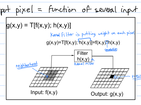
Neighborhood Operations

no LUT
use neighborhood kernel

neighborhood
use kernel
add them up
new pixel

output pixel = function of several input pixels



Spatial Filtering

- $h(x,y)$ = Kernel, Filter, Filter mask, or Window
- vals of filter are coefficients
- kernels are odd sizes: 3×3, 5×5, 7×7 . . .
  - why odd? want to create symmetry so centered on pixel

Filtering Process

- slide kernel left to right from top to bottom
- f_i = pixel grayvalue
- h_i = weights

The response of filter at each (x,y) pixel:

$R = h_1 f_1 + h_2 f_2 + \cdots + h_{mn} f_{mn}$ ← 1D indexing

$= \sum_{k=1}^{mn} h_k f_k$

← 2D indexing

Window centered at $(4,3)$:

$g_{43} = h_1 f_{32} + h_2 f_{33} + h_3 f_{34}$
$+ h_4 f_{42} + h_5 f_{43} + h_6 f_{44}$
$+ h_7 f_{52} + h_8 f_{53} + h_9 f_{54}$

Kernel slides across image in raster order

Convolution: flip kernel & slide across image

moving average: only window moves (used in finance)

suppose: add up all neighborhood pixel vals their weight on the center pixel

Linear Filtering

Linear Filtering: Image → put a 3×3 filter → (filter: image → add up pixel vals and ÷ by # of pixels and output it) → repeat

- let $f(x,y)$ = image ($M \times N$ size)
- let $h(i,j)$ = Filter Kernel ($m \times n$ size)

Linear filtering

$g(x,y) = \sum_{i=-s}^{s} \sum_{j=-t}^{t} h(i,j)\, f(x+i, y+j)$

(sum from $i=-s$ to s, sum from $j=-t$ to t)

where $s = (m-1)/2$ and $t = (n-1)/2$

- $h(i,j)$ = weight of mask at position (i,j)
- $f(x+i, y+j)$ = pixel

For completely filtered image: $x = 0, 1, \ldots, M-1$, $y = 0, 1, \ldots, N-1$

Spatial Averaging

- used for blurring and noise reduction
- output = avg of neighborhood pixels
  - → reduces "sharp" transitions of gray vals
- Smoothing: reduces noise in image (good)
  - blurs edges (bad)

(weights (edges) / random noise)

3×3 Smoothing Filters

- The constant $\frac{1}{\text{sum of vals in kernel}}$ multiplying the kernel

This is required to compute an average.

Same result as the filter:

Same as filter for each edge

Same as for the edges

all weights are the same

Box filter

Weighted average

The center is the most important and other pixels are inversely weighted as a function of their distance from the center of the mask. This reduces blurring in the smoothing process.

Smoothing

Box Filter (same weight) Unweighted

you done

Issue: notice not spread to edges! Guess if we reduce it? spread to edges!

Will make it smaller or . . .

Solution: make padding so that when output image will be same size!!!

will make it more smooth. If we keep refiltering → re-blurring . . .!!! make image smaller

Unweighted / Weighted Average

Unweighted averaging (smoothing filter):

$g(x,y) = \frac{1}{m} \sum_{i,j} f(i,j)$

just add up pixel vals in kernel and divide by kernel size

Weighted averaging:

$g(x,y) = \sum f(i,j) h(x-i, y-j)$

$f(x,y)$ = image ($M \times N$ size)
$h(i,j)$ = Filter kernel (mxn size)

If you keep smoothing it will slowly turn to

Blurring

Original image | 7x7 unweighted averaging | 7x7 Gaussian filter (weighted blur)

all pixels same blur

blur depending on weight

(a) Unweighted Averaging

- Unweighted Averaging over 5 pixel neighborhood along horizontal scanline
  - padding (5-1)/2 = 2 of padding on each side of row

```
for(x=2; x<w-2; x++)
    out[x]=(in[x-2]+in[x-1]+in[x]+in[x+1]+in[x+2])/5;
```

(2 add up 5 horizontal pixels, 3 divide)

- each output pixel needs 5 pixel access (4 adds, 1 divide)
- this is a simpler unweighted average

```
sum=in[0]+in[1]+in[2]+in[3]+in[4];
for(x=2; x<w-2; x++){
    out[x] = sum/5;
    sum+=(in[x+3] - in[x-2]);
}
```

Sum up 5 pixel vals

Limited excursions reduce size of output

pixel should be!

must subtract move sum to new pixel

Sum of 0 to 4: 422

- x=3: out[3] = 422/5 = 84
  sum = 422 + (111 − 100) = 433
- x=4: out[3] = 433/5 = 86.6
  sum = 433 + (60 − 50) = 443

in[x+3] = 111, in[x−3] = 100
in[x+3] = 60, in[x−2] = 50

(b) Convolution

$(I * k_1) * k_2$ ← (# of pixels, # of pixels)

$= I * (k_1 * k_2)$

$= I * k_2$

Convolution of the neighborhoods

Padded with zeros for horizontal vertical

1/9 1/9 1/9 | 1/9 1/9 1/9 | 1/9 1/9 1/9

1/9 + 1/9 = 2/9 | = 3/9 | = 2/9

1/9, 2/9, 3/9, 2/9, 1/9

this is a triangle

Instagram feed

3 pixels in 1/3

pixel

IF you keep convolving result with your will get gaussian wave . . .

1/9 1/9 1/9

convolving the triangle with a box again

Comparing increases by 1 for 3×3 comparison

remember this is just horizontally . . . needs do it vertically too

weights

1/27 3/27 6/27 7/27 6/27 3/27 1/27

1/27 3/27 6/27 7/27 6/27 3/27 1/27

1/27 3/27 6/27 7/27 6/27 3/27 1/27

3/27

Image Averaging

- Smoothing / Blur filter
- Sharpen

$g(x,y)$ = noisy image
$f(x,y)$ = original image
$\eta(x,y)$ = noise

$$g(x,y) = f(x,y) + \eta(x,y)$$ ← original image + noise

Idea: If you got a ton of noisy images (same image taken over and over) → can get rid of noise by adding up all images and dividing by total # of images added!

e.g. take 100 pics of the sky (don't move camera).
The pics will have some random noise
If we add all 100 pics up and ÷ by 100 → will get better pic

$$\bar{g}(x,y) = \frac{1}{K}\sum_{i=1}^{K} g_i(x,y)$$ ← if noise has 0 mean and uncorrelated, averaging lots of different noisy pictures

∴ as $K \uparrow$ the noise of each pixel at location $(x,y) \downarrow$ (assuming all pics are aligned)

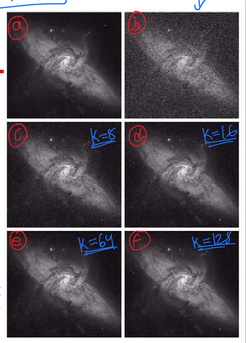
Noise Reduction (1)

So we take lots of the same picture to reduce the noise (8, 16, 64, 128 of the same pic)

Image corrupted Lots of noise

- Astronomy is an important application of image averaging.
- Low light levels cause sensor noise to render single images virtually useless for analysis.

C to F: results of averaging 8, 16, 64, and 128 noisy images

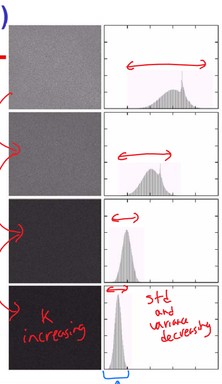


a b
c d
e f
FIGURE 3.30 (a) Image of Galaxy Pair NGC 3314. (b) Image corrupted by additive Gaussian noise with zero mean and a standard deviation of 64 gray levels. (c)–(f) Results of averaging $K = 8$, 16, 64, and 128 noisy images. (Original image courtesy of NASA.)

Noise Reduction (2)

- Difference images and their histograms yield better appreciation of noise reduction.
- Notice that the mean and standard deviation of the difference images decrease as K increases.

as K increases,
· stand deviation and mean decreases



a b
FIGURE 3.31 (a) Image from top to bottom: Difference images between Fig. 3.30(a) and the four images in Figs. 3.30(c) through (f), respectively. (b) Corresponding histograms.

smaller error

narrow so more confined error

Smoothing Mask: General Form

General Form: Smoothing Mask

- Filter of size $m \times n$ (where m and n are odd)

$$g(x,y) = \frac{\sum_{i=-s}^{s}\sum_{j=-t}^{t} h(i,j)\, f(x+i, y+j)}{\sum_{i=-s}^{s}\sum_{j=-t}^{t} h(i,j)}$$

summation of all coefficients of the mask

Note that s = (m-1)/2 and t = (n-1)/2

if weights don't add up to 1 this part will make it weight 1?
e.g. for 1/3 1/3 1/3 weigh → 1=1 so don't need this

from Linear Filtering

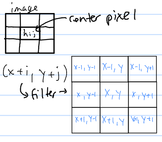
Linear Filtering

$$g(x,y) = \sum_{i=-s}^{s}\sum_{j=-t}^{t} h(i,j)\, f(x+i, y+j)$$

(kernel filter offsets from (0,0) pixels)

where s = (m-1)/2 and t = (n-1)/2

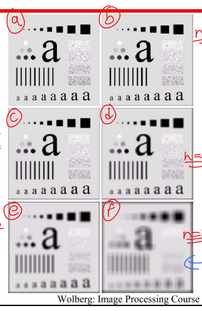
$f(x,y)$ = image (M×N size)
$h(i,j)$ = Filter Kernal (m×n size)



Smoothing = blurring

Result of Smoothing w/ Square Avg Filter of size n = 3, 5, 9, 15, 35

a	b
c	d
e	f



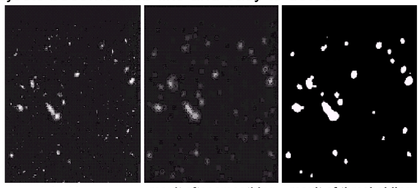
- a) original image 500x500 pixel
- b) - f) results of smoothing with square averaging filter of size n = 3, 5, 9, 15 and 35, respectively.
- Note:
  - big mask is used to eliminate small objects from an image.
  - the size of the mask establishes the relative size of the objects that will be blended with the background.

bigger neighborhood = larger blur

Wolberg: Image Processing Course Notes

Example

- Blur to get gross representation of objects.
- Intensity of smaller objects blend with background.
- Larger objects become blob-like and easy to detect.



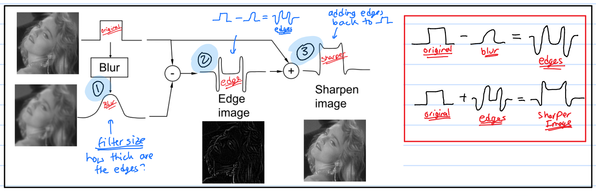
original image | result after smoothing with 15x15 filter | result of thresholding

Unsharp Masking

code!

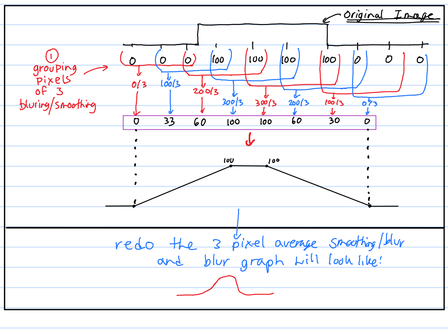
← can use blurring to make image sharper!

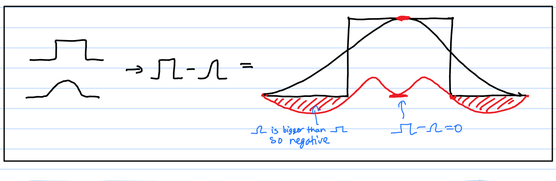
- smoothing → it affects the regions where grayscales transition from one val to another
- subtraction isolates these edge regions
- adding the edges back → causes edges to look sharper!



① Blur =

② Edge Graph =





③ Add original (⎍) and edge (∿) images

Code
① get blurred Images (size of filter given)
② Edges Image = (Original − Blurred) × factor (how thick are the lines?)
③ Sharpened Image = (Original + Edges)

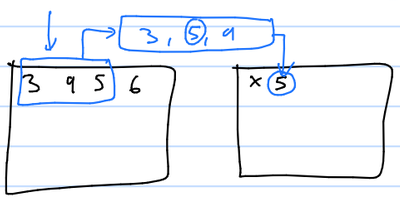
Median filters

Wake up 9
11
12
2

Order-Statistics Filters

- non-linear filter → its response is based on ranking the pixels in the pixel support

∴ move a 3×3 window → collect median of neighborhood → output it



Median Filter → used to rid noise

1) sort (using quicksort algos) neighborhood pxls in increasing order
2) replace neighborhood center with median values

Note: window doesn't need to be a square
Eliminate intensity spikes: with salt+pepper noise

Median filter used to REDUCE NOISE!

- When you pick a neighborhood, noise is usually @ beginning or end so picking median keeps you safe!
- BUT, if neighborhood is also noisy → only option is to border neighborhood

median

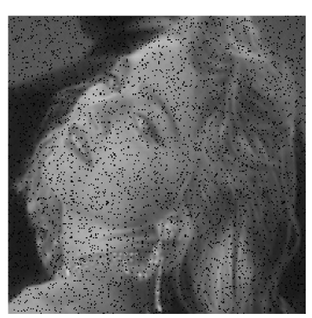
10	18	19
20	200	15
25	20	25

(10,15,18,19,20,20,25,25,200)

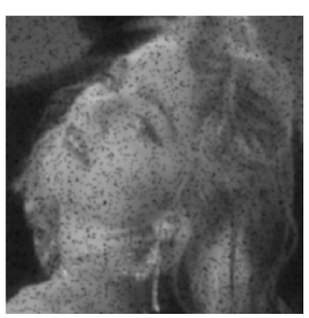
k=1: replace 200 with (19+20+20)/3
k=2: replace 200 with (18+19+20+20+25)/5
k=3: replace 200 with (15+18+19+20+20+25+25)/7
k=4: replace 200 with (10+15+18+19+20+20+25+25+200)/9

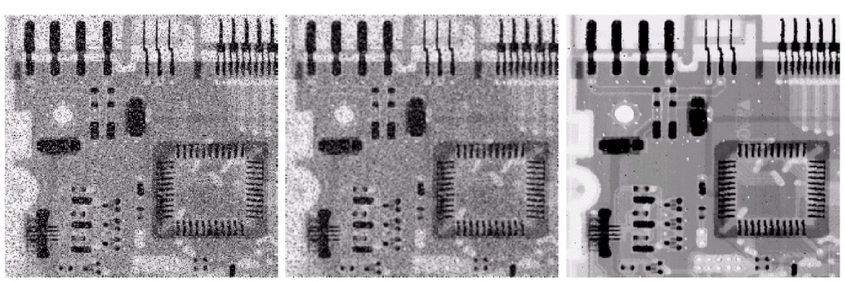
① sort neighborhood
② replace neighborhood center with median filter that's focused on the median val (20 here) and elems ±(k) away from center

Salt + Pepper


Additive salt & pepper noise


Median filter output


Blurring output



a b c

FIGURE 3.37 (a) X-ray image of circuit board corrupted by salt-and-pepper noise. (b) Noise reduction with a 3 × 3 averaging mask. (c) Noise reduction with a 3 × 3 median filter. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

Derivative Operators:
- enhances edges
- deemphasizes areas where gray lvls vary

Circle = what this filter/mask does

First Order Derivatives	Second Order Derivatives
• Constant grayval areas → 0	• Constant grayval areas → 0
• onset of grayval step/ramp → nonzero	• onset of grayval step/ramp → nonzero
• along ramps → nonzero	• along ramps → 0
$\frac{\partial f(x)}{\partial x} = f(x+1) - f(x)$ (right pixel, current pixel)	$\frac{\partial^2 f(x)}{\partial x^2} = \partial f(x) - \partial f(x-1) = f(x+1) + f(x-1) - 2f(x)$ (right pixel, left pixel, 2· current pixel)
• produce thicker lines	• strong response to fine details (thin lines, isolated points)
• strong response to graylevel steps	• double response @ step changes in graylvl

Example

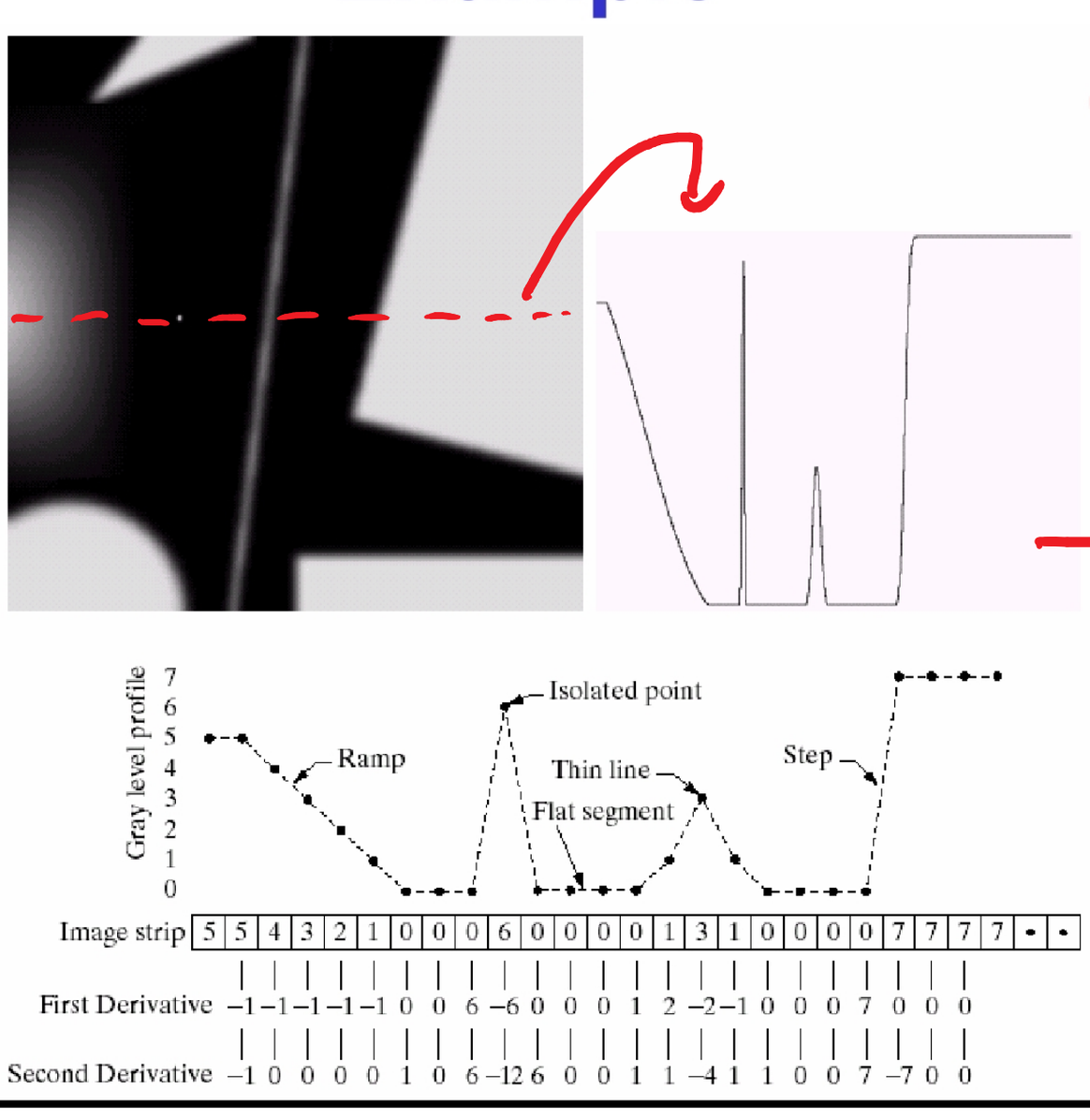


FIGURE 3.38 (a) A simple image. (b) 1-D horizontal gray-level profile along the center of the image and including the isolated noise point. (c) Simplified profile (the points are joined by dashed lines to simplify interpretation).

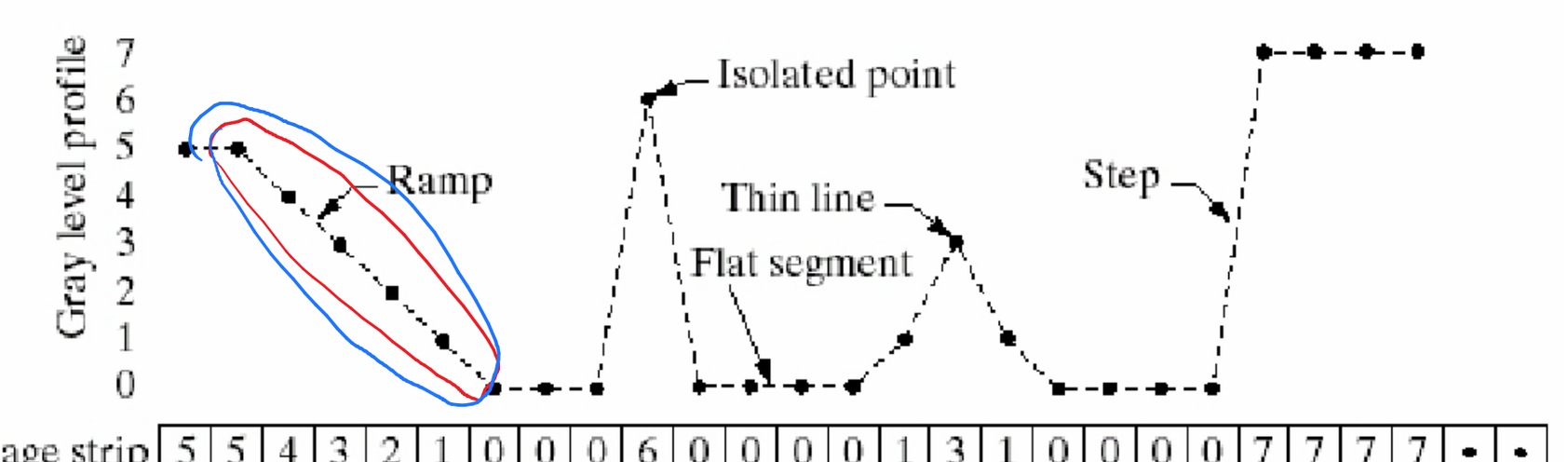


Image strip	5	5	4	3	2	1	0	0	0	6	0	0	0	0	1	3	1	0	0	0	0	7	7	7	7
First Derivative (right pxl) − (pixel)	−1	−1	−1	−1	−1	0	0	6	−6	0	0	0	1	2	−2	−1	0	0	0	7	0	0	0		
Second Derivative (left pxl) + (right pxl) − 2(pxl)	−1	0	0	0	0	1	0	6	−12	6	0	0	1	1	−4	1	1	0	0	7	−7	0	0		

1st Deriv: along ramp = non zero
2nd Deriv: along ramp = 0

Laplacian Operator
- simplist isotropic derivative operator
- rotation-invariant
- linear operator

Eq.

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

$$\frac{\partial^2 f}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y)$$
$$\frac{\partial^2 f}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y)$$

$$\nabla^2 f = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)]$$

Left Pixel, Right Pixel, Top Pixel, Bottom Pixel, 4·Curr pixel

Eq of Laplacian Filter

Laplacian 90° Masks

	1	
1	-4	1
	1	

or

	-1	
-1	4	-1
	-1	

Laplacian Mask

corners are 0 in 90°, −4×center
corners are 1 in 45°, −8×center

Isotropic result for rotations in increments of 90°			Isotropic result for rotations in increments of 45°		
0	1	0	1	1	1
1	−4	1	1	−8	1
0	1	0	1	1	1
0	−1	0	−1	−1	−1
−1	4	−1	−1	8	−1
0	−1	0	−1	−1	−1

Some people don't like negative @ center so flip

FIGURE 3.39 (a) Filter mask used to implement the digital Laplacian, as defined in Eq. (3.7-4). (b) Mask used to implement an extension of this equation that includes the diagonal neighbors. (c) and (d) Two other implementations of the Laplacian.

Another Derivation

1/9*

1	1	1
1	1	1
1	1	1

Unweighted Average Smoothing Filter

0	0	0
0	1	0
0	0	0

Retain Original — original image

1/9*

-1	-1	-1
-1	8	-1
-1	-1	-1

Original – Average (negative of Laplacian Operator) — like sharpening: original − blurred = sharpened edge; "edge detection"

In constant areas: 0 ⟵ **Summation of coefficients in masks equals 0.** $8 + (-1 + -1 + -1 + -1 + -1 + -1 + -1 + -1) = 0$

Near edges: high values

Example

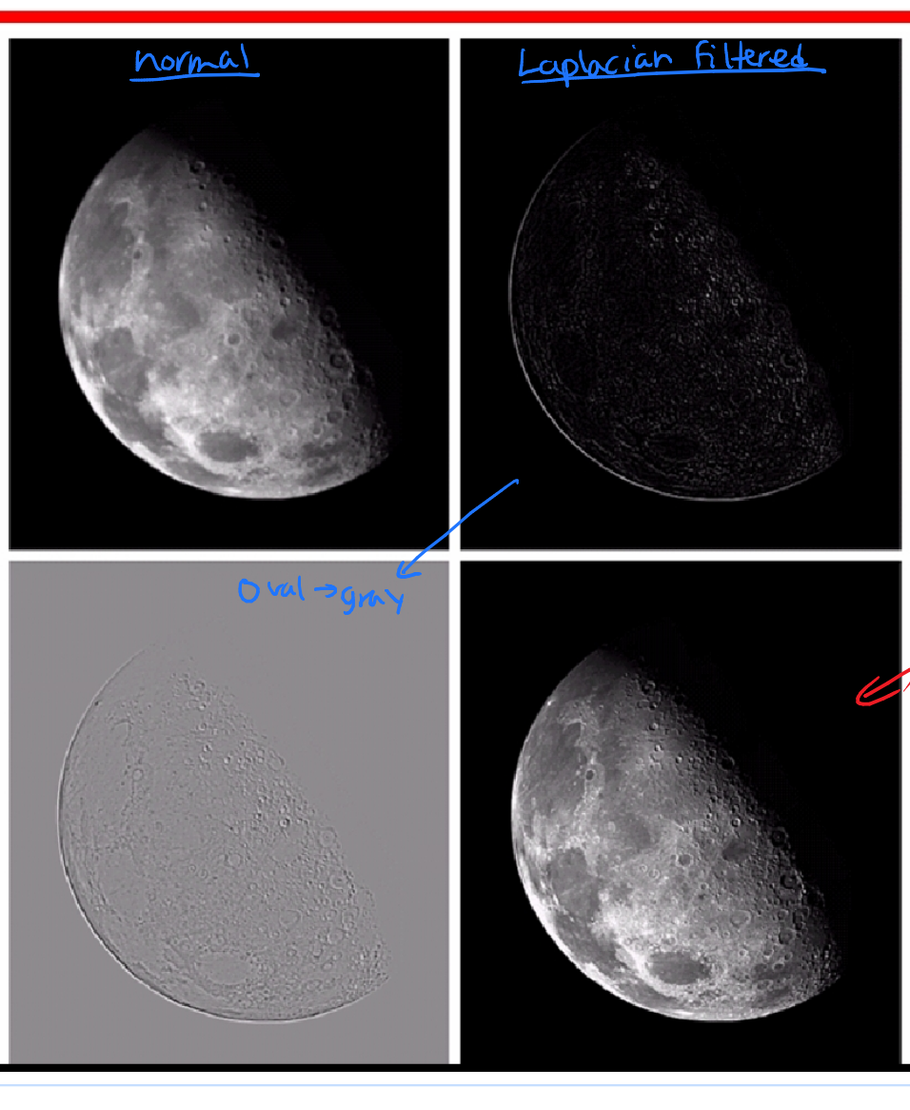


FIGURE 3.40 (a) Image of the North Pole of the moon. (b) Laplacian-filtered image. (c) Laplacian image scaled for display purposes. (d) Image enhanced by using Eq. (3.7-5). (Original image courtesy of NASA.)

45° Laplacian Mask

-1	-1	-1
-1	8	-1
-1	-1	-1

WOW Looks Sharper!!

Effects of Laplace Operator
- Since its a derivative operator: it highlights edges only (sharp spikes)
  - → highlights graylevel discontinities (sharp vals)
  - → deemphaizes areas with slowly varying gray levels (goes to 0 → black)
- produce images that has gray edge lines on a black background (bc)

Simplification

Adding an image + Laplacian can be combined into:

original

0	0	0
0	1	0
0	0	0

+ Laplacial Filter

0	-1	0
-1	4	-1
0	-1	0

=

0	-1	0
-1	5	-1
0	-1	0

$$g(x,y) = f(x,y) - [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)]$$
$$= 5f(x,y) - [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)]$$

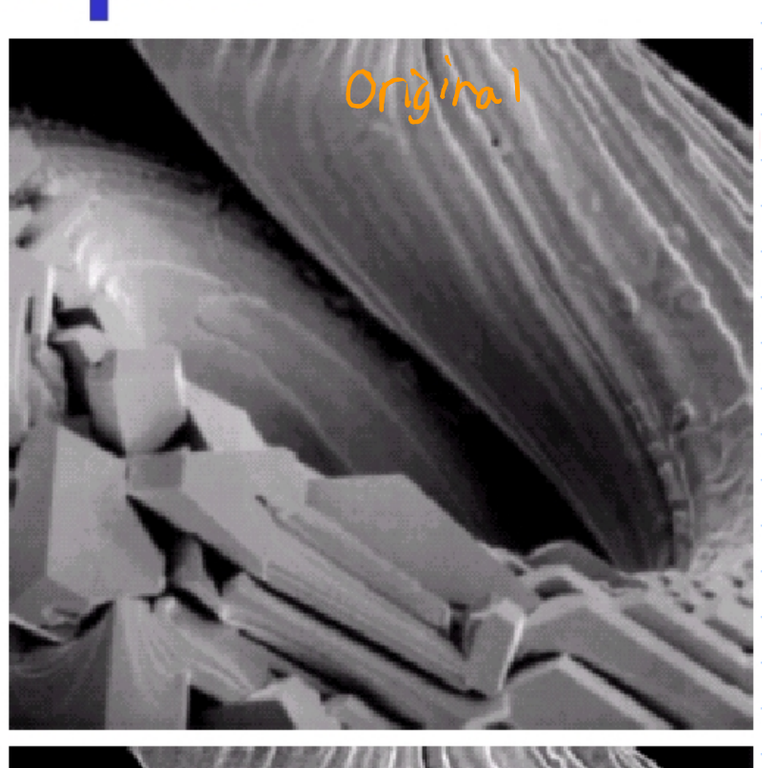
Original Image + Laplacian Filter

Example

0	−1	0
−1	5	−1
0	−1	0

−1	−1	−1
−1	9	−1
−1	−1	−1

Laplacian is 4. 4+1=5
Laplacian is 8. 8+1=9 → Sharper mask



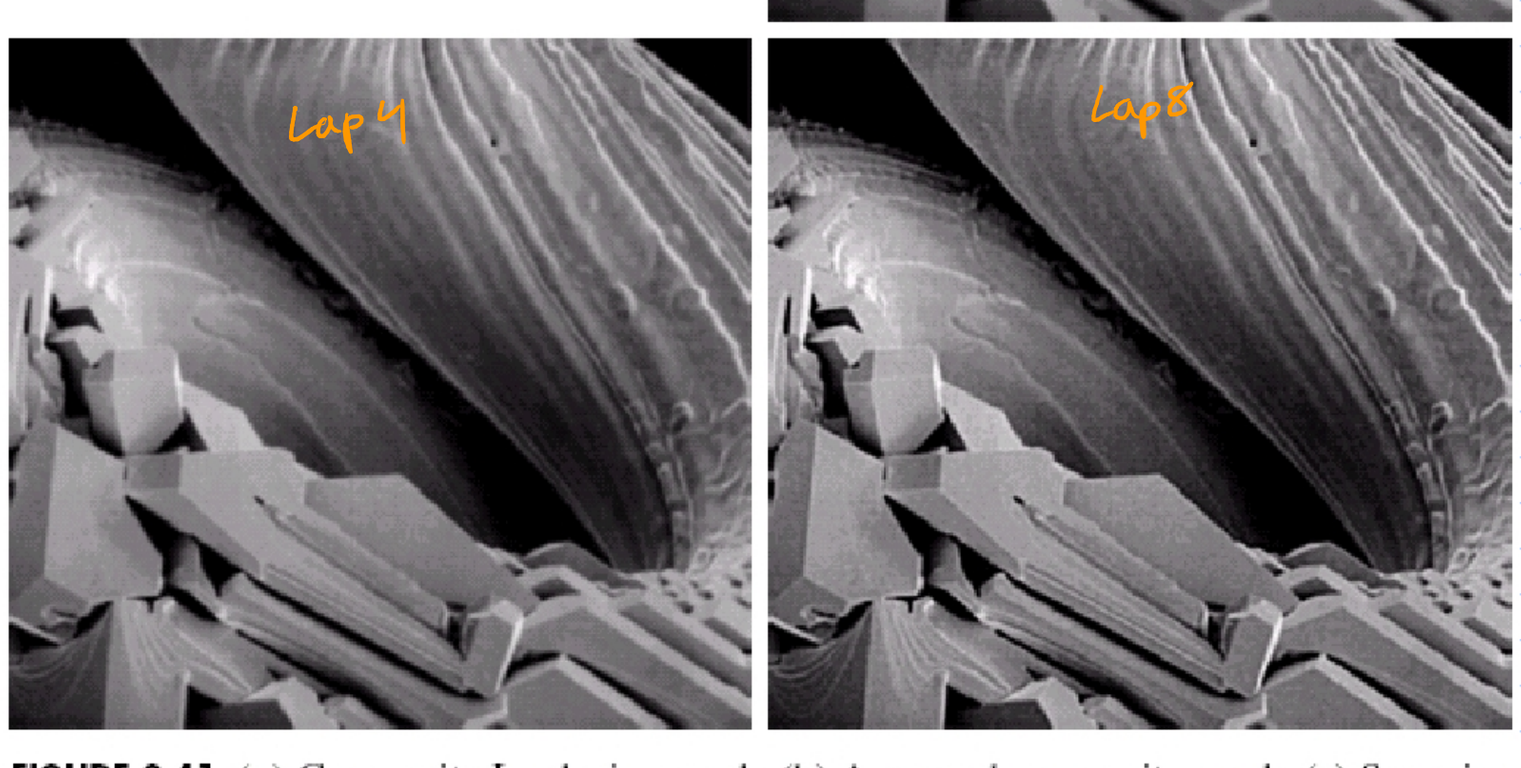


FIGURE 3.41 (a) Composite Laplacian mask. (b) A second composite mask. (c) Scanning electron microscope image. (d) and (e) Results of filtering with the masks in (a) and (b), respectively. Note how much sharper (e) is than (d). (Original image courtesy of Mr. Michael Shaffer, Department of Geological Sciences, University of Oregon, Eugene.)

Gradient Operator
← Unlike Laplacian (scalar) → gives both magnitude and direction

- gradient = vector of directional derivatives

$$\nabla \mathbf{f} = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

- magnitude of gradient vector ≈ (called) gradient

$$\nabla f = mag(\nabla f) = \sqrt{(f_x)^2 + (f_y)^2} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

approximation: $\nabla f \approx |f_x| + |f_y|$

$f_{mag}(i,j) = \sqrt{f_x^2 + f_y^2}$ or $f_{mag}(i,j) = |f_x| + |f_y|$

$f_{angle}(i,j) = \tan^{-1}\frac{f_y}{f_x}$ where $f_x(i,j) = f(i+1,j) - f(i-1,j)$, $f_y(i,j) = f(i,j+1) - f(i,j-1)$

f_x = right − left pixel; f_y = top − bottom pixel

- The components of the gradient vector are linear operators, but the magnitude is not (square, square root).
- The partial derivatives are not rotation invariant (isotropic), but the magnitude is.
- The Laplacian operator yields a scalar: a single number indicating edge strength at point.
- The gradient is actually a vector from which we can compute edge magnitude and direction. (gives mag + angle)

Summary
- components of gradient vector = linear operators
  - magnitude = not linear bc of √
- partial derivatives → not rotation invariant (isotrophic)
  - magnitude → is rotation invariant
- Laplacian operator → yeilds scalar (single # indicating edge strength @ point)
- gradient → a vector → can compute edge magnitude + direction (angle + magnitude)

Summary (1)

Continuous	Digital
$f(x)$	$v(i)$
$f'(x)$	$v'(i) = v(i) - v(i-1)$
$f''(x) = \nabla^2 f(x)$	$v''(i) = v'(i) - v'(i-1)$

$$= [v(i) - v(i-1)] - [v(i-1) - v(i-2)]$$
$$= v(i-2) - 2v(i-1) + v(i)$$
$$= (1 \ \ -2 \ \ 1)(v(i-2) \ \ v(i-1) \ \ v(i))$$ → centered @ (i−1)
$$= (1 \ \ -2 \ \ 1)(v(i-1) \ \ v(i) \ \ v(i+1))$$ → centered @ i

	-1	
-1	4	-1
	-1	

-1	-1	-1
-1	8	-1
-1	-1	-1

The Laplacian is a scalar, giving only the magnitude about the change in pixel values at a point. The gradient gives both magnitude and direction.

Summary (2)

Find mag and angle of image:

One dimensional: $mask_x = [-1 \ \ 0 \ \ 1]$, $mask_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$

(2) put results in vector

Twodimensional:

SobelOperator: $mask_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ $mask_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

PrewittOperator: $mask_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ $mask_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

(1) use both masks to find edges of image

(3) find mag + angle

$f_{mag}(i,j) = \sqrt{f_x^2 + f_y^2}$ or $f_{mag}(i,j) = |f_x| + |f_y|$

$f_{angle}(i,j) = \tan^{-1}\frac{f_y}{f_x}$

in → Gradient → Mag, Angle
in → Laplacian → Mag

Gradient makes mag + phase while Laplacian only mag

ex

0	0	0
100	100	100
100	100	100

← horizontal edge (found by going ↕)

0	100	100
0	100	100
0	100	100

vertical edge (found by going ↔)

Produce new images: input image → f_x, f_y → Make new image: magnitude, phase

Example (1)

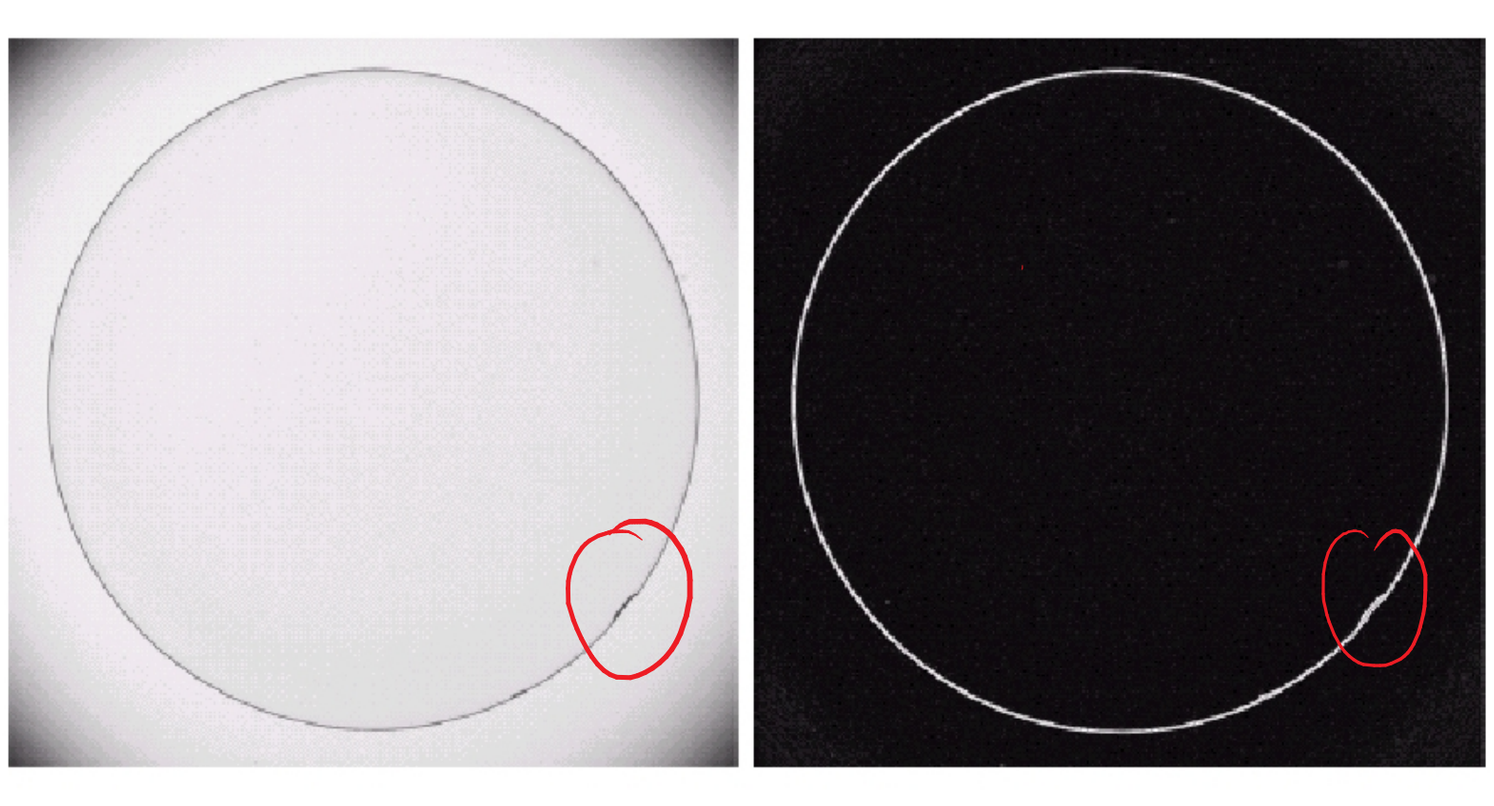


FIGURE 3.45 Optical image of contact lens (note defects on the boundary at 4 and 5 o'clock). (b) Sobel gradient. (Original image courtesy of Mr. Pete Sites, Perceptics Corporation.)

Example (2)

- **Goal:** sharpen image and bring out more skeletal detail.
- **Problem:** narrow dynamic range and high noise content makes the image difficult to enhance.
- **Solution:**
  1. Apply Laplacian operator to highlight fine detail
  2. Apply gradient operator to enhance prominent edges
  3. Apply graylevel transformation to increase dynamic range

1) Laplacian → highlight fine detail
2) gradient → enhance edges
3) graylvl transformation to increase ranges

Original; Laplacian: 1) non extreme grays → 0 2) 0 to gray background

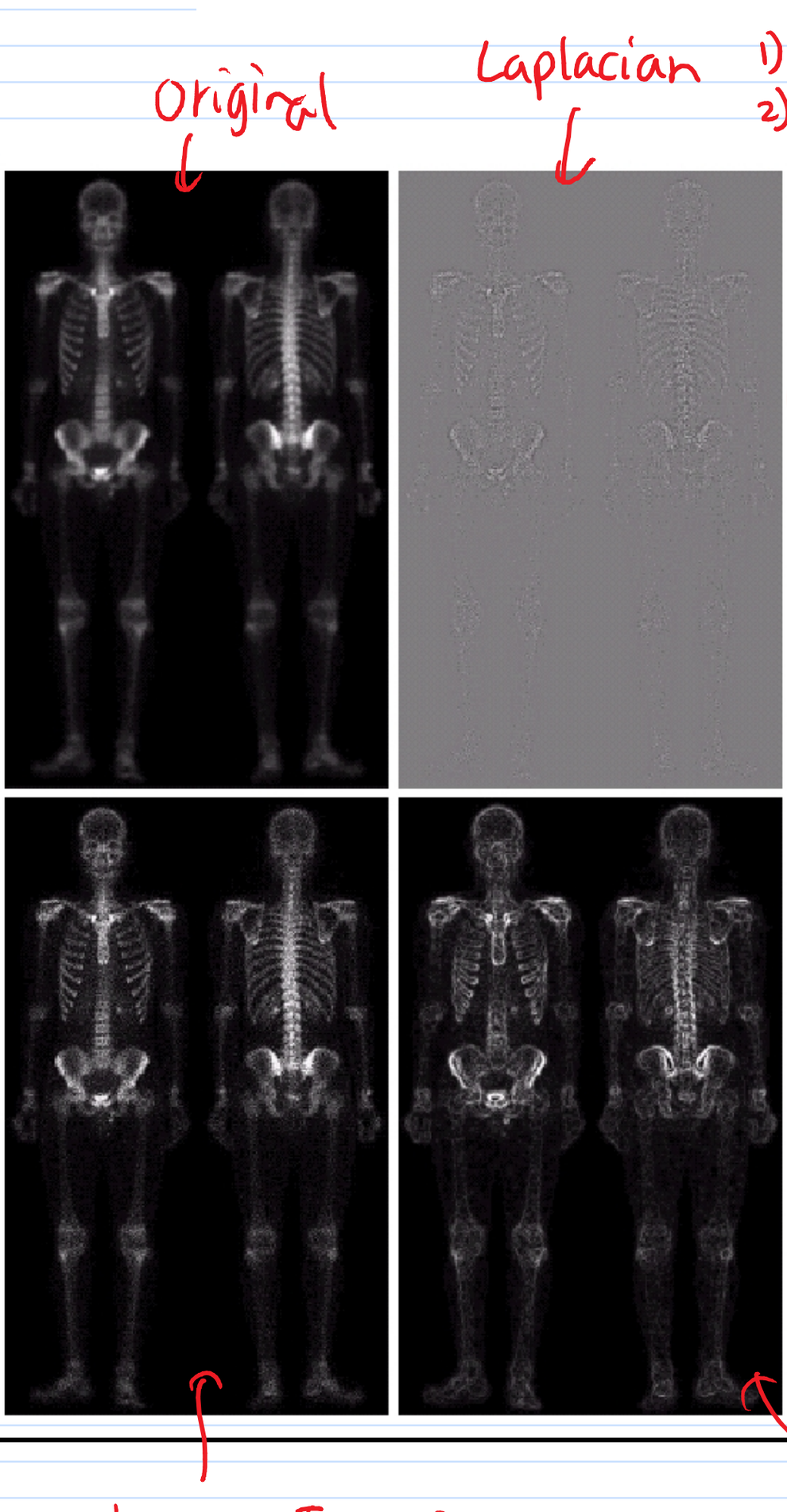


FIGURE 3.46 (a) Image of whole body bone scan. (b) Laplacian of (a). (c) Sharpened image obtained by adding (a) and (b). (d) Sobel of (a).

Sharper Image (original + Laplacian version); another one.

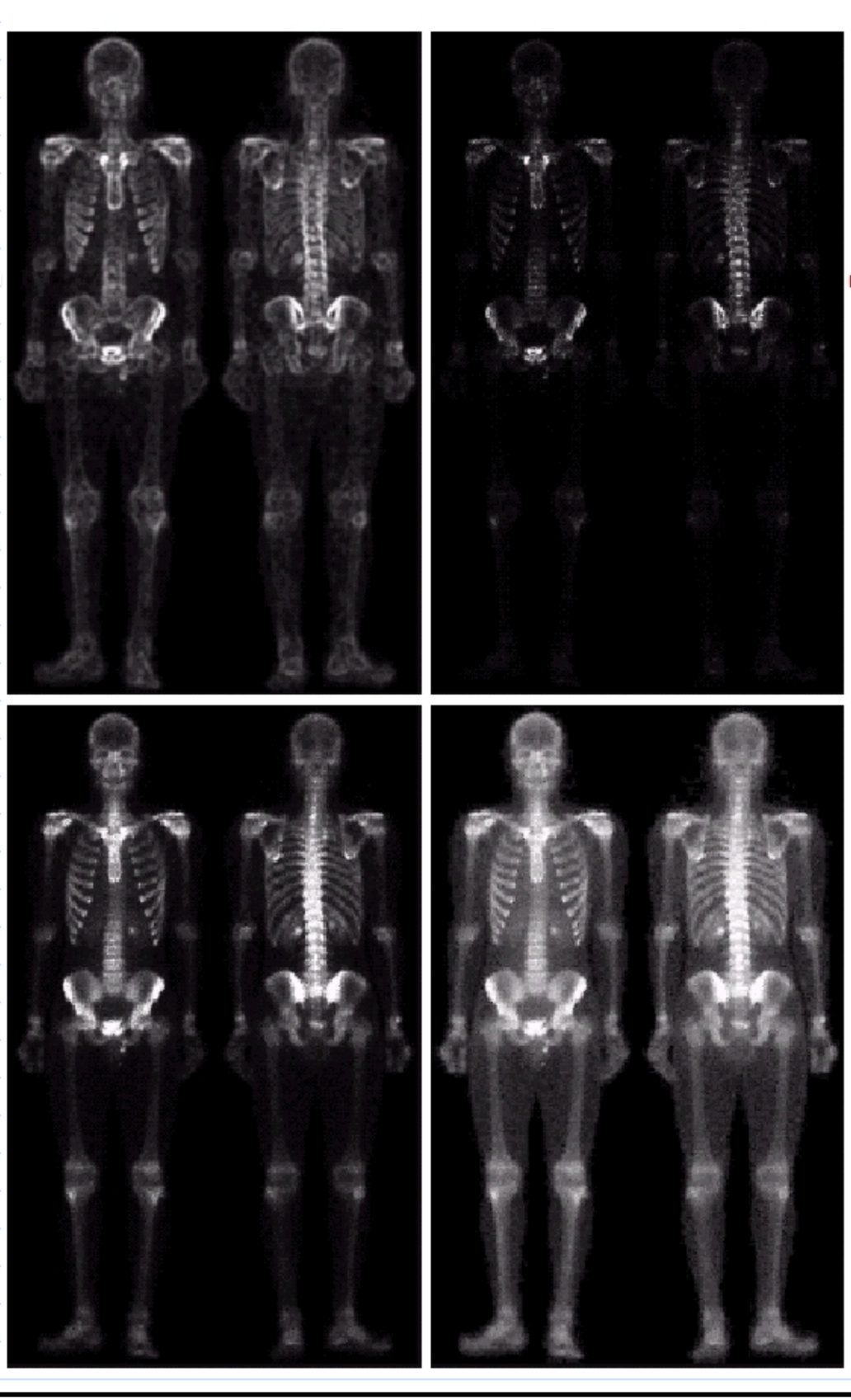
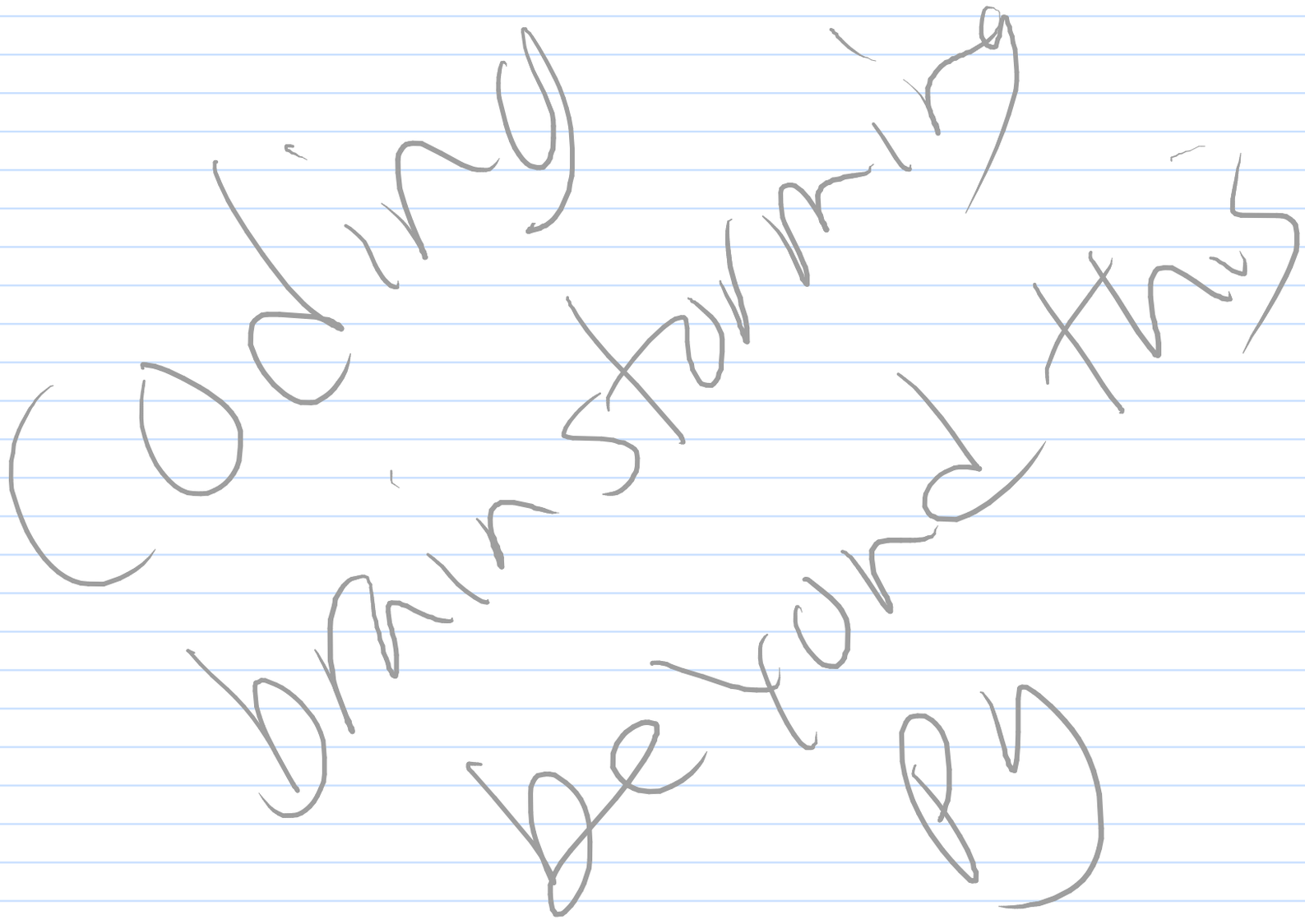
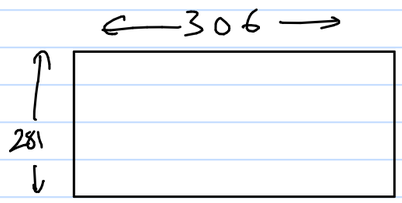


FIGURE 3.46 (Continued) (e) Sobel image smoothed with a 5 × 5 averaging filter. (f) Mask image formed by the product of (c) and (e). (g) Sharpened image obtained by the sum of (a) and (f). (h) Final result obtained by applying a power-law transformation to (g). Compare (g) and (h) with (a). (Original image courtesy of G.E. Medical Systems.)


Coding
brainstorming
be turned this
By

Coding brainstorming blur



w = 306
h = 281
filter = 5x5
pad = 2 on sides

vector

0	1	2	3	4	5	6	...	305	306	307	308	309
~~10~~	~~20~~	30	40	50	60	70						
		0	1	2	3	4	...	303	304	305	306	307

size = 310

loop index 1 to 305
paddingNumberN to < vector.size() - paddingNumberN
310 - 2 = 308

expect: 36 @ 1st ind

horizontal blur working

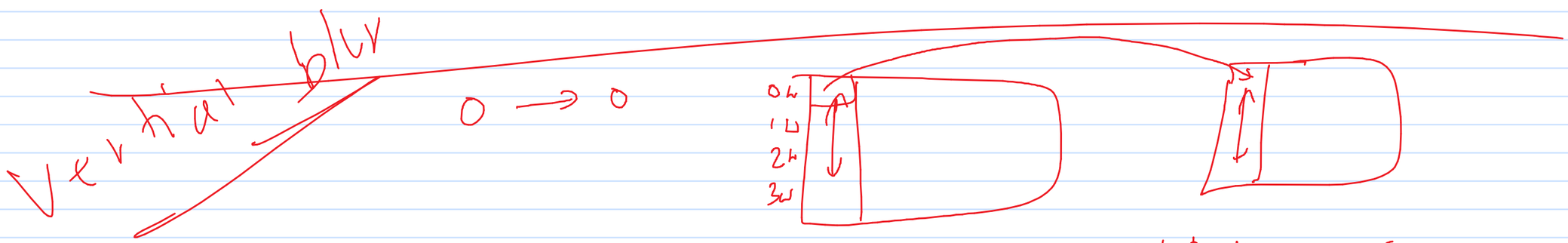
```
for (win= 2 , to  < 308 , win++){

    sum=0
    sum += vector(win)

    for (i=~~0~~ 1 to  <=2, i++) {
        s+= vector(win - i)
        s+= vector(win+i)
    }
```

win = 2
s = 0 set
s = 30 ↩

i=~~0~~ 1 sum [1] [s]
s = 30 + 30 + 40 = 100

i=2 sum + [0] + [4]
s = 100 + 30 + 50 = 180

180/5 = 36 ✓



col ↕ then row ↔

9x8 image

N = 9
col = 2
row = 3
row.N ⇒ 27

col offset

for (coloffset = 0 to <= col

row offset

row * N *

	0	1	2	3	4	5	6	7	8
0			~~0~~	x	x	x	x	x	x
1	x	x	x	x	x	x	x	x	x
2	x	x	x	x	x	x	x	x	x
3	x	x	~~x~~						
4									
5									
6									
7									

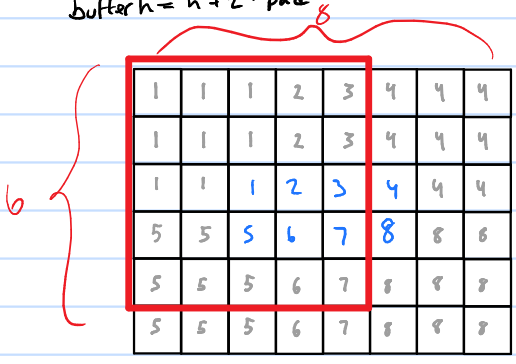
median coding

5x5 pixel → (5-1)/2 = 2 padd around

pad = 2

bufferW = w + 2·pad

bufferh = h + 2·pad



pad = 2

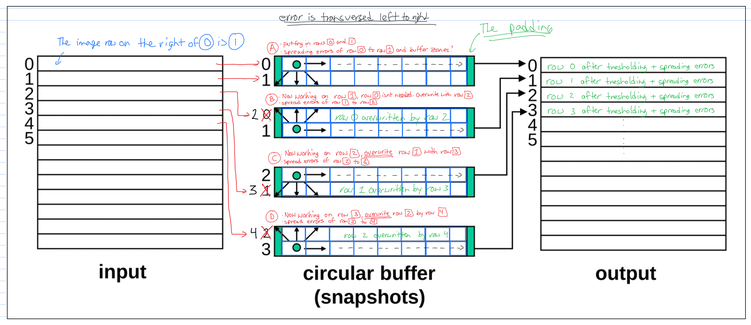
bufferW = w + 2·pad

bufferh =

vector ← padder row

```
for row 0 to <height:
    for col 0 to <weight:
        int pixel = *in++;
        if (row == 0 and pad != 0):
            for paddedTop=0 to < pad:
                for paddedTopCol=0 to < bufferW
                    vector.push_back(pixel);
```

median with 0 buffer



(better) Better Implementation Code.

buf: an array of 2 pointers that points to 2 memory locations

```
thr = MXGRAY /2;                // init threshold value
copyRowToCircBuffer(0);         // copy row 0 to circular buffer
for(y=0; y<h; y++){             // visit all input rows
    copyRowToCircBuffer(y+1);   // copy next row to circ buffer
    in1 = buf[  y  %2] + 1;     // circ buffer ptr; skip over pad
    in2 = buf[(y+1)%2] + 1;     // circ buffer ptr; skip over pad
    for(x=0; x<w; x++) {        // visit all input cols
        *out = (*in1 < thr)? BLACK : WHITE;   // threshold

        e = *in1 - *out;        // eval error
        in1[ 1] +=(e*7/16.);    // add error to E  nbr
        in2[-1] +=(e*3/16.);    // add error to SW nbr
        in2[ 0] +=(e*5/16.);    // add error to S  nbr
        in2[ 1] +=(e*1/16.);    // add error to SE nbr

        in1++; in2++            // advance circ buffer ptrs
        out++;                  // advance output ptr
    }
}
```

Copy row 0 to circular buffer

If I iterate by rows (no more rows and col)

1) copy next row to buffer

2) input pixel to start with

buf[row] returns same memory address

buf[0or1] +1 Skips the padding

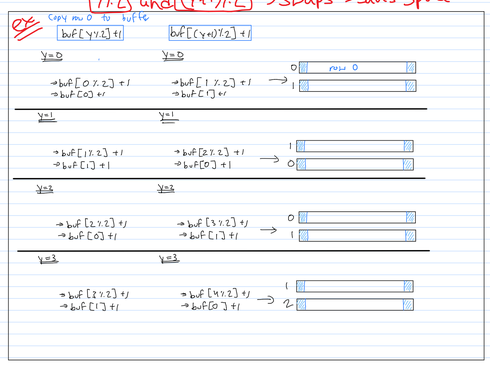
∴ in1 or in2 is the 1st pixel of that row!

The buf[y%2] and buf[(y+1)%2] also swaps

error diffusion

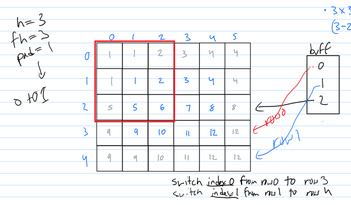
y%2 and (y+1)%2 → circular swaps → saves space

ex: copy row 0 to buffer



∴ in1 and in2 swaps every y increment → overwriting!

h=3
fh=3
pad=1



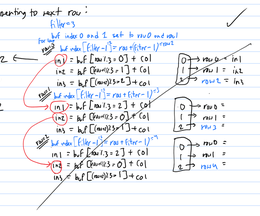
switch index 0 from row 0 to row 3
switch index1 from row1 to row h

filter=3
- say a (3×4) matrix
- 3×3 filter window
- (3-2)/2 = (1 padd) → 5×6

buff size = filter size

// add rows to the buffer (copy entire row + padding)

// when incrementing to next row:



0%3 = 0
1%3 = 3/3 =1
2%3 = 6/3 =2
3/3=0
4%3=1
5%3=2

run this from row=0 to row <=(h-filter-1)

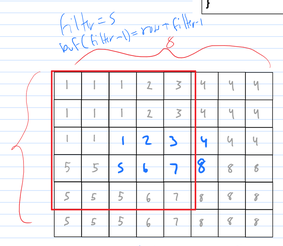
switch index 0 from row 0 to row 3 ← buf[row] = copyRow(row+filter)

row+(filter)

switch index1 from row1 to row h ← buf[row] = copyRow(row+filter)

row+(filter)

filter=5
buf[filter-1] = row+filter-1

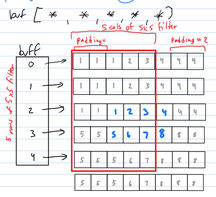


< N - filter = 8 - 5 = 3

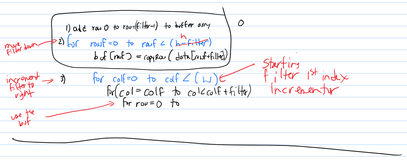
Say 5×5 window
padding Num = (5-1)/2 = 2

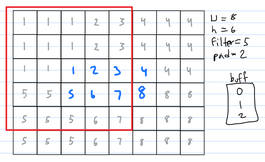
buf [*, *, *, *, *, *]

Scale of 5×5 filter



w=4
h=2
filter=5×5
pad=2





1) copy the first filter-1 rows from data to buff! ← fill up the initial filter

for bro=0 to bro < filter ← fill up the vector 2D

- for col=0 to col < w
  - if bro == filter: break ← break bc array done (will break anyway)
- If bro=0: ← 1st row copy to buff index 0 to pad
  - int pixel(=*in++);
  - if col==0: for padi=0 to padi<=padd buff[bro].push_back(pixel) ← Add the left padding + the 1st pxl
  - if col==w-1: for padi=0 to padi<=pad buff[bro].push_back(pixel) ← add the right padding + last pxl
  - else: buff[bro].push_back(pixel) ← add middle pixel

← padded row copied

copy row=0 to index 0 to pad

- for copyRow=1 to copyRow<=pad buff[copyRow].push_back(buff[0]) bro++;

filled up the buff with initial rows

2) kernal thru the 3 rows

3)

1) copy the first filter-1 rows from data to buff! ← fill up the initial filter

for bro=0 to bro < filter ← fill up the vector 2D

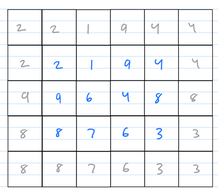
- for col=0 to col < w
  - if bro == filter: break ← break bc array done
- If bro=0: ← 1st row copy to buff index 0 to pad
  - int pixel(=*in++);
  - if col==0: for padi=0 to padi<=padd buff[bro].push_back(pixel) ← Add the left padding + the 1st pxl
  - if col==w-1: for padi=0 to padi<=pad buff[bro].push_back(pixel) ← add the right padding + last pxl
  - else: buff[bro].push_back(pixel) ← add middle pixel

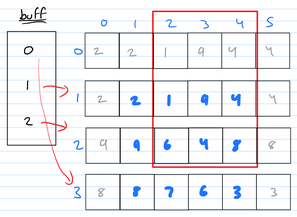
← padded row copied

copy row=0 to index 0 to pad

- for copyRow=1 to copyRow<=pad buff[copyRow].push_back(buff[0]) bro++;

w=4 filter=3×3
h=3 padd=1





Window goes down 0 to h-1

buff[0,1,2] } copy r3 to i0
[3,1,2] } copy r4 to i1
[3,4,2]

K=4
3 4 5

2 2 2 2 2 2 6 1 4

1/2 ← medial
1/2 ← med+1
2 6 ← +2
1 4 ← +3
1 4

K=0
0 1 2 3

error @ y inc = 4
in proc code: h=5 → error
error is in the last line of code when I switch rows

0 1 2
3 1 2
3 4 2
3 4 5

I see error: need a way to go back to overwrite index 0 (row 3)
need to use buffers!!!
that was why he used pointer?