Image Processing

Images by Pawan Sinha

Today's readings
- Forsyth & Ponce, chapters 8.1-8.2
  - http://www.cs.washington.edu/education/courses/490cv/02wi/readings/book-7-revised-a-indx.pdf

For Monday
- Watt, 10.3-10.4 (handout)
- Cipolla and Gee (handout)
  - supplemental: Forsyth, chapter 9

Announcements

- **HW0** due now
- **Project 0** is out today. Li will give a help-session in the last 10 minutes of class.
- If you don't have a CSE account
  - turn in your account forms right after class!
- If you want to add
  - go ahead and register in EE400B, SLN 8590
- If you don't have access to 228
  - talk to me after class today

All announcements are posted via the class webpage
You may also use the mailing list (cse490cv@cs) for questions

What's 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)
- typically only defined over a rectangle, with a finite range

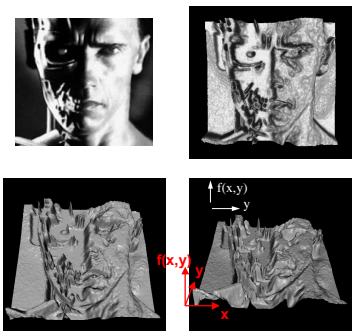
$$f : [a, b] \times [c, d] \rightarrow [0, 1]$$

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

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

scanalyze demo

Images as functions



What's a *digital* image?

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 *d* apart, we can write this as:

$$F[x, y] = \text{quantize}\{f(xd, yd)\}$$

Digital images (or just "images") are typically stored in a matrix
- Different coordinate systems (x,y) vs. (i=row,j=column)
- Helpful to use macros to convert when coding things up

j / i

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

scanalyze demo

Image processing

An **image processing** operation defines a new image g in terms of an existing image f.
We can transform either the domain or the range of f

Range transformation

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

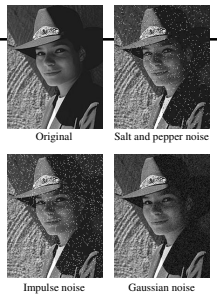
- Changes the color of the image
- Examples: contrast adjustment, RGB to grayscale

Domain transformation

$$g(x, y) = f(t_x(x, y), t_y(x, y))$$

- Changes the shape of the image
- Example: image warping (http://www.colonize.com/warp/)

Noise



Common types of noise:

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

Reducing Noise

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

Filtering

- look at the neighborhood N around each pixel
- replace each pixel with new value as a function of pixels in N

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

- The behavior of a filter depends on N and f

Mean filtering

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

$F[x,y]$ $\quad$ $G[x,y]$

Replace each pixel with an average of the pixels in the k×k box around it

- 3×3 case: $G[x,y] = \frac{1}{9}\sum_{u=0}^{2}\sum_{v=0}^{2} F[x+u-1, y+v-1]$

Mean filtering

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

	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	

$F[x,y]$ $\quad$ $G[x,y]$

Replace each pixel with an average of the pixels in the k×k box around it

- 3×3 case: $G[x,y] = \frac{1}{9}\sum_{u=0}^{2}\sum_{v=0}^{2} F[x+u-1, y+v-1]$

What about border pixels?

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

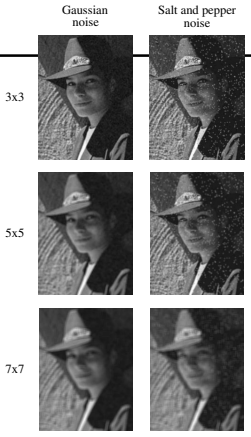
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$F[x,y]$ $\quad$ $G[x,y]$

Some options

- don't evaluate—image gets smaller each time a filter is applied
- pad the image with more rows and columns on the top, bottom, left, and right
  - option 1: copy the border pixels: add [0 0 0] to F in above case
  - option 2: reflect the image about the border: add [0 90 0] to F in above case

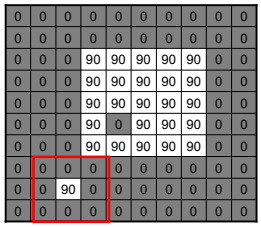
Effect of filter size



What happens if we use a larger mean filter?

5×5? 7×7?

Weighted average filtering



$F[x, y]$

$$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

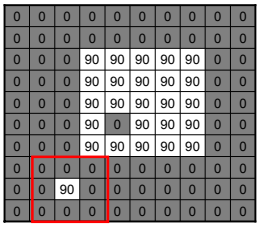
$H[u, v]$

Replace each pixel with a *weighted* average of the pixels in the k×k box

- 3×3 case: $G[x, y] = \sum_{u=0}^{2}\sum_{v=0}^{2} H[u, v] * F[x + u - 1, y + v - 1]$

Show filter demo

Gaussian filter



$F[x, y]$

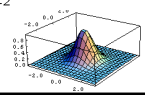
$$\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$$

$H[u, v]$

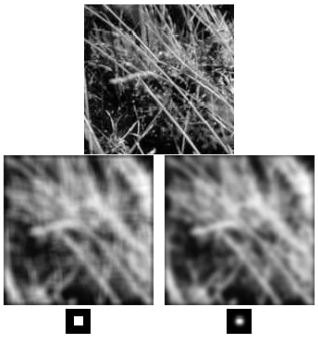
This filter H is a good approximation to $h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$

Properties of Gaussian

- more weight to the center
- good model of blurring in optical systems
- σ corresponds to width of the Gaussian
- more later...

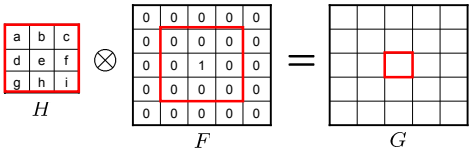


Comparison of mean vs. gaussian filter



Linear shift-invariant filters

What happens when a filter is applied to an impulse?



- helps explain why mean filters can give "blocky" results

Cross correlation

For a k×k filter, the equation becomes (assume k is odd):

$$G[x, y] = \sum_{u=0}^{k}\sum_{v=0}^{k} H[u, v] * F[x + u - \frac{k-1}{2}, y + v - \frac{k-1}{2}]$$

If we choose the origin of h, F, and G to be the center, we can simplify:

$$G[x, y] = \sum_{u}\sum_{v} H[u, v] * F[x + u, y + v]$$

This filtering operation is known as a **cross-correlation**, written

$$G = H \otimes F$$

For continuous images, cross-correlation is defined as:

$$\begin{aligned} g(x, y) &= (h \otimes f)(x, y) \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(u, v) f(x + u, y + v) du dv \end{aligned}$$

Similar to **convolution**:

- Convolution flips the filter horizontally and vertically before applying it
  - how does convolution with a Gaussian filter differ from cross-correlation?

Linear shift-invariant filters

Both cross-correlation and convolution filters have the following properties

- linear: $h \otimes (f_1 + f_2) = (h \otimes f_1) + (h \otimes f_2)$
  - how about scale? Is the following true?
    $h \otimes sf = s(h \otimes f)$
- shift-invariant: same operation applied for every x,y in f

Any filter with these properties (LSI) may be written as a cross-correlation

- Is this also true for convolution?

Convolution has nice properties in the *Fourier Domain*

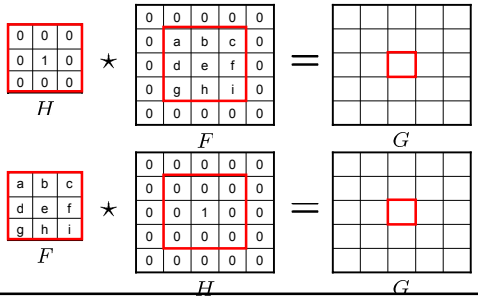
- we won't cover this, but see Forsyth Chap. 8.3 for a nice discussion

Symmetry in Convolutions

Convolutions have a symmetry property:

$$h \star f = f \star h$$

Try it out:



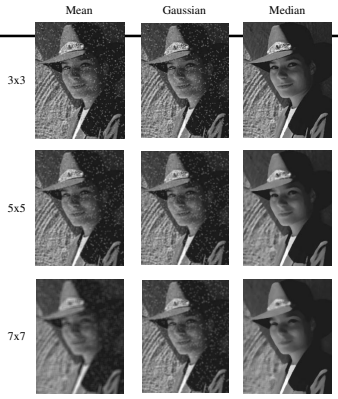
Median filter

A **median filter** operates over a k×k window by returning the median pixel value in that window

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

Can a median filter be implemented by a cross-correlation?

Filtering an image corrupted by salt and pepper noise



Filtering an image corrupted by Gaussian noise

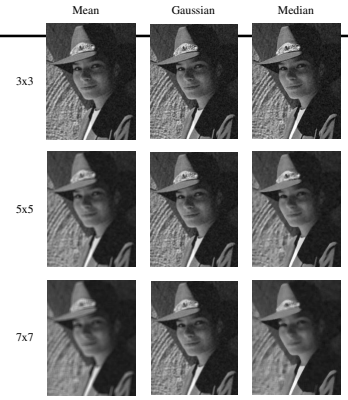


Image Scaling

This image is too big to fit on the screen. How can we reduce it?

How to generate a half-sized version?

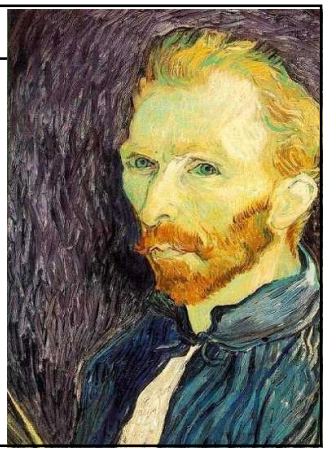
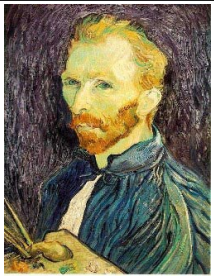


Image sub-sampling



1/4

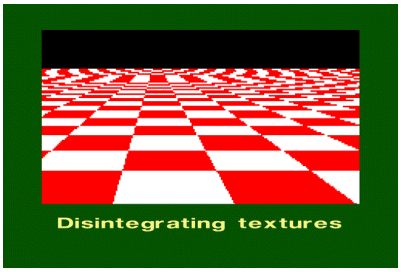
1/8

Throw away every other row and column to create a 1/2 size image

Why does this look so crufty?

- Called **nearest-neighbor** sampling

Even worse for synthetic images



Sampling and the Nyquist rate

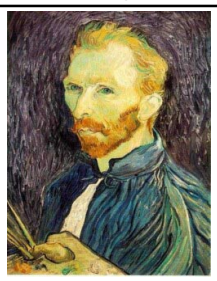
Aliasing can arise when you sample a continuous signal or image

- Demo applet
  http://www.cs.brown.edu/exploratory/research/applets/repository/nyquist/nyquist.html
- occurs when your sampling rate is not high enough to capture the amount of detail in your image
- formally, the image contains structure at different scales
  - called "frequencies" in the Fourier domain
- the sampling rate must be high enough to capture the highest frequency in the image

To avoid aliasing:

- sampling rate > 2 * max frequency in the image
  - i.e., need more than two samples per period
- This minimum sampling rate is called the **Nyquist rate**

Subsampling with Gaussian pre-filtering



Gaussian 1/2

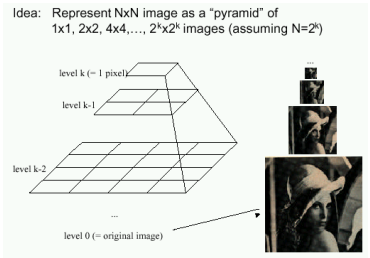
G 1/4

G 1/8

Solution: filter the image, *then* subsample

- Filter size should double for each ½ size reduction. Why?
- How can we speed this up?

Some times we want many resolutions



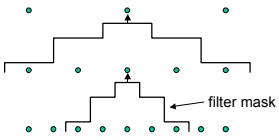
Known as a **Gaussian Pyramid** [Burt and Adelson, 1983]

- In computer graphics, a *mip map* [Williams, 1983]
- A precursor to *wavelet transform*

Gaussian Pyramids have all sorts of applications in computer vision

- We'll talk about these later in the course

Gaussian pyramid construction



Repeat

- Filter
- Subsample

Until minimum resolution reached

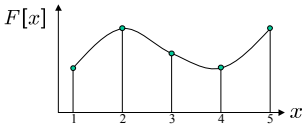
- can specify desired number of levels (e.g., 3-level pyramid)

The whole pyramid is only 4/3 the size of the original image!

Image resampling

So far, we considered only power-of-two subsampling

- What about arbitrary scale reduction?
- How can we increase the size of the image?



d = 1 in this example

Recall how a digital image is formed

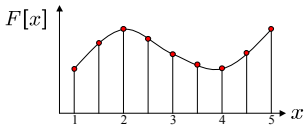
$$F[x, y] = \text{quantize}\{f(xd, yd)\}$$

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

Image resampling

So far, we considered only power-of-two subsampling

- What about arbitrary scale reduction?
- How can we increase the size of the image?



d = 1 in this example

Recall how a digital image is formed

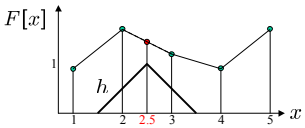
$$F[x, y] = \text{quantize}\{f(xd, yd)\}$$

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

Image resampling

So what to do if we don't know f

- Answer: guess an approximation $\tilde{f}$
- Can be done in a principled way: filtering



d = 1 in this example

Image reconstruction

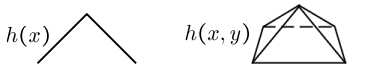
- Convert F to a continuous function

  $f_F(x) = F(\frac{x}{d})$ when $\frac{x}{d}$ is an integer, 0 otherwise
- Reconstruct by cross-correlation:

$$\tilde{f} = h \otimes f_F$$

Image resampling

What does the 2D version of this hat function look like?



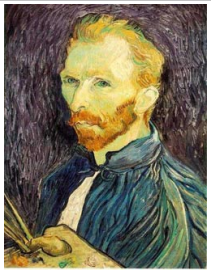
performs linear interpolation

performs **bilinear interpolation**

Better filters give better resampled images

- Bicubic is common choice
  - fit 3rd degree polynomial surface to pixels in neighborhood
  - can also be implemented by a convolution or cross-correlation

Subsampling with bilinear pre-filtering



Bilinear 1/2

BL 1/4

BL 1/8

Things to take away from this lecture

Things to take away from this lecture

- An image as a function
- Digital vs. continuous images
- Image transformation: range vs. domain
- Types of noise
- LSI filters
  - cross-correlation and convolution
  - properties of LSI filters
  - mean, Gaussian, bilinear filters
- Median filtering
- Image scaling
- Image resampling
- Aliasing
- Gaussian pyramids