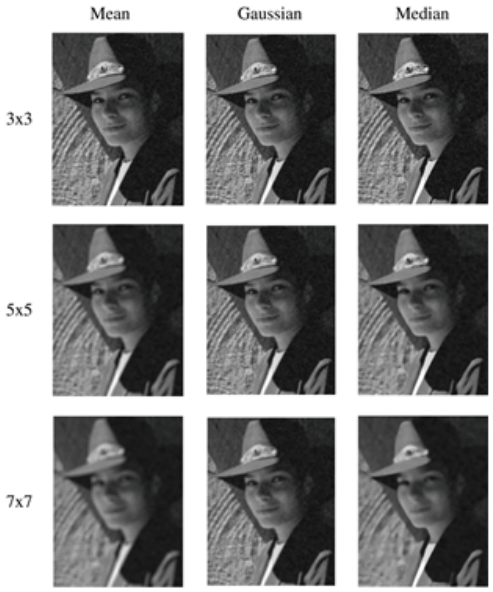
Is a median filter a kind of convolution? No



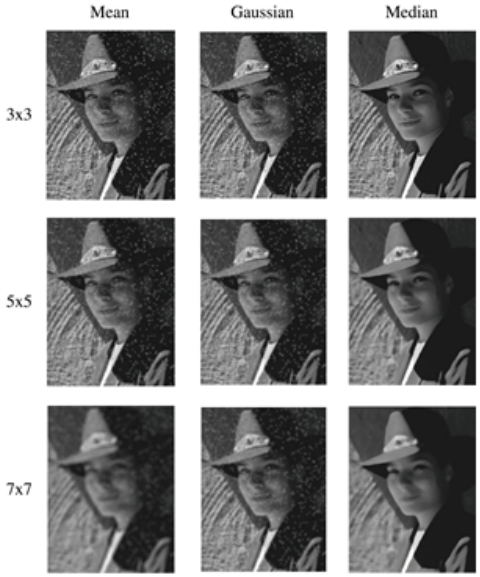
Effect of median filters



Comparison: Gaussian noise

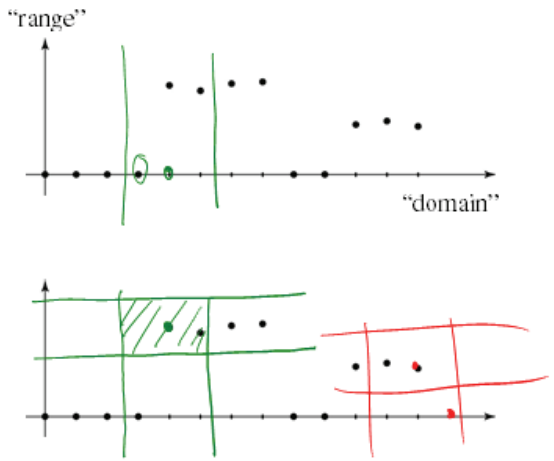


Comparison: salt and pepper noise



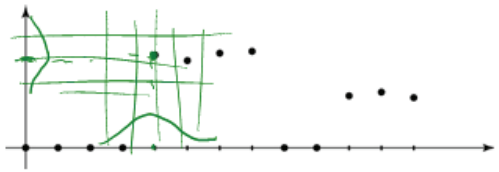
Bilateral filtering

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:

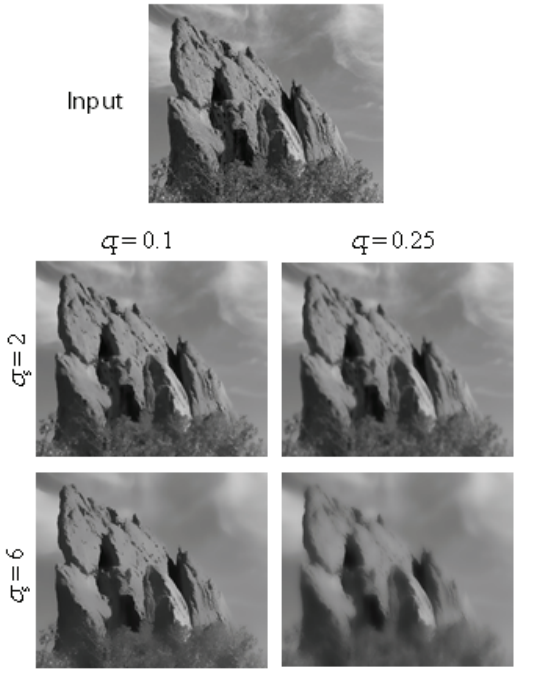
$$g[n] = \sum_{n'} f[n']h[n-n']$$

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

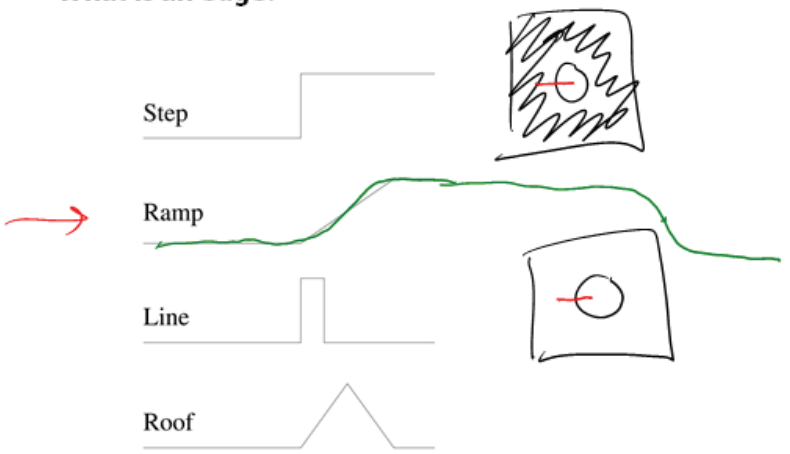
Edge detection

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?



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

$$\left|\frac{df}{dx}\right| > T$$

Gradients

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

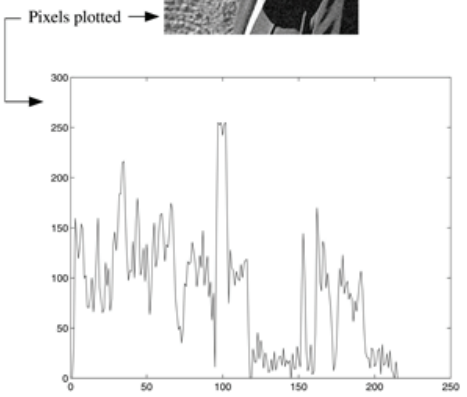
$$\nabla f(x,y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

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?

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 beyond pixels

Edge enhancement

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

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

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

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

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

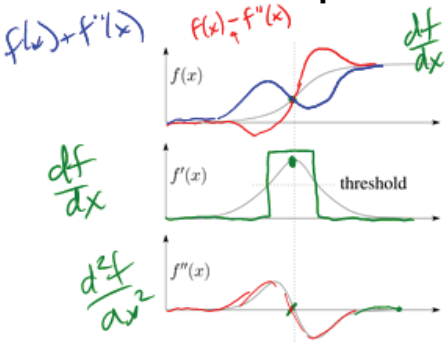

Sy + 128


Magnitude


Threshold = 64


Threshold = 128

Second derivative operators



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?

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

Localization with the Laplacian

A common measure of the second derivative in 2D is the **Laplacian**:

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

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

$$\Delta^2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

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

Zero crossings in a Laplacian filtered image correspond to localized edge positions.

Localization with the Laplacian

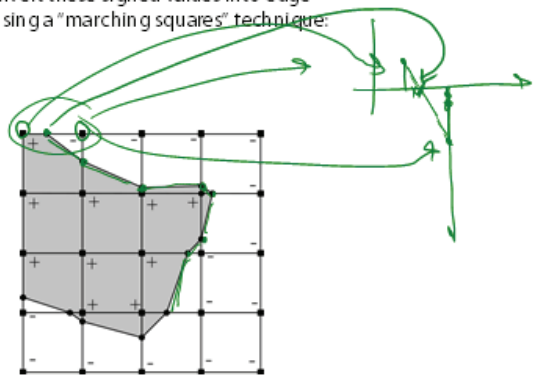

Original


Smoothed


Laplacian (+128)

Marching squares

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



Sharpening with the Laplacian

Original

Laplacian (+128)

Original + Laplacian

Original - Laplacian

Why does the sign make a difference?

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