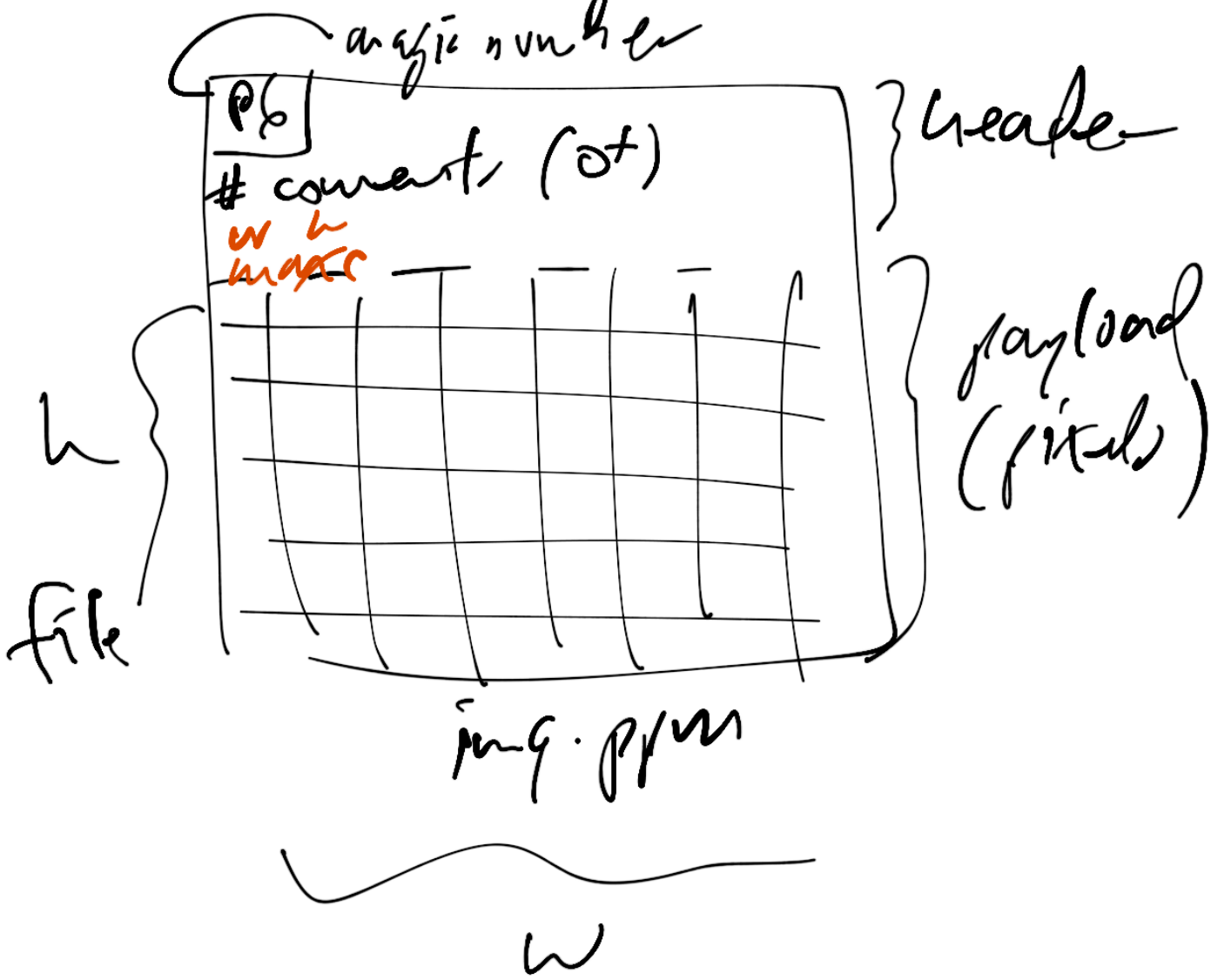
What is an image?



w x h pixel matrix
(2D array)

this array pixel

Header:

P6

comment (0^+ of these)

w h

maxc (max color)

→ when parsing header, read char by char

getc

ungetc

P6 PPM binary

color : 3 channel

P5 PGM binary

greyscale : 1 channel

P4 PBM binary

binary : 1 channel (0/1)

b/w image

P1, P2, P3 } ASCII encoded

images (not binary)

we use P6, P5

use fread, fwrite

to read, write 1 byte

(unsigned char) at a

time.

	P5	P6
image :	header	header
	g g g ...	r g b r g b ...
	1 channel per pixel (1 fread)	3 channels per pixel (3 fread)

when reading the g (greyscale) uchar

or the r, g, b (color) uchar

a) cast to float } [0,255]

b) divide by (float) maxc

255

normalize pixel channel val to [0,1] save as float

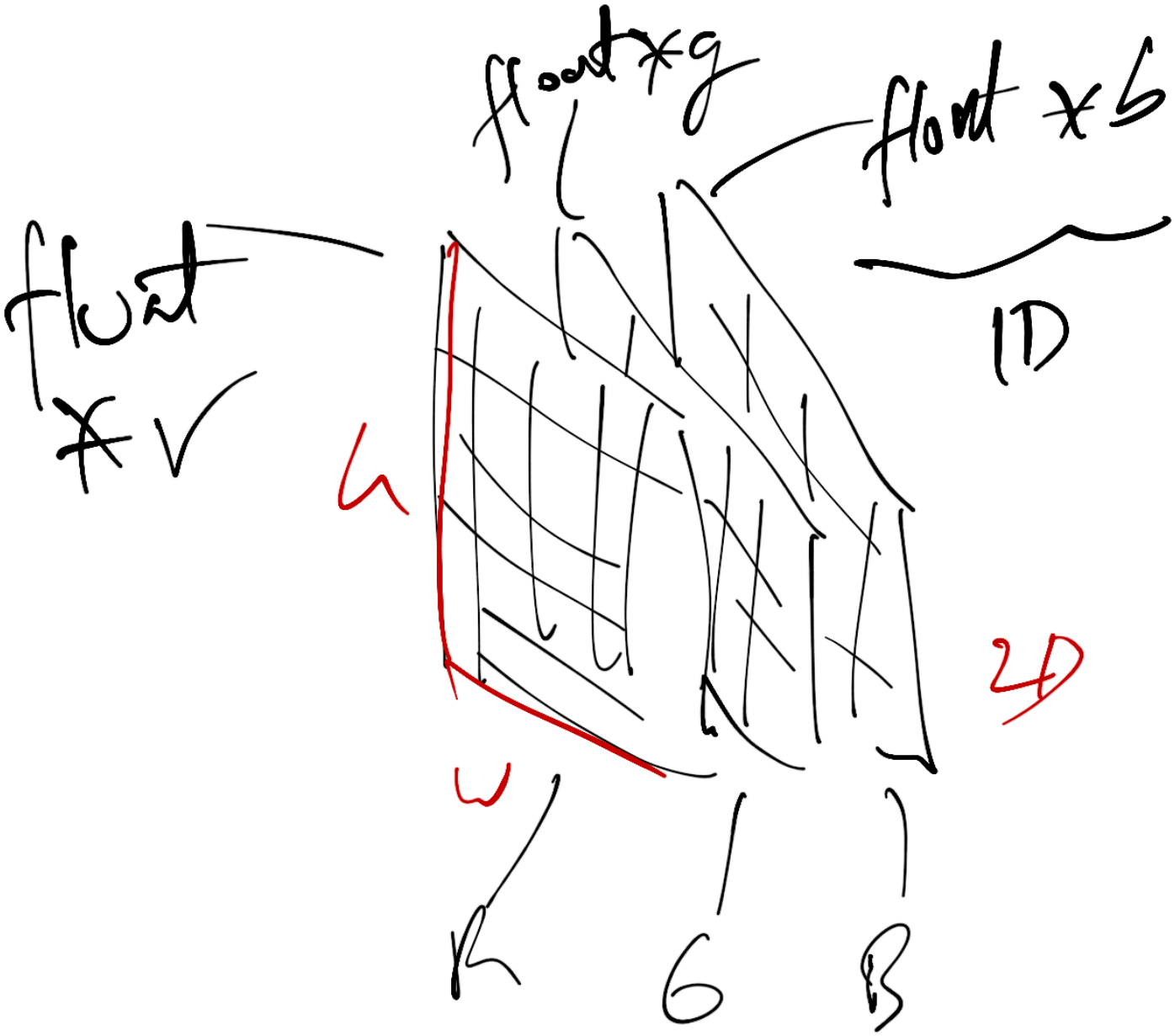


image planes

1 1D array per channel.

pgm_read
pgm_alloc(rows, cols)
pgm_write
pgm_free

pgm.h
pgm.c

---

ppm_read
ppm_alloc
ppm_write
ppm_free

ppm.h
ppm.c

asgd:

ppm → pgm

r, g, b → gray

$r/3 + g/3 + b/3 = g$

in pgm or ppm?

ppm extends pgm class

(oop terminology)

asq01:

$$I_g(x,y) = \frac{1}{3} I_r(x,y) + \frac{1}{3} I_g(x,y) + \frac{1}{3} I_b(x,y)$$

$I(x,y)$ or $\boxed{I(i,j)}$

$$(f * g)(i) = \sum_{j=1}^{m} g(i)\, f\left(i - j + \frac{m}{2}\right)$$

1D function

filter

$g(i) = \boxed{0 \mid 1 \mid .5}$ $\boxed{.5 \mid 1 \mid 0}$

$f(i) = \boxed{1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7}$ . . .

$\boxed{.5 \mid 1 \mid 0}$

$\boxed{.5 \mid 1 \mid 0}$

$\boxed{.5 \mid 1 \mid 0}$ ... $\boxed{\quad 11 \quad}$

0
1
2.5
4
5.5
7
8.5
10

Numpy: 'save'
'full'
'valid'

separability of filters:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}_{\text{tensor product}}$$

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

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

Sobel filter

dx

g_x

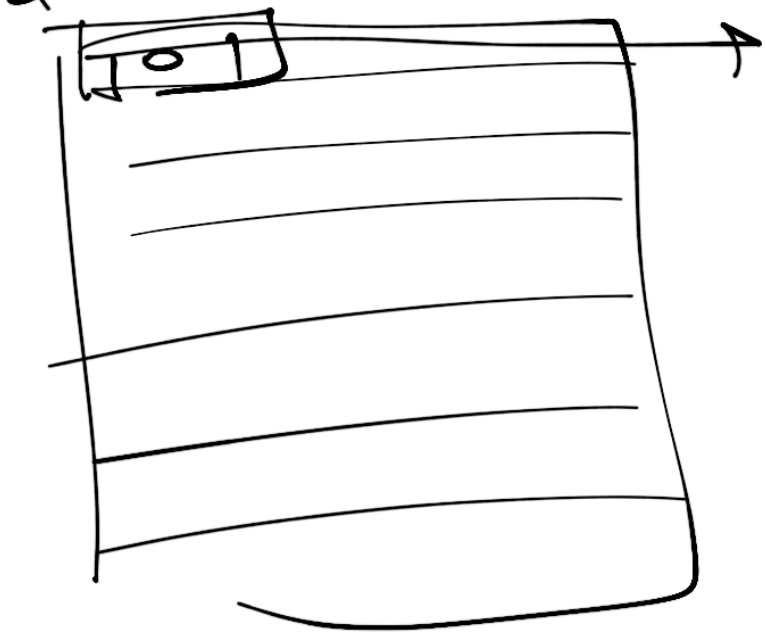
1	0	-1

dy

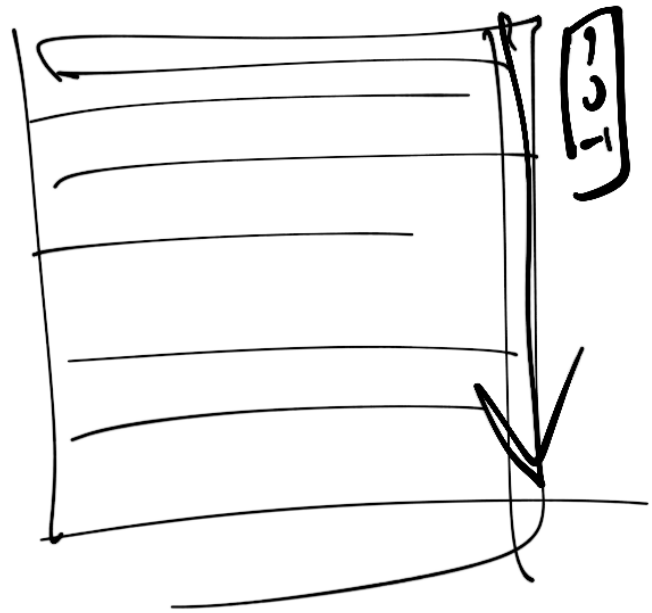
g_y

1
0
-1

dx

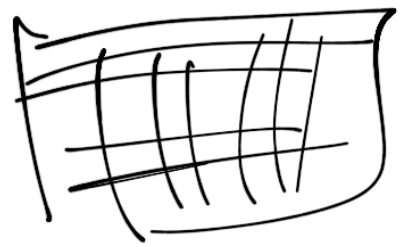


I



$I * g_x$

$(g_x)^2 + (g_y)^2$



Sobel filters

2 2D filter

1	0	-1
2	0	-2
1	0	-1

$g_x = dx$

1	2	1
0	0	0
-1	-2	-1

$g_y = dy$

$$r_x(\vec{x}) = g_x(\vec{x}) * I \quad \Big\} \; dx/dt$$

image

$$r_y(\vec{x}) = g_y(\vec{x}) * I \quad \Big\} \; dy/dt$$

2 image

gradient : 3rd image

$$\vec{\nabla} r(\vec{x}) = \begin{pmatrix} r_x(\vec{x}) \\ r_y(\vec{b}) \end{pmatrix}$$

∇ : gradient

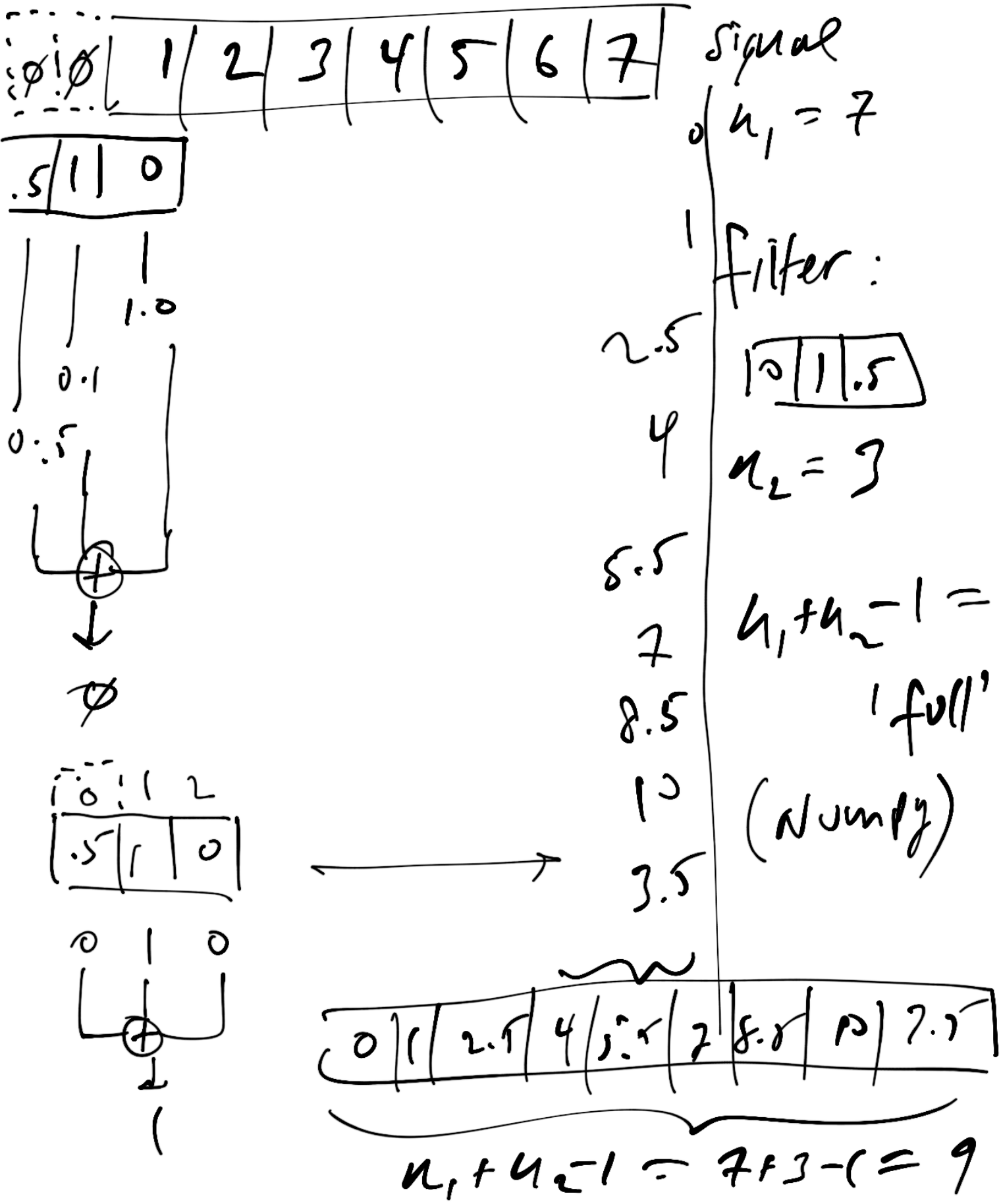
$$\theta = \operatorname{atan2}\left(r_y(\vec{x}),\ r_x(\vec{x})\right)$$

$$\underbrace{\left\| \vec{\nabla} r(\vec{x}) \right\|}_{\text{gradient magnitude}} = \sqrt{r_x()^2 + r_y()^2}$$

div

direction to sam N, E, S, W

$$\vec{u}(\theta) = \left(\cos(\theta), \sin(\theta)\right)$$


signal
$n_1 = 7$
filter:
$n_2 = 3$
$n_1 + n_2 - 1 = 9$
'full'
(numpy)
$n_1 + n_2 - 1 = 7 + 3 - 1 = 9$

$n_1 + n_2 - 1 = 9$ 'full'

'same'

$\max(n_1, n_2) = 7$ [same length as signal]

$\max(n_1, n_2) -$ 'valid'

$\min(n_1, n_2) + 1 = 5$

$g(i) = \boxed{0, 1, 5}$ $f(i) = \boxed{1\ 2\ 3\ 4\ 5\ 6\ 7}$

$$(f * g)(i) = \sum_{j=1}^{m} g(i) f(i-j)$$

$$\sum_{k=1}^{m} g(i)\,f(i-k)$$

mult

v_2 v_1

Σ : for loop

how to code? implied:

signal (m) filter length of signal

```
for(n=0; n < n1+n2-1; n++)
    for(k=0; k <= n &&
            k < max(n1,n2); k++)
        fv1[n] += (k < n1 ? v1[k] : 0) *
            (n-k < n2 ? v2[n-k] : 0);
```

$g(i)$: the filter V2 in code

$f(i)$: the signal V1 in code

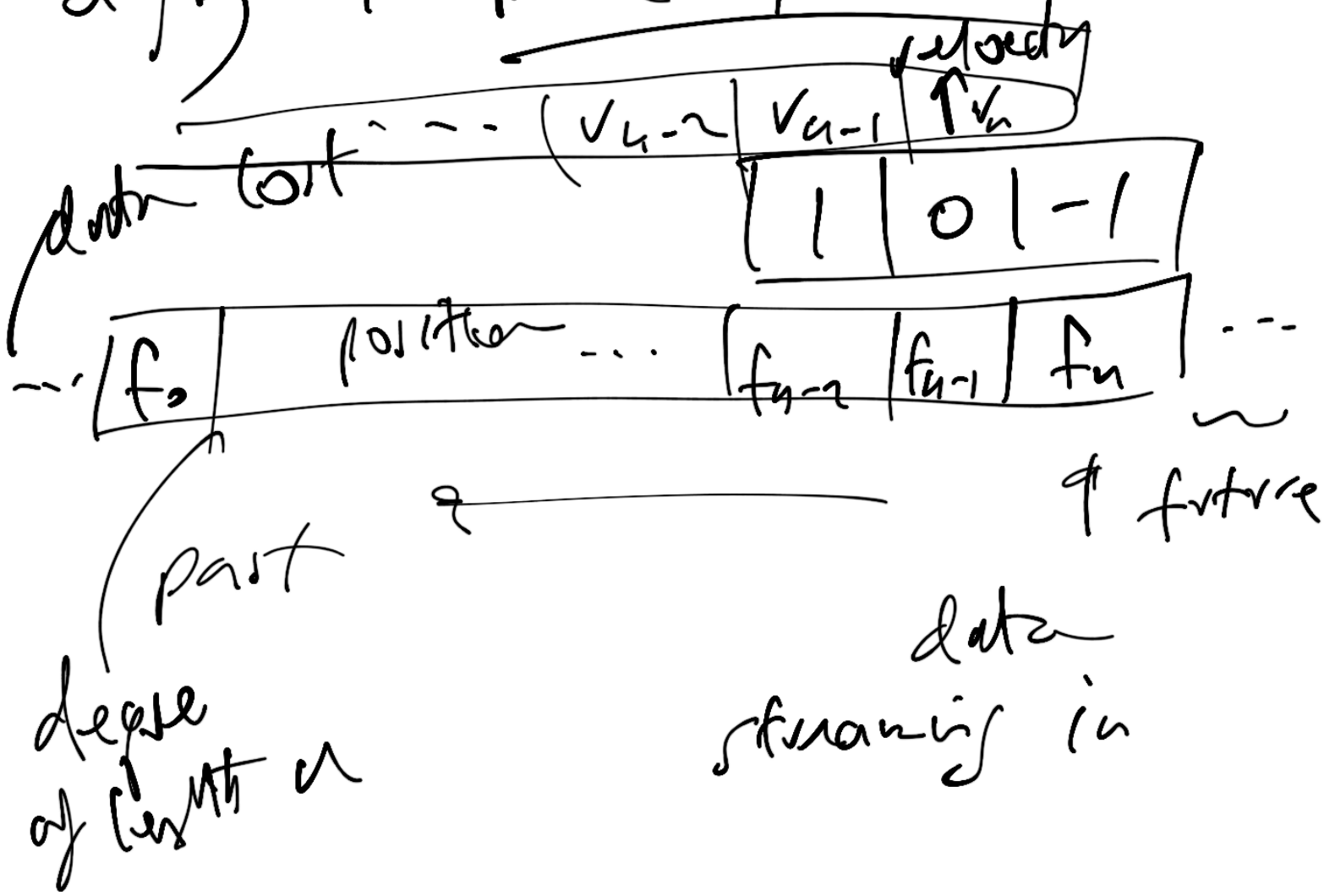
$$(f * g)(i) = \sum_{k=1}^{?} f(i)\, g(i-k)$$

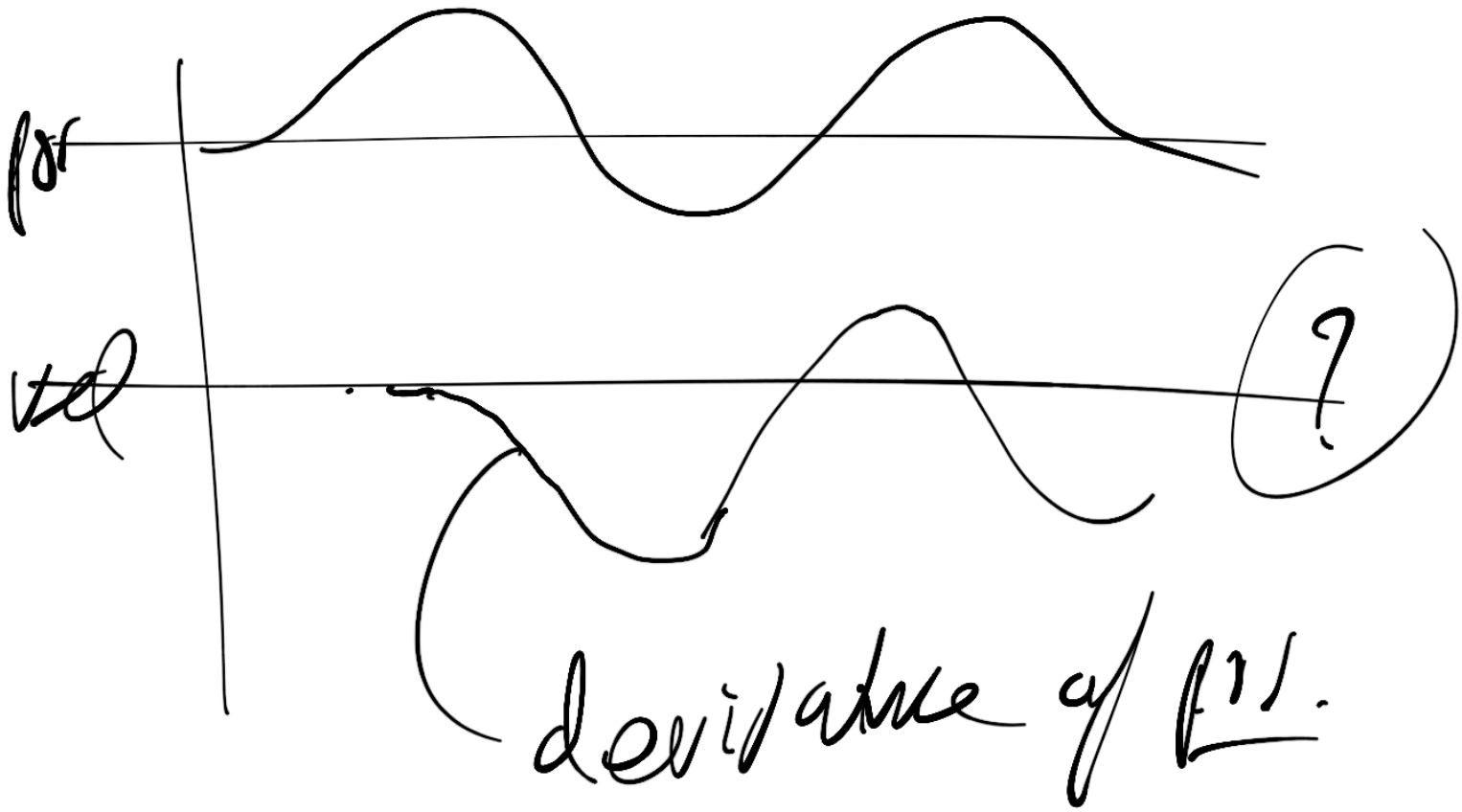
						0	1	.5	V2 | V1
7	6	5	4	3	2	1			V1 | V2

like holding filter in place, sliding signal along.

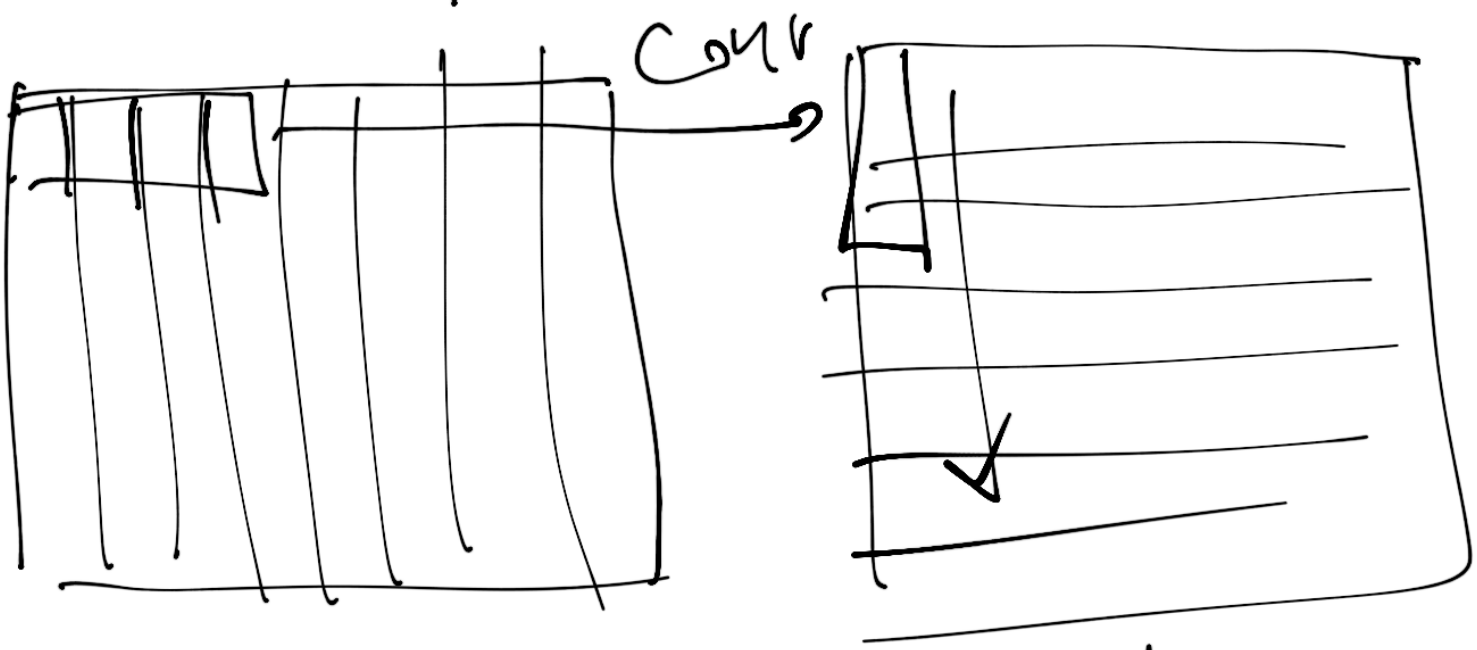
when to 'hold filter in place'?

when streaming data, i.e.) in real-time

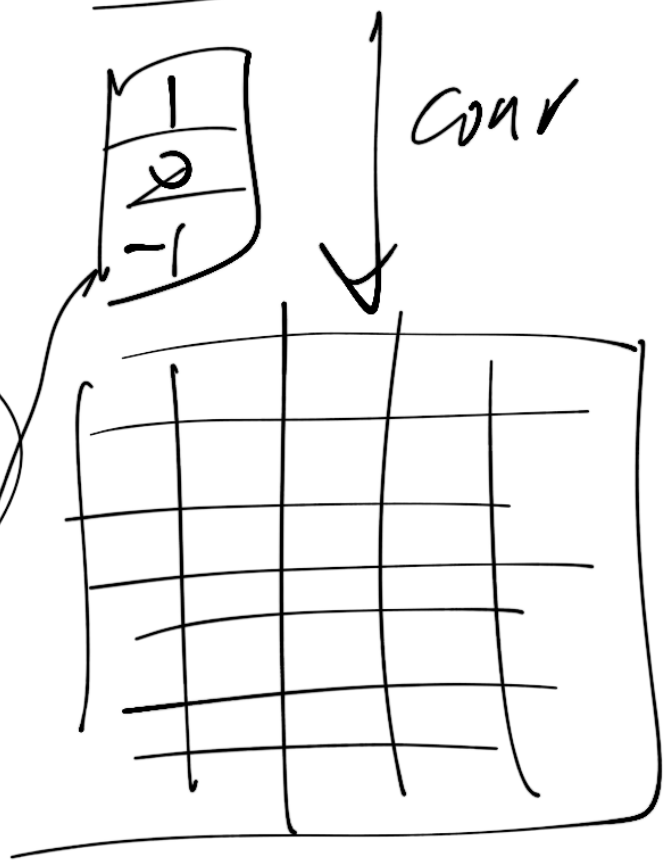



f(x)
?
derivative of f(x).

Back to arg02 : use separable filter



step 1: along rows



step 2: along col

G_x

Gy:

$$\begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline \end{array} \qquad \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array}$$

$$1 \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \quad 0 \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \quad -1 \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

$$\begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array}$$

G_x

1	2	1

1
0
-1

$$1\begin{pmatrix}1\\0\\-1\end{pmatrix} 2\begin{pmatrix}1\\0\\-1\end{pmatrix} 1\begin{pmatrix}1\\0\\-1\end{pmatrix}$$

1	2	1
0	0	0
-1	-2	-1

1	2	1
0	0	0
-1	-2	-1

$m=3$

$n=3$

```
for(i=0; i<rows; i++)
  for(j=0; j<cols; j++)
    for(k=0; k<m; k++)
      ~~for(l=0; l<n; l++)~~
```

separable filters do away with

$O(rows * cols * m * n) \rightarrow$

$O(\text{rows} * \text{cols} * m) +$

$O(\text{rows} * \text{cols} * n)$

a little bit of <u>savings</u>

In theory, separable filters are a little bit more efficient

G_x, G_y images : you get 2,

one with horiz. edges,

" " vert. edges

- to output these as 'visible' images don't forget to normalize

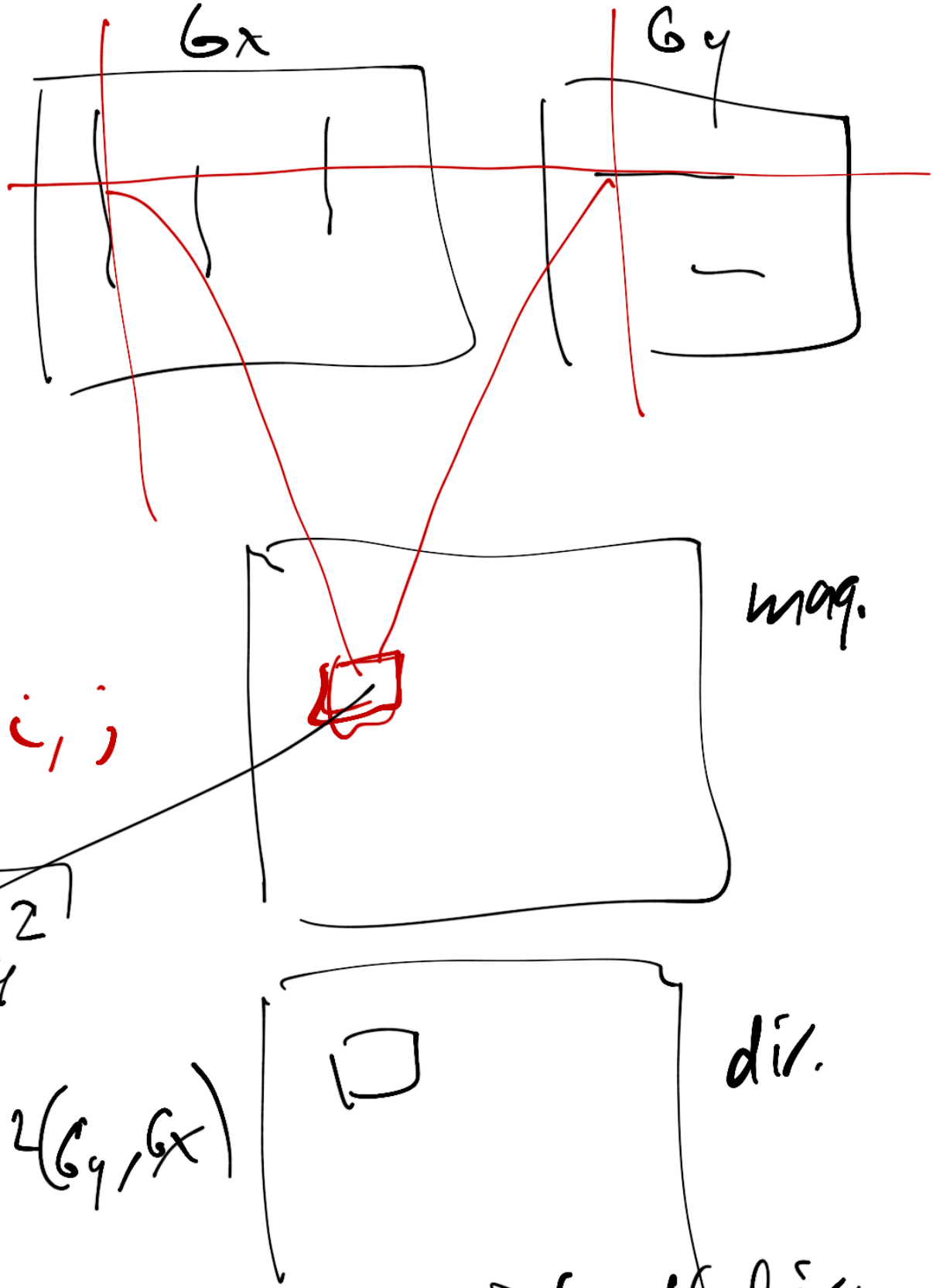
1) find max value
2) divide each pixel by max

then write out to PGM

Normalizing: semi-final step before output

to <u>use</u> edge data (e.g. getting gradient info) don't normalize

G_x, G_y images:



save i, j

$$\sqrt{G_x^2 + G_y^2}$$

$$\theta = \operatorname{atan2}(G_y, G_x)$$

discretize this — info 4 dir, N, E, W, S

- can do something like colorize image to show dir. of edges

asg01:

- couple of instances of not compiling
- couple of C++ implementations
- only 1 subclass implementation.

PNM — no such image

|

PBM

|

PGM

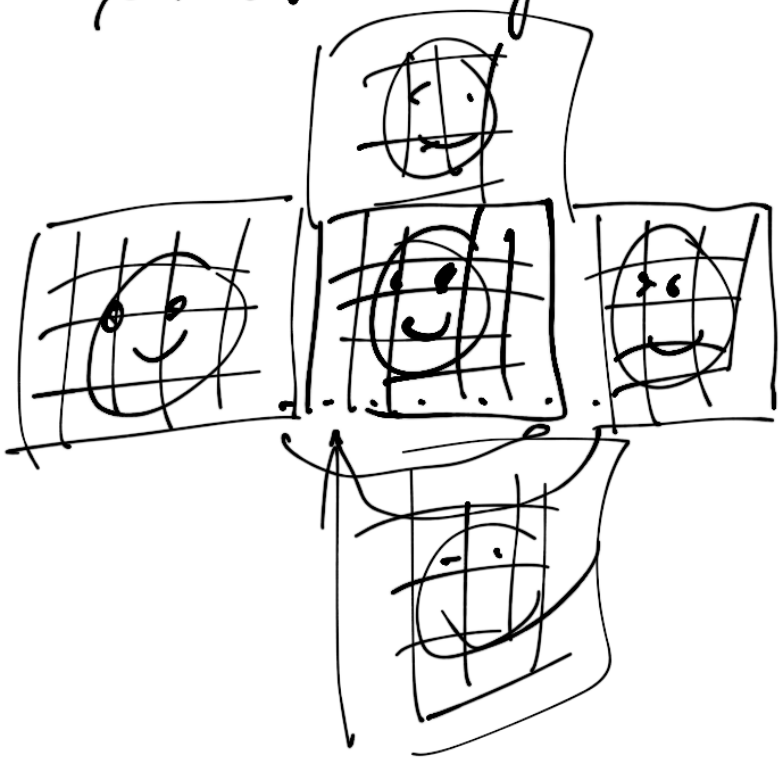
PPM

- boundary conditions for convol.:
  - use either $\emptyset$ padding
  - use periodic padding
    (wrap-around indices)
    a la Python [-1]

given r, c dimensions of image,

```
# define I(i,r) (((r)+(i))%(r))
# define J(j,c) (((c)+(j))%(c))
# define M2A(i,j,r,c) (I((i),(r))*(c) +
                        J((j),(c)))
```

Periodic extension of image



in ppm.c: //why in main.c ?

```
PGM *pgm_togrey(PPM *cing)
{
    int i,j;
    PGM *ging = NULL;
    float r,g,b, l; // l for luminance

    ging = pgm_alloc(cing->rows, cing->cols);
```

```
for(i=0; i<cing->rows; i++) {
    for(j=0; j<cing->cols; j) {
        r = cing->rpix[i*cing->cols+j];
        g = "   gpix   "
        b = "   bpix   "
        l = 0.3*r + .3*g + .3*b;
        ging->gpix[i*ging->cols+j] = l;
    }
}
return ging;
```

very cool

- Fourier Transforms etc. → DWT
  (FT) (Discrete Wavelet Trans.)

- Key aspects:
  - both FT & DWT are lossless transforms

image in frequency space

$$I(i,j) \leftrightarrow F(I(i,j))$$

image in spatial/image coordinates

"perfect reconstruction"

- PROJECTION ONTO BASIS FUNCTIONS (VECTORS)

- texts to consult:

  Szeliski (section 3.4)

  Glassner: Digital Image Synthesis

  ← comp. graphics: ray tracing

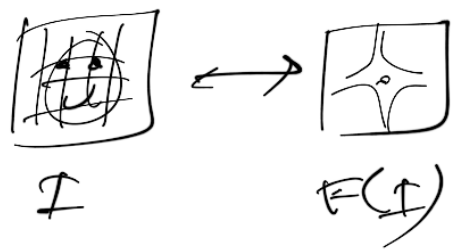
- Often intro goes like this:

given the function $\underbrace{h(k,l)}_{\text{e.g. smoothing, Gauss, filter}}$ or $h(x,y)$

its Fourier Transform is:

$$H(\omega_x, \omega_y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x,y)\, e^{-j(\omega_x x + \omega_y y)}\, dx\, dy$$

omega, or frequency



- discrete form:

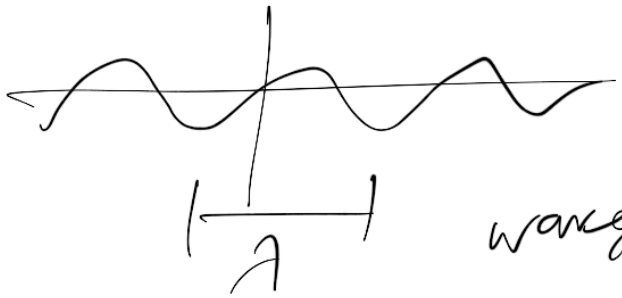
$$H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x,y)\, e^{\frac{-j2\pi k_x x + k_y y}{MN}}$$

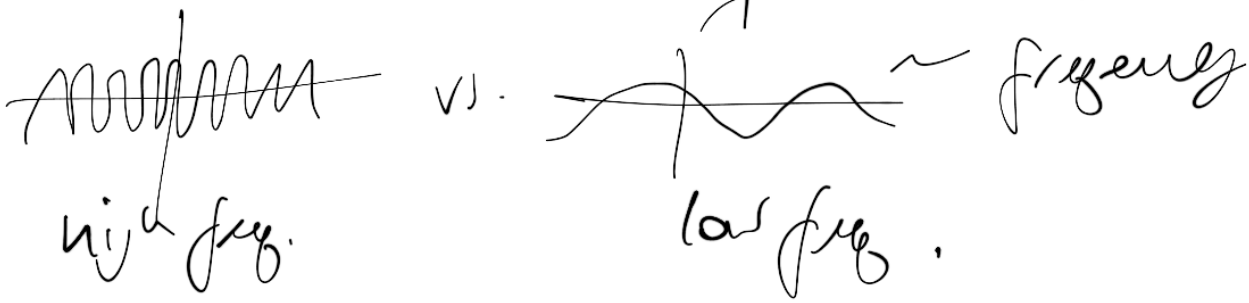
- I used to be quite confused by this
- what's the $e^{-j(w_x x + w_y y)}$ mean?
- rewrite as $e^{-i(w_x x + w_y y)}$ $(i \text{ vs. } j)$

using Euler's formula:

$$e^{ix} = \cos w_x x + i \sin w_y y$$

- cos, sin :





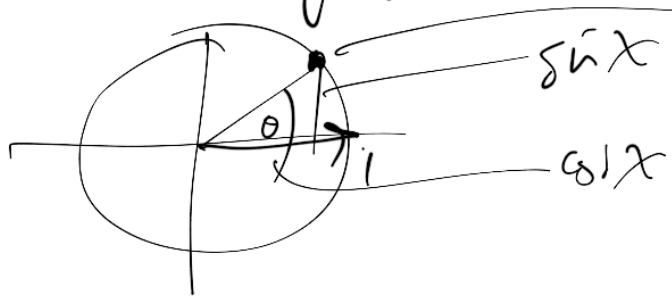
same cos function, diff parameters

↳ diff frequencies!

It's a basis function!

- FT: just project spatial info (x, y) onto cos, sin basis (freq info)

- also, don't forget $e^{ix} = \cos x + i \sin y$ reminiscent of polar coordinates



$(r\cos\theta, r\sin\theta)$

unit circle, radius 1

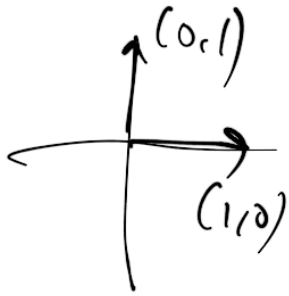
just a different coordinate system.
vs $\vec{x}$

- FT : takes a signal (e.g. 2D image)
  & represents it in frequency space

  — FT is a mathematical operator
  that <u>decomposes</u> signal to
  sum of weighted <u>sines & cosines</u>
  basis functions

- Any signal can be represented as combination of basis functions — even our Euclidean space

- <u>orthogonal</u> vectors ($\perp$ perpendicular)



$v_x = [1\ 0]$ $\boxed{\text{normalized}}$

$v_y = [0\ 1]$ (length 1)

<u>orthogonal</u> and normal : <u>orthonormal</u>

$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1 \cdot 0 + 0 \cdot 1 = 0$

<u>dot product</u>

$\|v_x\| = \sqrt{1^2 + 0^2} = 1$

length $\|v_y\| = \sqrt{0^2 + 1^2} = 1$

- test for orthogonality (are they $\perp$)?

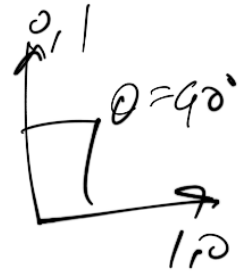
dot product : <u>sum</u> of element <u>product</u>

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \boxed{x_1 y_1 + x_2 y_2 + \dots + x_n y_n} = \sum_{i=1}^{n} x_i y_i$$

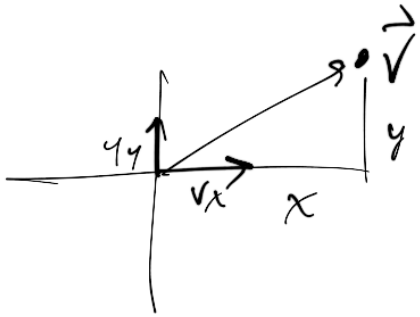
$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1 \cdot 0 + 0 \cdot 1 = 0 + 0 = 0$$

if $\cos(\theta) = 0$
$\theta = 90°$, $\perp$

$$\boxed{\cos(\theta)} = \frac{v_x \cdot v_y}{\|v_x\| \, \|v_y\|} \Rightarrow \theta = \cos^{-1}\left(\frac{v_x \cdot v_y}{\|v_x\| \, \|v_y\|}\right)$$



- any vector $\vec{v} = (x, y)$ can be represented as a combination of <u>orthogonal</u> basis vectors

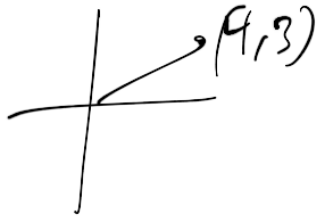


dot product which is a <u>scalar</u>

(just a number, not a vector)

$$\vec{v} = (x, y) = (v \cdot v_x) v_x + \underbrace{(v \cdot v_y) v_y}_{\text{scaling } v_y}$$

example:



$$\left(\begin{bmatrix}4\\3\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\right)\underbrace{\begin{bmatrix}1\\0\end{bmatrix}}_{V_x}+\left(\begin{bmatrix}4\\3\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\right)\underbrace{\begin{bmatrix}0\\1\end{bmatrix}}_{V_y}$$

$$(4)(1)+(3)(0)\begin{bmatrix}1\\0\end{bmatrix}+(4)(0)+(3)(1)\begin{bmatrix}0\\1\end{bmatrix}$$

$$4\begin{bmatrix}1\\0\end{bmatrix}+3\begin{bmatrix}0\\1\end{bmatrix}=\begin{bmatrix}4\\0\end{bmatrix}+\begin{bmatrix}0\\3\end{bmatrix}=\begin{bmatrix}4\\3\end{bmatrix}$$

QED

dot product: $\vec{v} \cdot \vec{w} = \sum_{i=1}^{n} v_i w_i = \cos(\theta)$

$$\theta = \cos^{-1}\left(\frac{\vec{v} \cdot \vec{w}}{\|v\| \, \|w\|}\right)$$

ex.

$$\underset{1\times 3}{\begin{bmatrix} 1 & 3 & -5 \end{bmatrix}} \underset{3\times 1}{\begin{bmatrix} 4 \\ -2 \\ -1 \end{bmatrix}} = (1)(4) + (3)(-2) + (-5)(-1)$$

$$= 4 + -6 + 5$$

$$\underset{1\times 1}{} = 3$$

- Key observation is that basis vectors were <u>scaled</u>

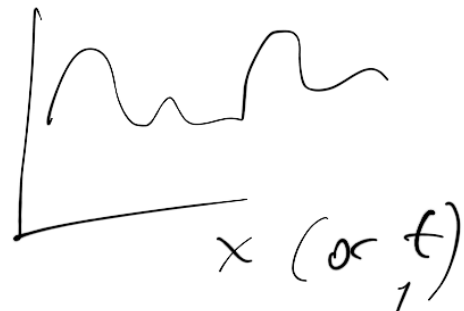
$$V = \underbrace{\underbrace{(v \cdot v_x)}_{\text{scalar}} v_x + \underbrace{(v \cdot v_y)}_{\text{scalar}} v_y}$$

projection of vector onto space defined by basis vectors (or functions)

- FT discussion usually starts with 1D function $y = f(x)$

- $y = f(x)$

spatial domain



temporal domain

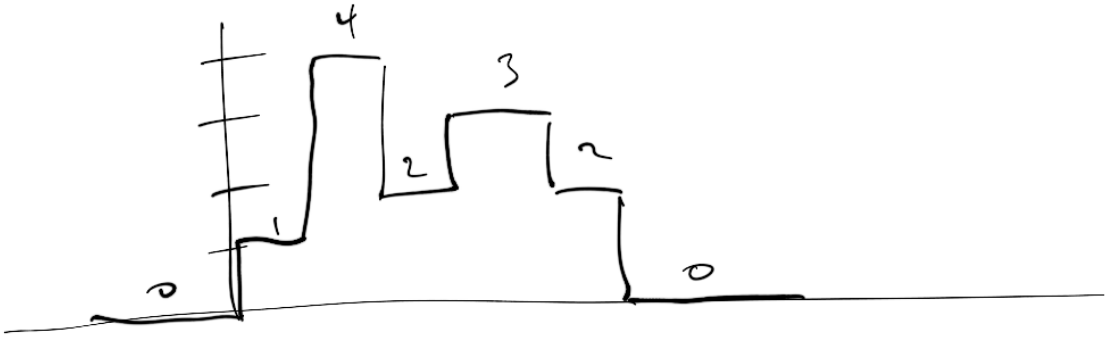
$f(t)$ can be projected onto basis functions $\phi_i(t)$:

weight or coefficients

$$f(t) = c_1 \phi_1(t) + c_2 \phi(t) + \cdots + c_n \phi(t)$$

linear combination of <u>weighted</u> basis functions

- example: 'bar chart' function:



$$f(t) = (1, 4, 2, 3, 2)$$

$$t = 1 \quad 2 \quad 3 \quad 4 \quad 5$$

$$f(t) \in \begin{cases} [1,5] \\ 0 \end{cases} \Rightarrow f(t) = \begin{cases} f(t), & 1 \le t \le 5 \\ 0, & \text{otherwise} \end{cases}$$

asg02 recap

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

3x3

m2d

i+k, j+l

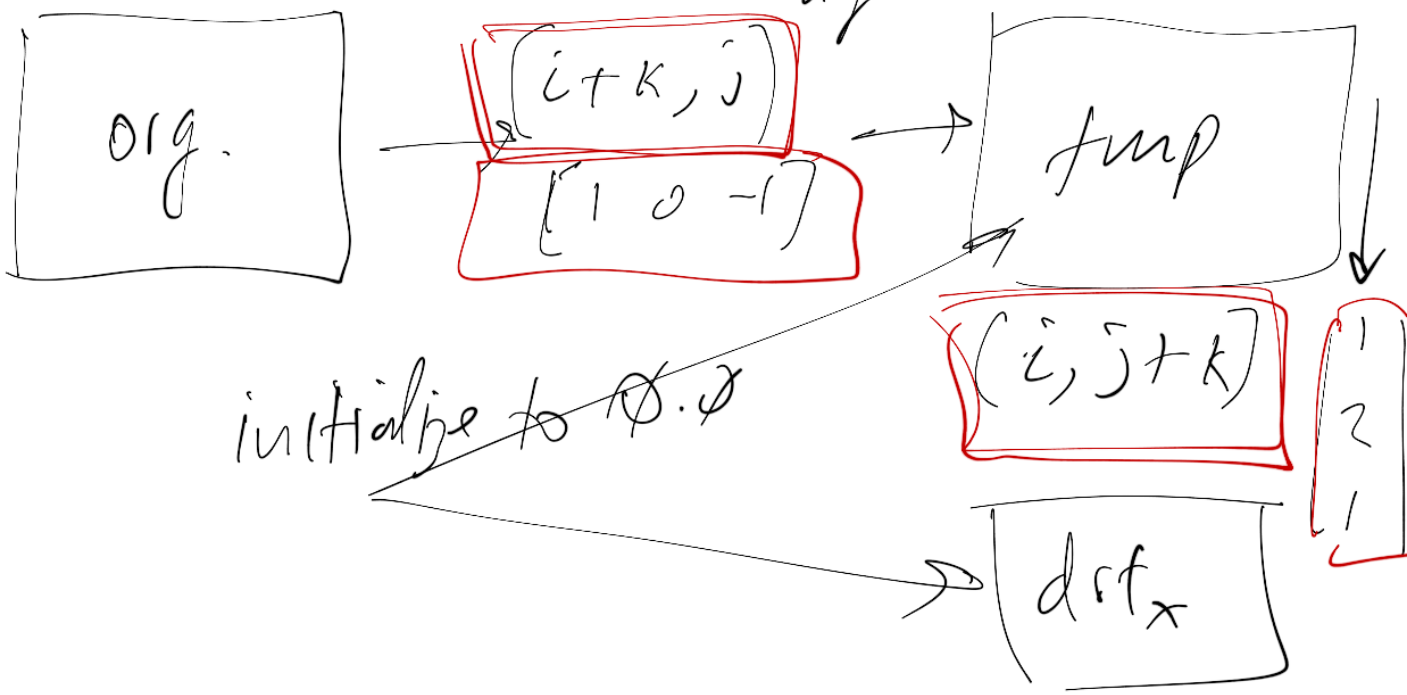
img[i*cols+j]+=

m2d[k*3+l]

```
for(i=0; i< rows; i++)
    for(j=0; j< cols; j++)
        f(k=0; k<3; k++)
            for(l=0; l<3; l++)
                img[i*cols+j]+=
```

using 2D kernel coefficient

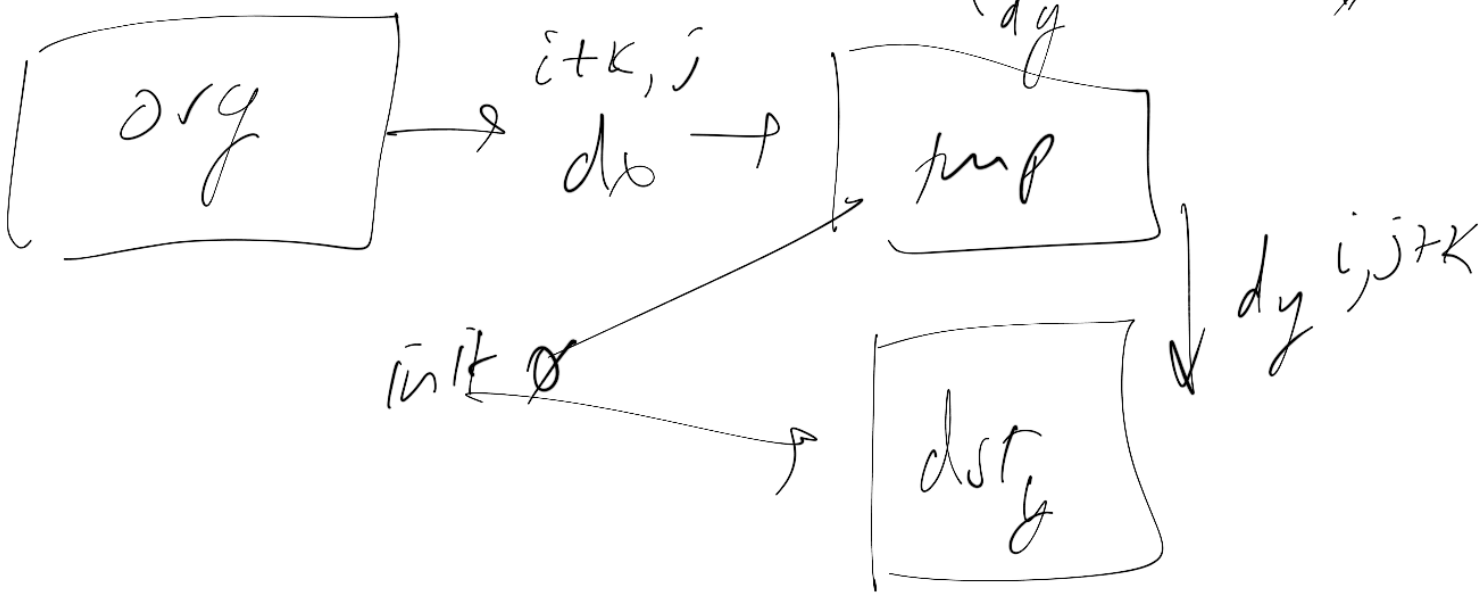
$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}_{dy} \underbrace{\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}}_{dx}$$



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

d_x

d_y



result: org image (unfolded)

dst_x image } these are
dst_y image } edge-detected images,

UNNORMALIZED

→ at this point, you can use dst_x, dst_y for gradient image info if you wish

→ BUT, for output, must normalize dst_x dst_y

dst_x → pass 1. find min val
2. find max val → $udst_x$

uchar [0, 255]

$* 255$

$\in [0, 1]$

$\hat{I}(i,j)$

pass 2:

$$\frac{I(i,j) - \min}{\max - \min} =$$

$$\frac{I(i,j) - a}{b - a} \cancel{*1} \cancel{-0}$$

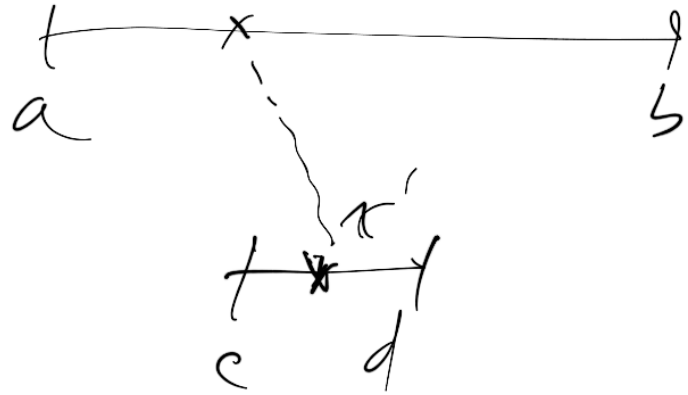
not 255

would be −ve

a : min val of image
b : max val of u

} 2D loop to find

normalization — remapping — linear remapping



from range [a, b] to range [c, d]

min max

$$x' = \frac{(x-a)}{(b-a)} * (d-c) + c$$

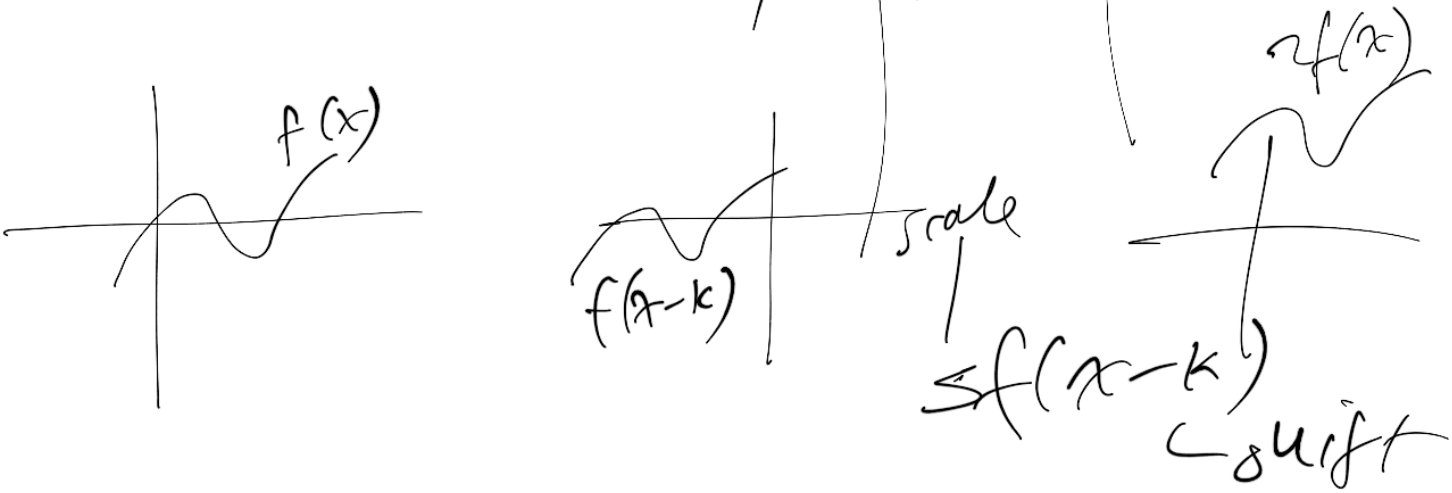
normalization

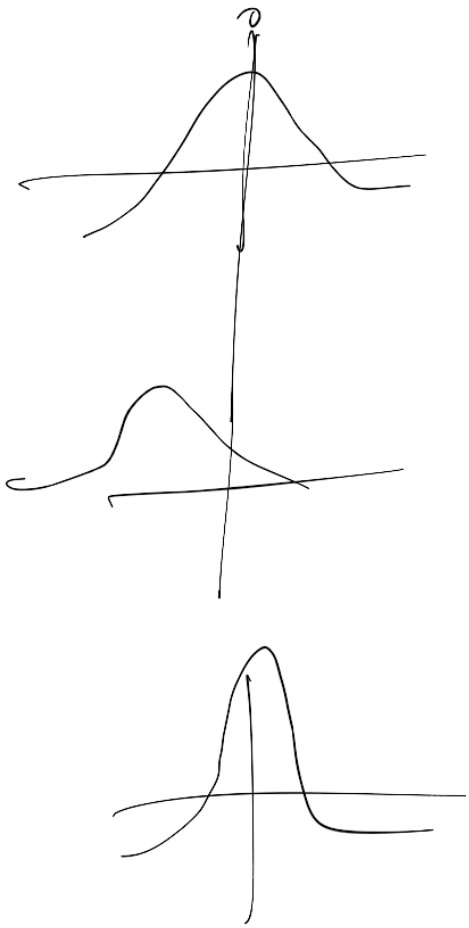
scaling

shifting

$x-k$

x	$x-k$	k
1	-2	3





$$e^{-\frac{x}{\sigma^2}}$$

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

$$e^{-\frac{x}{2\sigma^2}}$$

1.j. $\left(\frac{195}{255} \right.$ — $195 - 0$

is this normalization? yes

$255 - 0$ $\quad \times 1 \Big) - 0$

max min

Back to basis functions

$f(t)$ $(1, 4, 2, 3, 2)$

$f(5)$

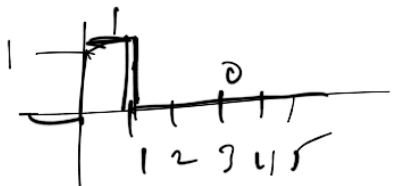
t 1 2 3 4 5

$$f(t) \in [1,5] \in \begin{cases} f(t), & 1 \le t \le 5 \\ \emptyset, & \text{elsewhere} \end{cases}$$

- basis functions $\phi_1(t)$ to $\phi_5(t)$

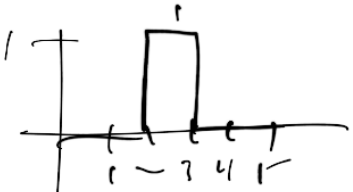
$$\phi_i(t) = \begin{cases} 0, & t < i-1 \\ 1, & i-1 \leq t < i \\ 0, & t > i \end{cases}$$

$i \in \{1, 2, 3, 4, 5\}$

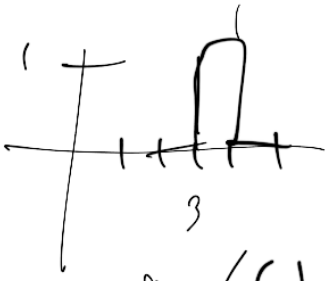


$\phi_1(t)$

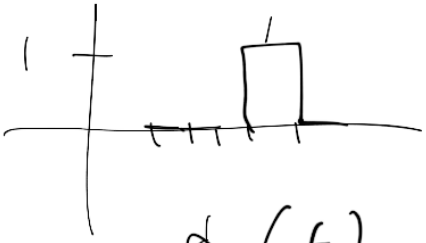
$\phi_2(t)$



$\phi_3(t)$



$\phi_4(t)$



$\phi_5(t)$

- bar chart $p(t)$

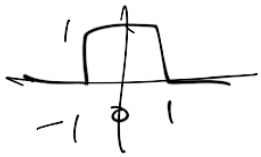
$$p(t) = p_1 \phi_1(t) + p_2 \phi_2(t) + \cdots + p_5 \phi_5(t)$$

$$= \sum_{i=1}^{5} p_i \phi_i(t)$$

coefficients

basis functions

- normally, we'd just have 1 basis function & shift it



$\phi(t)$ ⇒ then shift to where you need to sample.

$\phi(t-k)$

— $\phi(t-k)$ needs to be mutually orthogonal to span space

In general,

$$\int_{t_1}^{t_2} \phi_i(t)\,\phi_j(t)\,dt = \begin{cases} 0, & i \neq j \\ \neq 0, & i = j \end{cases}$$

orthogonal

usually 1 if orthonormal

$$= \langle \phi_i, \phi_j \rangle$$

generalized dot product

- for vectors $v_x, v_y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, show they are orthonormal

$$\langle v_x, v_y \rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad i \neq j$$

(a)

$$= 0(1) + 1(0) = 0$$

(no overlap $\Rightarrow$ orthogonal)

(b) when $i = j$

$$\boxed{\langle v_x, v_x \rangle} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1(1) + 0(0) = 1$$

normal $\Rightarrow$ orthonormal

$\boxed{\text{same for } \langle v_y, v_y \rangle}$ (c)

- for ϕ, system is normalized if

$$\langle \phi_i, \phi_i \rangle = 1$$

$$\langle \phi_i, \phi_j \rangle = 0 \quad \text{orthogonal}$$

- one can normalize by dividing by the norm or length

$$\langle \phi_i, \phi_i \rangle = \| \phi_i \|$$

asg02:

- generally well done
- I appreciate included images
- C++ OK, BUT... try to update Makefile
- one person used 2D kernel
- do use the object-oriented C approach (avoid the huge main() function)

- let's say $f(t) \approx \sum_{i=1}^{n} c_i \phi_i(t)$

is an approximation

(mapping onto basis fctns, coefficients (weights))

- how to minimize the error?
- define Mean-Squared Error (MSE):

$$MSE = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \left[f(t) - \sum_{i=1}^{n} c_i \phi_i(t) \right]^2 dt$$

over interval $[t_1, t_2]$

- rewrite in long form:

$$\frac{1}{t_2-t_1}\int_{t_1}^{t_2}\left[f(t)-c_1\phi_1(t)-c_2\phi_2(t)-\cdots-c_n\phi_n(t)\right]^2 dt$$

- take the square:

$$\frac{1}{t_2-t_1}\int_{t_1}^{t_2}\Big[f^2(t)+c_1^2\phi_1^2(t)+\cdots+c_n^2\phi_n^2(t)-2c_1f(t)\phi_1(t)-2c_2f(t)\phi_2(t)-\cdots-2c_nf(t)\phi_n(t)\Big]dt$$

$$[a-b][a-b]=a^2-2ab+b^2=a^2+b^2-2ab$$

$$= \frac{1}{t_2 - t_1} \left\{ \left(\int_{t_1}^{t_2} f^2(t)\, dt \right) + c_1^2 k_1 + c_2^2 k_2 + \dots + c_n^2 k_n - 2c_1\gamma_1 - 2c_2\gamma_2 - \dots - 2c_n\gamma_n \right\}$$

where $k_i = \langle \phi_i, \phi_i \rangle$

$\gamma_i = \langle f, \phi_i \rangle$

– the $c_1^2 k_1 + c_2^2 k_2 + \cdots + c_n^2 k_n - 2c_1\gamma_1 - 2c_2\gamma_2 \cdots 2c_n\gamma_n$

can be written as

$$\left(c_i^2 k_i - 2c_i\gamma_i\right) = \boxed{\left(c_i\sqrt{k_i} - \frac{\gamma_i}{\sqrt{k_i}}\right)^2 - \frac{\gamma_i^2}{k_i}}$$

to complete the square

$$\left(c_i\sqrt{k_i} - \frac{\gamma_i}{\sqrt{k_i}}\right)\left(c_i\sqrt{k_i} - \frac{\gamma_i}{\sqrt{k_i}}\right) - \frac{\gamma_i^2}{\sqrt{k_i}}$$

$$= c_i^2 k_i - 2c_i\gamma_i + \cancel{\frac{\gamma_i^2}{k_i}} - \cancel{\frac{\gamma_i^2}{k_i}}$$

- plug back in:

$$MSE = \frac{1}{t_2 - t_1}\left\{ \int_{t_1}^{t_2} f^2(t)\,dt + \sum \left(c_i\sqrt{k_i} - \frac{\gamma_i}{\sqrt{k_i}} \right) - \sum \frac{\gamma_i^2}{k_i} \right.$$

to minimize, set $\boxed{c_i\sqrt{k_i} = \frac{\gamma_i}{\sqrt{k_i}}}$

so that for each i, the squared error goes to $\emptyset$

- divide both sides by $\sqrt{k_i}$ & solve for c_i

$$c_i = \frac{\gamma_i}{k_i} = \frac{\int_{t_1}^{t_2} f(t)\,\phi_i(t)\,dt}{\int_{t_1}^{t_2} \phi_i^2(t)\,dt} = \frac{\langle f, \phi_i \rangle}{\langle \phi_i, \phi_i \rangle}$$

- in the end, we basically convolve $f(t)$ with basis functions $\phi_i(t)$
- basically like $(v \cdot v_x) v_x + (v \cdot v_y) v_y$

- In other words, we just project $f(t)$ onto space defined by $\phi_i(t)$
- basically a change of coordinate space from

pixels (space, time) $\boxed{\text{spatial domain}}$ onto frequencies $\boxed{\text{frequency space}}$

- The FT uses $\varphi_n(t) = e^{inwt}$

$n \in \mathbb{Z}$

$\mathbb{Z}$ set of integers

(as opposed to $\mathbb{R}$ set of reals)

- Wavelet transform projects onto ϕ and ψ

scaling function

wavelet function

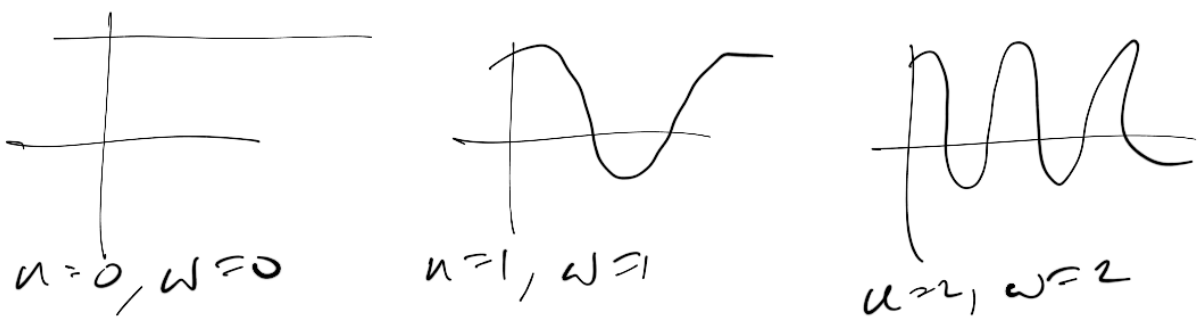
– Fourier series (discrete: $\sum$ not $\int$)

$$x(t) = \sum_k a_k e^{ik\omega t} = \langle a_k, \psi_k \rangle$$

signal like $f(t)$

$$a_k = \langle x, \psi_k \rangle$$

on the interval $T = \frac{2\pi}{\omega}$ ← omega, frequency



- Fourier Transform

$$X(\omega) = \frac{1}{\sqrt{2\pi}} \int x(t) e^{-i\omega t} dt = \frac{1}{\sqrt{2\pi}} \langle x, \psi \rangle$$

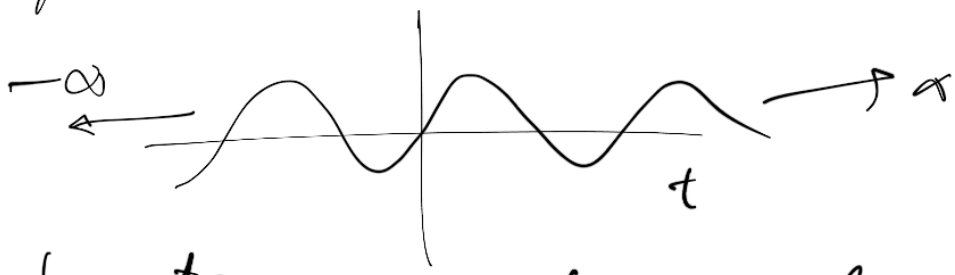
$$x(t) = \frac{1}{\sqrt{2\pi}} \int X(\omega) e^{i\omega t} d\omega = \frac{1}{\sqrt{2\pi}} \langle \bar{x}, \psi \rangle$$

$$X(\omega) = F(x(t))$$

$$x(t) = F^{-1}(X(\omega)) = F^{-1}(F(x(t)))$$

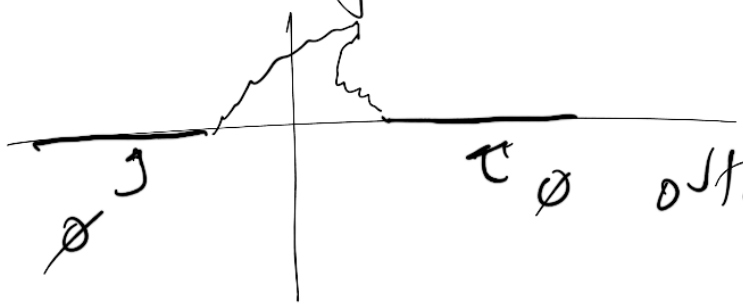
$$x(t) \xleftrightarrow{F} X(\omega)$$

- Key thing: Fourier basis functions have infinite support — they (basically) exist everywhere in the signal domain



- FT integrates over <u>entire</u> signal
- if there's a high freq. <u>burst</u> somewhere, FT will say it's there but not where it is

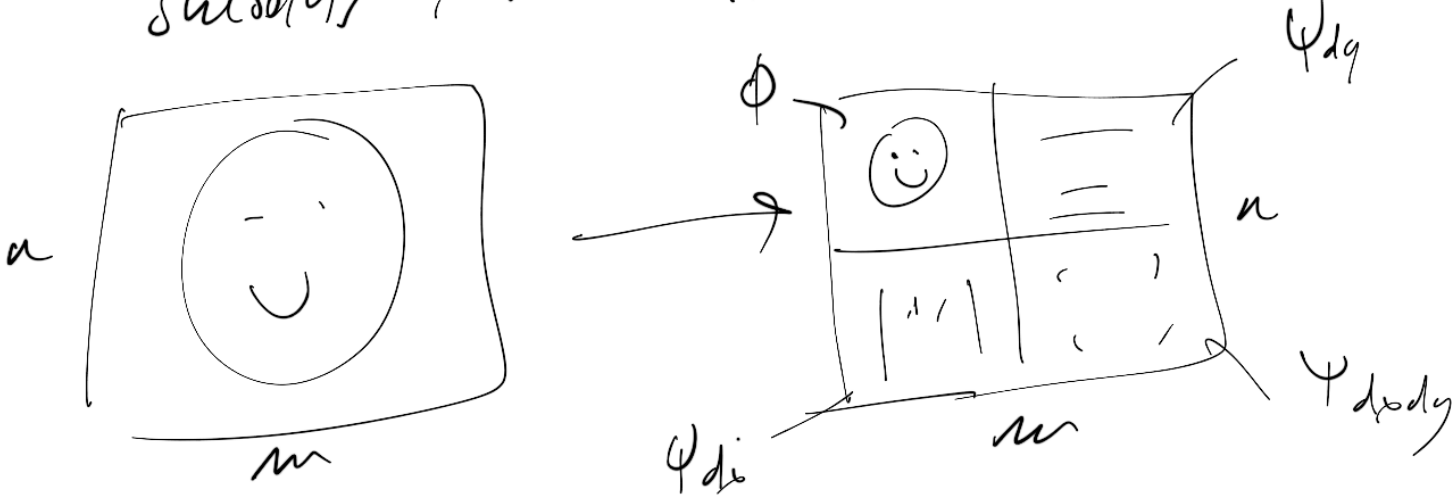
- wavelet basis functions: compact support



outside local support

Daubechies

- Wavelets :
- 2D family of functions derived from scaling function ϕ — the one that smooths & reduces signal in resolution



(scaling fn) ϕ : low-pass filter

(detail coeff.) ψ : high-pass filter
wavelet

- 2D implementation (using 1D wavelet filters, e.g., from wavelet browser page) $\phi : h$

e.g., $h : \{1, 1\}$, $g : \{-1, 1\}$ $\quad \psi : g$

unnormalized Haar filters

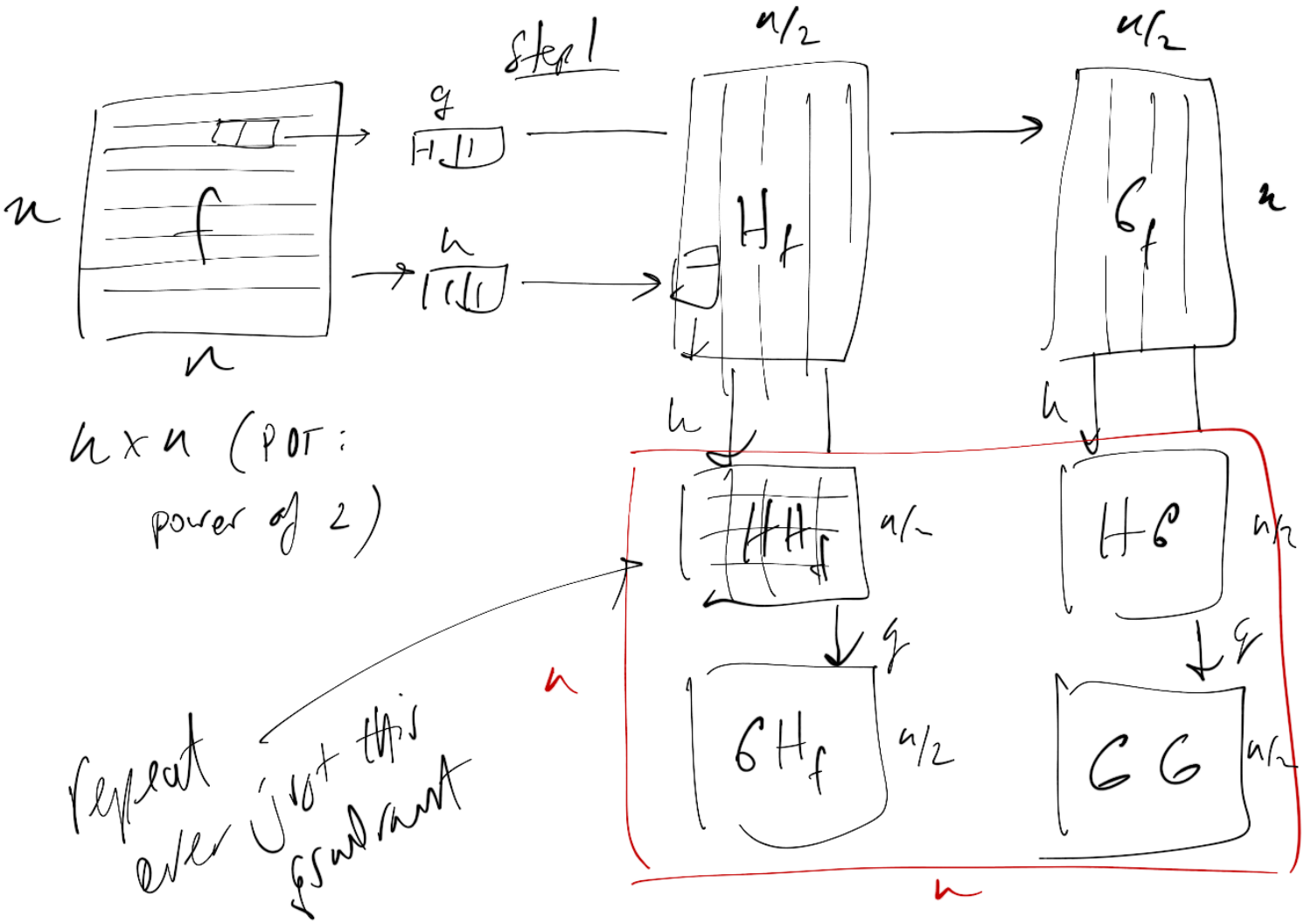
normalized Haar filters:

$$\boxed{\frac{1}{\sqrt{2}} \;\Big|\; \frac{1}{\sqrt{2}}} = h : \left\{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right\}, \; g = \left\{\boxed{-}\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right\}$$

important

$$\| $$

$$.707\ldots, .707\ldots$$



$n \times n$ (POT: power of 2)

repeat over just this quadrant

- what is the effect of 1D seq. filter:

$$HH: \phi \otimes \phi^T : \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

tensor product

$$= \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$ 2x2 filter.

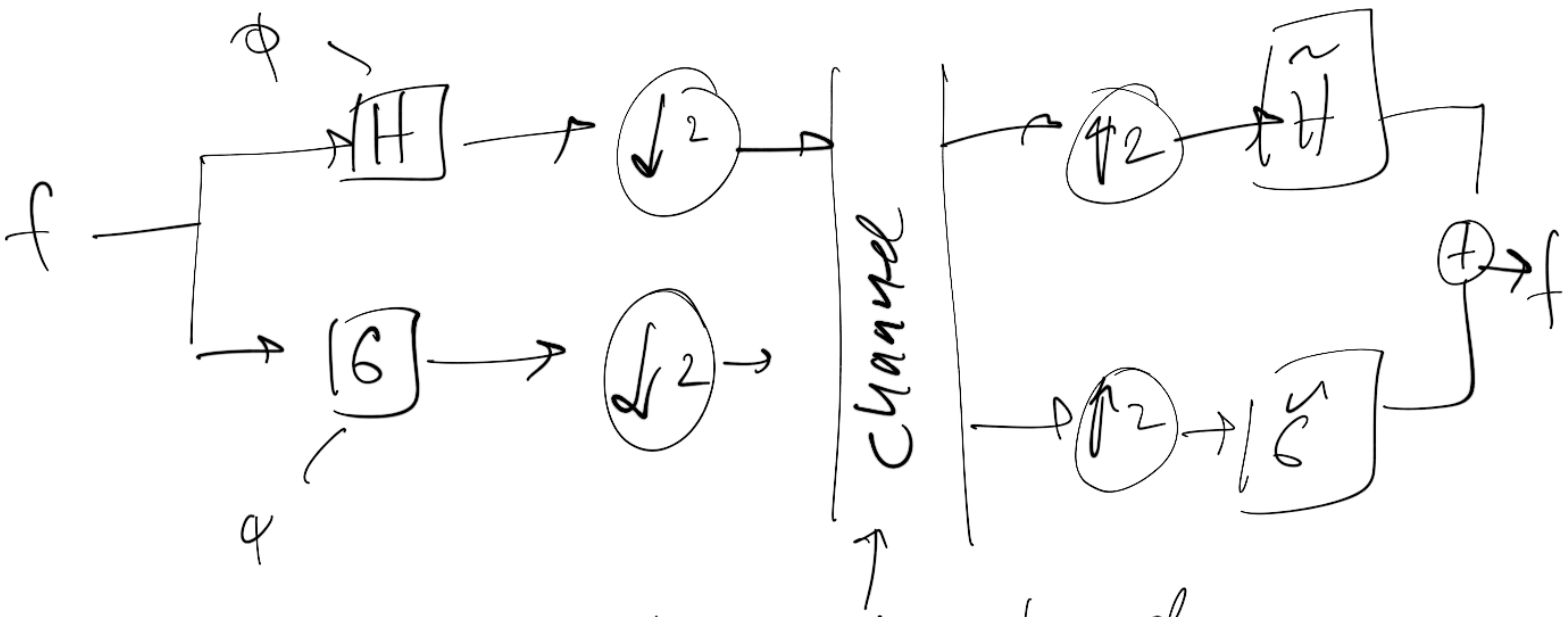
$$HG: \phi \otimes \varphi^T = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \otimes \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\frac{1}{\sqrt{2}} \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \quad \frac{1}{\sqrt{2}} \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

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

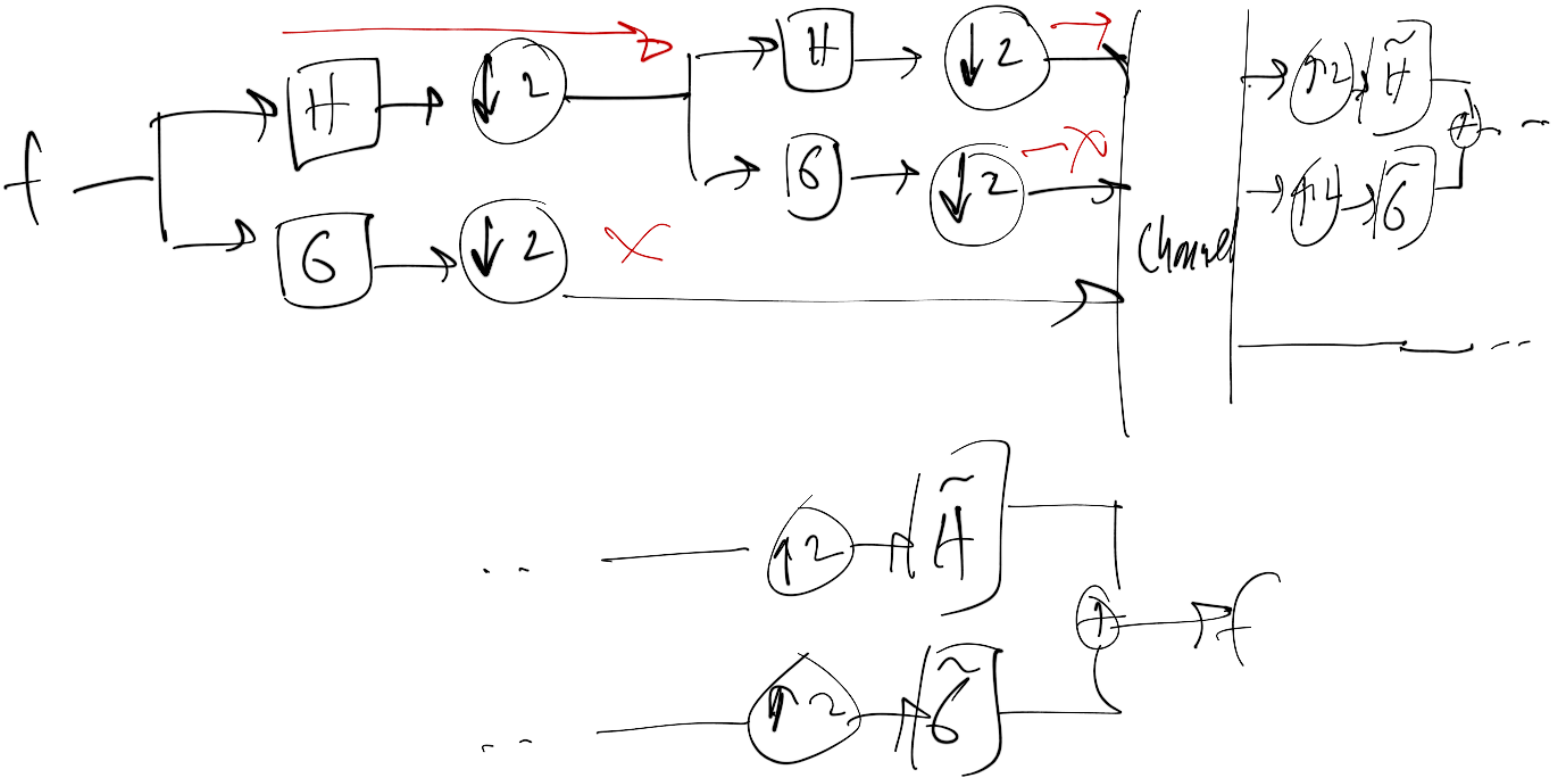
$GH: \psi \otimes \phi^T$ $\qquad$ $GG: \varphi \otimes \varphi^T$

- another (schematic) way of looking at it: (in 1D)



e.g. transmission channel

1-level



2-level

- asg 03 extras:

a) MSE error orig. img vs. idwt (any)

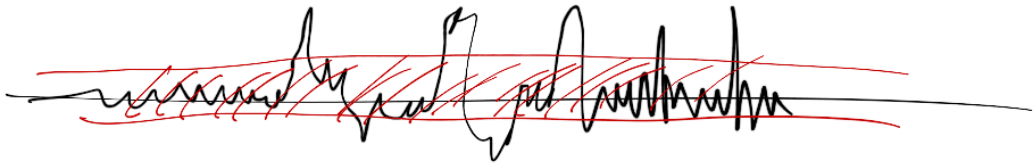
b) threshold wavelet coefficient before reconstruction

decimate coefficients — set to 0 — if below some threshold

(not the smoothed pixels)

- derivation:

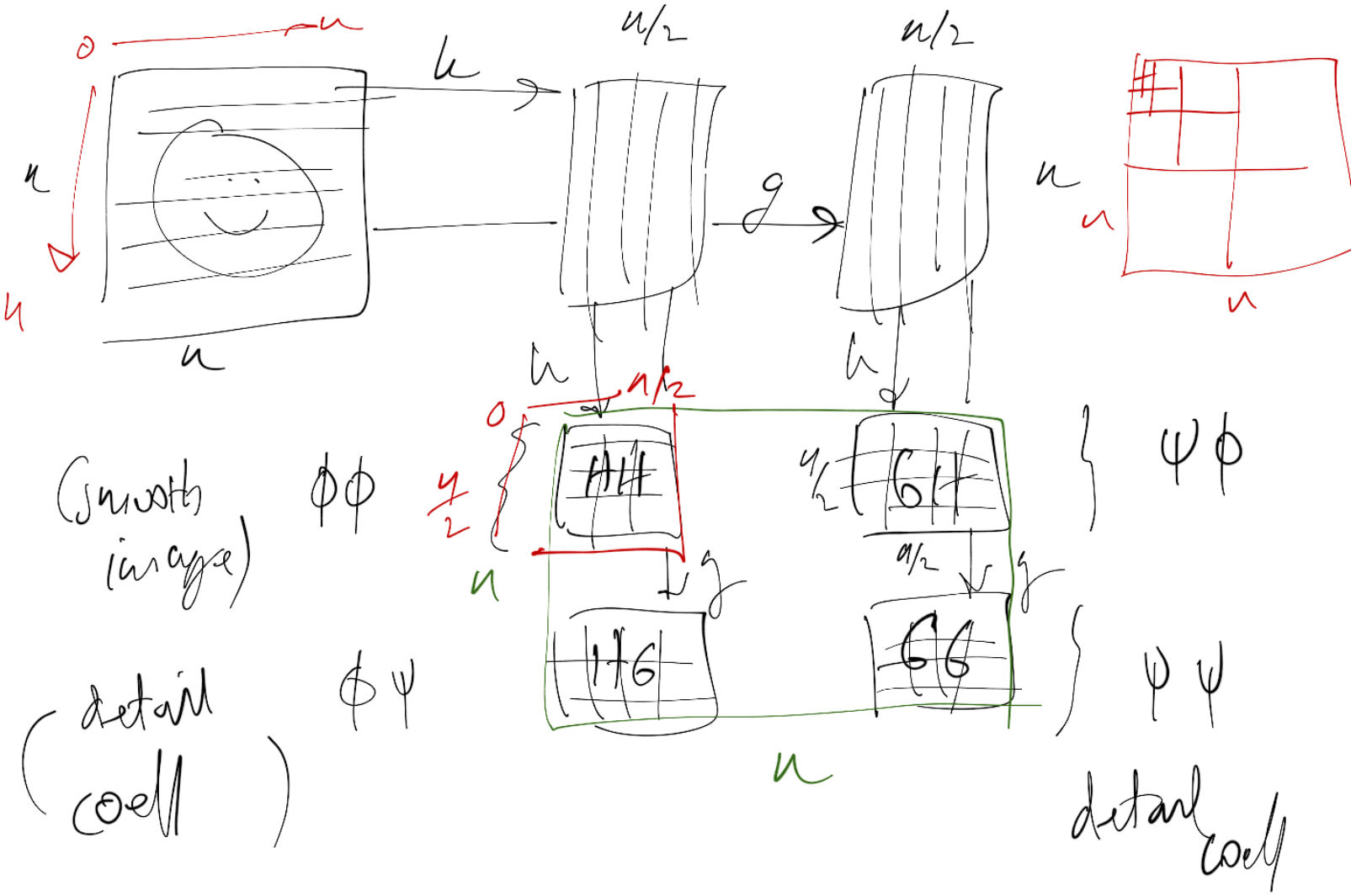
wavelet coeffs



simulating image compression, e.g.

JPEG 2000 uses wavelets

- then do MSE


0
n
n
n
h
n/2
g
n/2
n
n
n
h
g
n/2
0
n/2
n
HH
GH
HG
GG
n
n/2
n/2
n
ψφ
ψψ
(smooth image) φφ
(detail coeff) φψ
detail coeff

3	1	0	-1
2	1	9	6
4	1	7	2
Ø	8	5	4

3	4	1	-1
2	3	10	15
4	5	8	9
Ø	8	13	9

3	4	1	-1
-1	-1	9	16
2	2	-2	-6
-4	3	5	Ø

Ø	3	1	0	-1

0 + 3

3 + 1

- multiscale concept — recursion
- a mother wavelet is derived which is then

function $\psi(t)$  $\boxed{[-1,1]}$

	scaled, (multiplied)	dilated, (power)	shifted (translate)
j	$2\psi(t)$	$\psi(2^j t)$	$\psi(t-k)$

0001 1
<<1 0010 2
<<1 0100 4

in code, set $j=1$ then $\boxed{j = j << 1}$

$$2^j \psi(2^j t - k), \quad j \in \mathbb{Z}$$

$0, 1, \ldots, n.$

- wavelets: start with scaling function

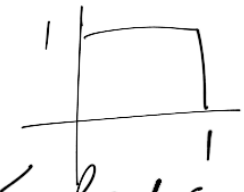
$$\phi(x) = \sum_{k=0}^{N} c_k \phi(2x-k)$$

coefficients dilation shift

- dilation is applied recursively:

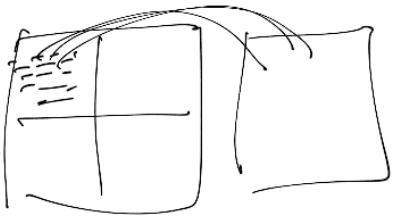
$$\phi_j(x) = \sum_k c_k \phi_{j-1}(2x-k)$$

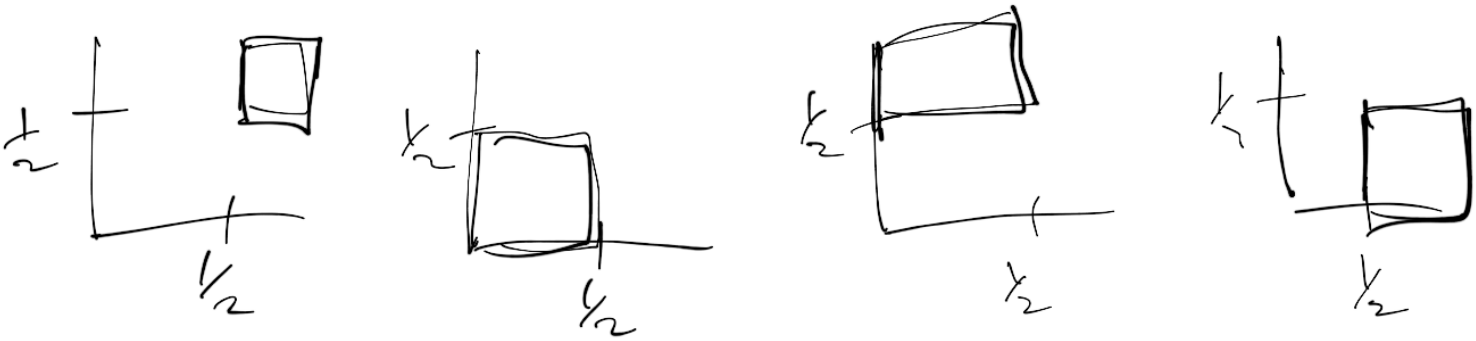
- example: unit square



$$s(x, y, lx, ly, s) = \begin{cases} 1, & lx \le x < lx+s \text{ and } ly \le y < ly+s \\ 0, & \text{otherwise} \end{cases}$$

$$s(x, y, lx, ly, s) = s(x, y, x, y, \tfrac{s}{2}) + s(x, y, x+\tfrac{s}{2}, y, \tfrac{s}{2}) + s(x, y, x, y+\tfrac{s}{2}, \tfrac{s}{2}) + s(x, y, x+s/2, y+\tfrac{s}{2}, \tfrac{s}{2})$$

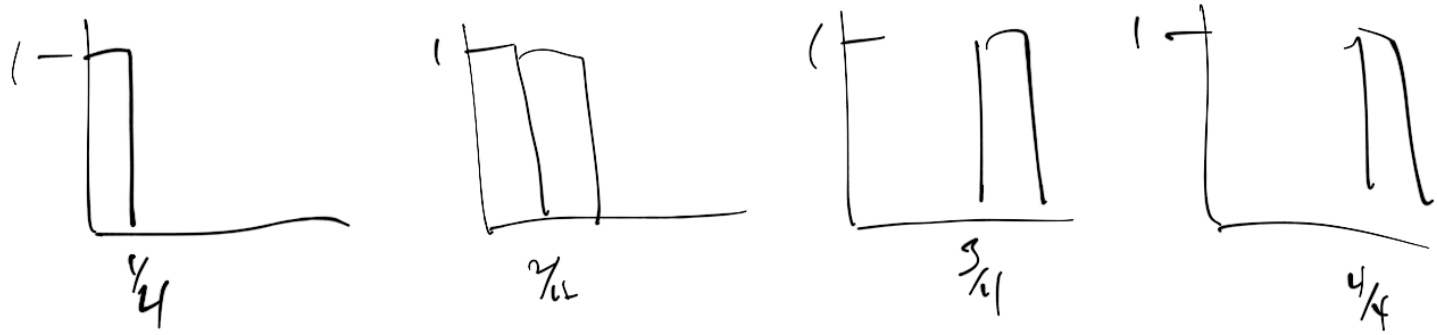




box function: $g_0(t) = \begin{cases} 1, & 0 \le t \le 1 \\ 0, & \text{otherwise} \end{cases}$

$$y_1(t) = y_0(2t) + y_0(2t-1)$$

$$y_2(t) = y_0(2t) + y_0(2t-1) + y_0(2t-2) + y_0(2t-3)$$



scaling ftn $\quad \phi_j(x) = \sum_k c_k \phi_{j-1}(2x-k)$

"recipe" for wavelet function is:

$$\psi = \sum_k (-1)^k c_{k+1} \phi(2x-k)$$

(some texts use c_{1-k}

$\boxed{\text{Quadrature mirror}}$ of ϕ

ϕ :

0	1	2	3
1	2	3	4

ψ : a) reverse ϕ (1-k)

b) negate every odd value

a)

0	1	2	3
4	3	2	1

b

4	-3	2	-1

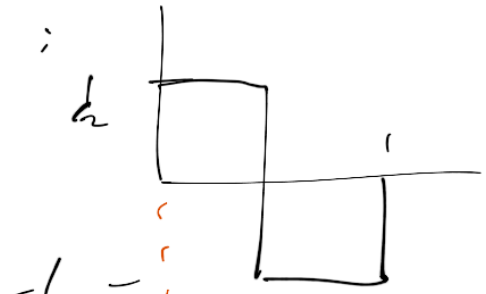
Quadrature mirror of

on test

- with box: $\phi(x)$:

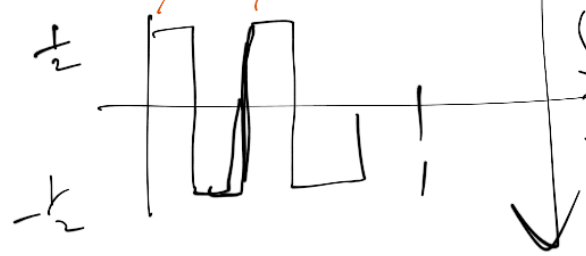
— low pass

$\psi(x)$:



— deriv. at a certain freq.

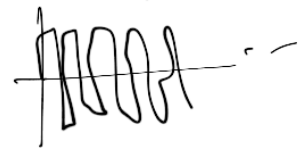
dilation :



MULTI-SCALE TRANSFORM

— deriv. at 2 × freq

;



and $\langle \phi, \psi \rangle = \emptyset$ they are orthogonal
zero via above construction

in general,

$$\phi_{j,k}(x) = 2^{j/2} \phi(2^j x - k)$$

$$\psi_{j,k}(x) = 2^{j/2} \psi(2^j x - k)$$

- powers of 2 mean <u>dyadic</u> dilation
(the $<<1$ shift operation in code)

- how does this get applied?

a 1D function

$$f^N(x) = f^{N-1}(x) + g^{N-1}(x)$$

where

$$f^j(x) = \sum_k c_{j,k}\,\phi(2^j x - k) \quad \Big\} \text{ scaling}$$

| scaling coeff (filter h)

$$g^j(x) = \sum_k d_{j,k}\,\psi(2^j x - k) \quad \Big\} \text{ wavelet}$$

$$\underbrace{\text{detail wavelet coefficient (filter } g)}_{\text{CONVOLUTION}}$$

DECOMPOSITION

- recursively,

$$f^{N}(x) = g^{N-1}(x) + g^{N-2}(x) + \cdots + g^{N-M}(x) + f^{N-M}(x)$$

- 1-level decomposition

$$f^{1}(x) = \underbrace{f^{0}(x) + g^{0}(x)}$$

$$\boxed{f^{1}} \rightarrow \begin{array}{|c|c|} \hline f^{0} & g^{0} \\ \hline g^{0} & g^{0} \\ \hline \end{array}$$

2- level:

$$f^{2}(x) = g^{1}(x) + f^{1}(x)$$

$$= g^{1}(x) + g^{0}(x) + f^{0}(x)$$

$$\boxed{f^{2}} \rightarrow \begin{array}{|c|c|c|} \hline f^{0} & g^{1} & g^{0} \\ \hline g^{1} & g & \\ \hline g^{0} & & g^{0} \\ \hline \end{array}$$

$$\vdots$$

$$f^{j}(x) = g^{j-1}(x) + f^{j-1}(x)$$

$$f^{j+1}(x) = g^{j}(x) + f^{j}(x)$$

$$= \sum d_{j,k}\,\psi_{j+k}(x) + \sum c_{j,k}\,\phi_{j,k}(x),$$

$$j, k \in \mathbb{Z} \quad \text{(integers)}$$

- schematically

$$\underbrace{C^N \xrightarrow{d^{N-1}} C^{N-1} \xrightarrow{d^{N-2}} C^{N-2} \cdots \xrightarrow{d^{N-m}} C^{N-m}}_{\text{decomposition}} \Big\} \text{ only decompose scaled fter}$$

reconstruction

$$C^N \xleftarrow{d^{N-1}} C^{N-1} \xleftarrow{} \cdots \xleftarrow{d^{N-m+1}} C^{N-m+1} \xleftarrow{d^{N-m}} C^{N-1}$$

e.g. H: low-pass filter, (ϕ)

Haar h_k: $\frac{1}{\sqrt{2}}\boxed{1 \mid 1}$

G: high-pass filter, (ψ)

Haar, g_k: $\frac{1}{\sqrt{2}}\boxed{-1 \mid 1}$

$$g_k = \underbrace{(-1)^k}_{\text{negate every other element}} \; \underbrace{h_{1-k}}_{\text{reverse}} \Big\} \text{ quadrature mirror}$$

e.g. some arbitrary random filter h_k:

h_k: $\boxed{-1 \mid 2 \mid 5 \mid 7 \mid 4 \mid 12 \mid -4}$

g_k: $\boxed{-4 \mid -12 \mid 4 \mid -7 \mid 5 \mid -2 \mid -1}$

– mathematically, decomposition at resolution level j:

$$f_\phi^{j-1}(x) = \sum_k h_k f_\phi^j(2x+k)$$

$$f_\psi^{j-1}(x) = \sum_k g_k f_{\boxed{\phi}}^j(2x+k)$$

the low-pass image at previous level

h_k, g_k are the 1D filters

- reconstruction is a bit trickier:

$$f^{j-1}_{\phi \bowtie \psi}(2x+p) = (1-p)\, f^{j-1}_{\phi}(x) + (p)\, f^{j-1}_{\psi}(x)$$

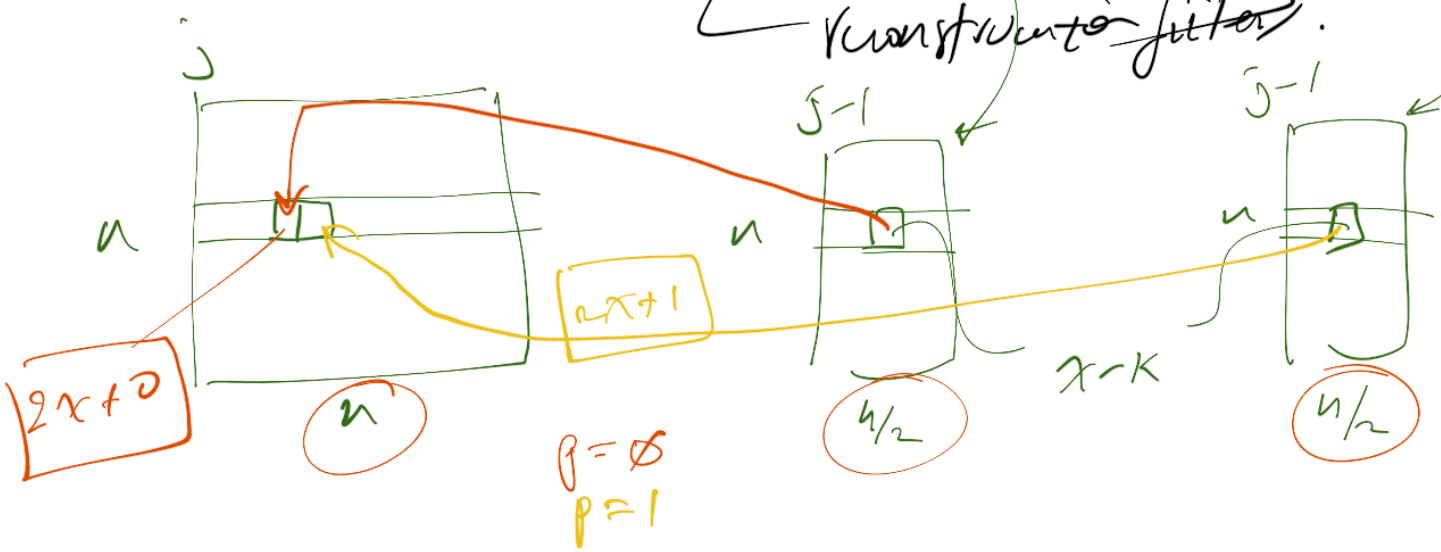
$\uparrow$ interleave

for $p \in \{0,1\}$

above is a pre-processing step, reconstruction is:

$$f^{j}_{\phi}(2x+p) = (1-p)\sum_k \bar{h}_k f^{j-1}_{\phi\bowtie\psi}(x-k) + (p)\sum_k \bar{g}_k f^{j-1}_{\phi\bowtie\psi}(x-k)$$

reconstruction filter.



- just a convenient way of writing:

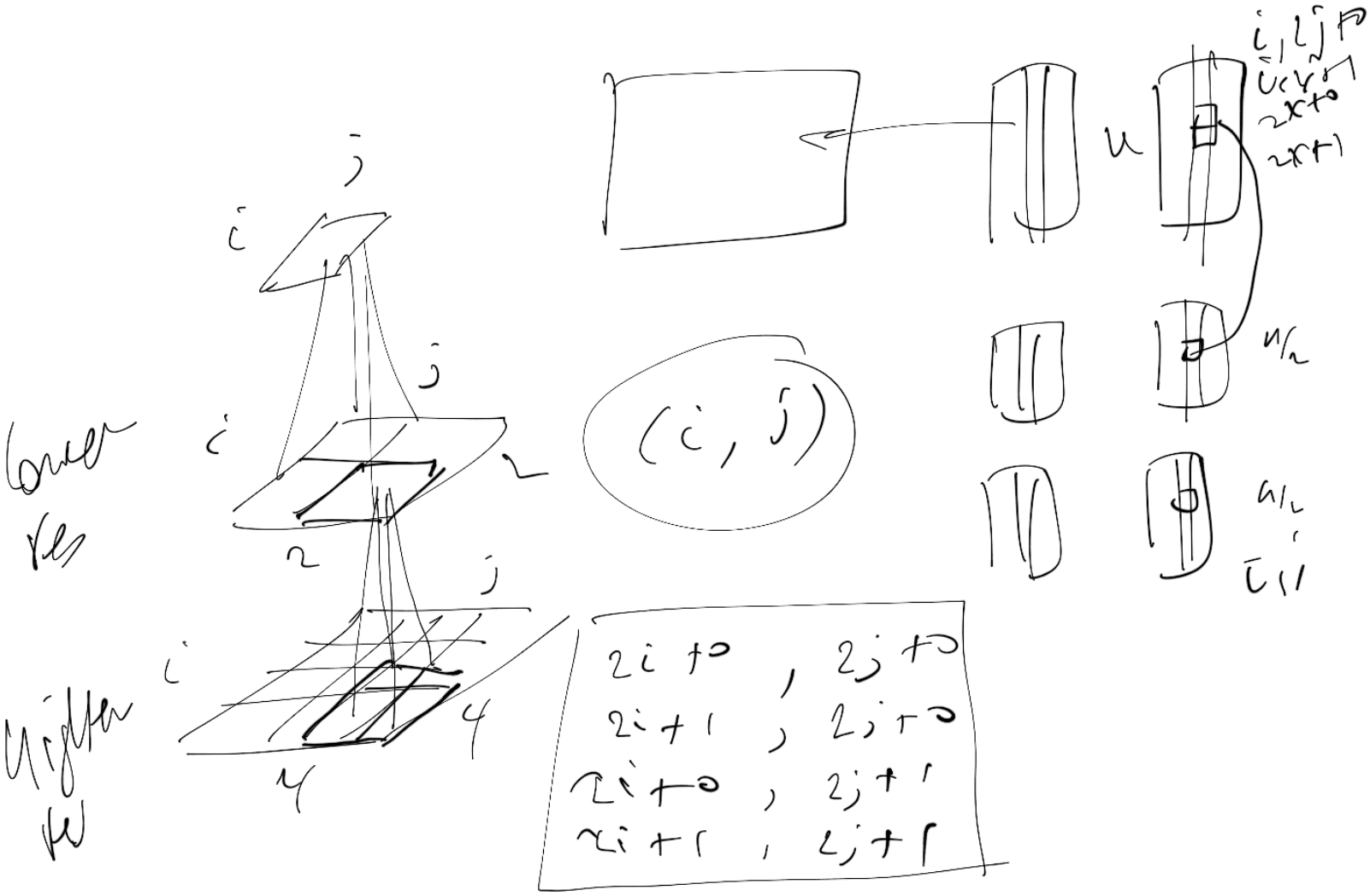
$$f_\phi^j(2x) = \sum_k \bar{h}_k f_{\phi \oplus \psi}^{j-1}(x-k)$$

$$f_\phi^j(2x+1) = \sum_k \bar{g}_k f_{\phi \oplus \psi}(x-k)$$

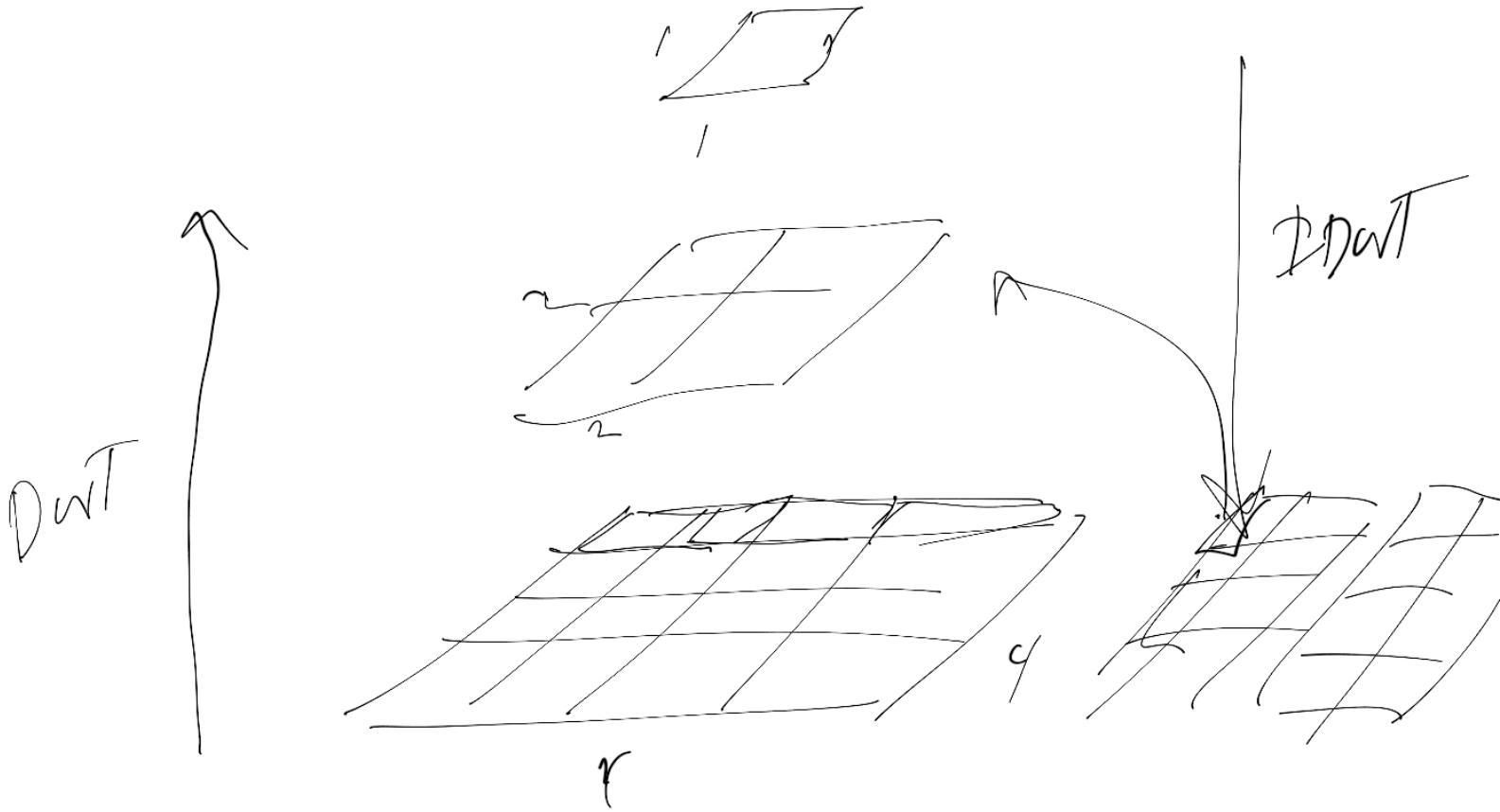
---

h_k: $\frac{1}{\sqrt{2}}$ $\frac{1}{\sqrt{2}}$ $\qquad$ $\bar{h}_k$: $\frac{1}{\sqrt{2}}$ $\frac{1}{\sqrt{2}}$

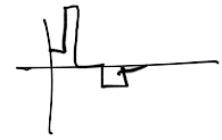
g_k: $\frac{1}{\sqrt{2}}$ $\frac{-1}{\sqrt{2}}$ $\qquad$ $\bar{g}_k$: $\frac{1}{\sqrt{2}}$ $\frac{-1}{\sqrt{2}}$



$2i+0$	$2j+0$
$2i+1$	$2j+0$
$2i+0$	$2j+1$
$2i+1$	$2j+1$


1
1
2
2
DWT
IDWT
4
4

- a numerical example in 1D :



$f^2 (= c^2)$: $\quad 1 \quad 4 \quad 0 \quad -2$

g_k

detail d^1 : $\quad \frac{1}{\sqrt{2}}(1) + \frac{-1}{\sqrt{2}}(4) \qquad \frac{1}{\sqrt{2}}(0) + \frac{-1}{\sqrt{2}}(-2)$

$$\boxed{\frac{1}{\sqrt{2}} \;\Big|\; \frac{-1}{\sqrt{2}}}$$

coeff

$$\frac{-3}{\sqrt{2}} \qquad \frac{2}{\sqrt{2}}$$

h_k

smooth pass c^1 : $\quad \frac{1}{\sqrt{2}}(1) + \frac{1}{\sqrt{2}}(4) \qquad \frac{1}{\sqrt{2}}(0) + \frac{1}{\sqrt{2}}(-2)$

$$\boxed{\frac{1}{\sqrt{2}} \;\Big|\; \frac{1}{\sqrt{2}}}$$

low pass

$$\frac{5}{\sqrt{2}} \qquad \frac{-2}{\sqrt{2}}$$

$\vdots$

$c^1 :$ $\frac{5}{\sqrt{2}}$ $\frac{-2}{\sqrt{2}}$

$d^0 :$ $\frac{1}{\sqrt{2}}\left(\frac{5}{\sqrt{2}}\right) + \frac{-1}{\sqrt{2}}\left(\frac{-2}{\sqrt{2}}\right)$ g_k

$\frac{7}{\sqrt{2}}$

$c^0 :$ $\frac{1}{\sqrt{2}}\left(\frac{5}{\sqrt{2}}\right) + \frac{1}{\sqrt{2}}\left(\frac{-2}{\sqrt{2}}\right)$ 4_k

$\frac{3}{\sqrt{2}}$

$$f^2 (= c^2) = \boxed{1 \mid 4 \mid 0 \mid -2}$$

$$W(f) = \boxed{\frac{3}{\sqrt{2}} \mid \frac{7}{\sqrt{2}} \mid \frac{-3}{\sqrt{2}} \mid \frac{2}{\sqrt{2}}}$$

$$\underbrace{c_0 \quad d_0}_{c_1} \quad d_1 \quad d_1$$

} reconstruct to get back to original

$$f = W^{-1}(W(f))$$

$$W(f): \quad \frac{3}{\sqrt{2}} \quad \frac{7}{\sqrt{2}} \quad \boxed{-\frac{3}{\sqrt{2}} \quad \frac{2}{\sqrt{2}}}$$

$$c_0 \quad d_0 \quad d_1 \quad d_1$$

$c_0 \bowtie d_0$

$$\frac{3}{\sqrt{2}} \quad \frac{7}{\sqrt{2}} \quad \frac{3}{\sqrt{2}} \quad \frac{7}{\sqrt{2}}$$

$$\bar{h}_k \quad \left(\frac{1}{\sqrt{2}} \;\middle|\; \frac{1}{\sqrt{2}}\right) \quad \left(\frac{1}{\sqrt{2}} \;\middle|\; \frac{-1}{\sqrt{2}}\right) \quad \bar{g}_k$$

$$\frac{1}{\sqrt{2}}\left(\frac{3}{2}+\frac{7}{2}\right) \quad \frac{1}{\sqrt{2}}\left(\frac{3}{2}-\frac{7}{2}\right)$$

$$\underbrace{\boxed{\frac{5}{\sqrt{2}} \quad -\frac{2}{\sqrt{2}}}}_{c_1}$$

$$c_1 \bowtie d_1: \quad \boxed{\frac{5}{\sqrt{2}} \;\middle|\; \frac{-3}{\sqrt{2}} \;\middle|\; -\frac{2}{\sqrt{2}} \;\middle|\; \frac{2}{\sqrt{2}}}$$

$c_1 \otimes d_1$:

$$\frac{5}{\sqrt{2}} \qquad \frac{-3}{\sqrt{2}} \qquad \frac{-2}{\sqrt{2}} \qquad \frac{2}{\sqrt{2}}$$

$$u_k \qquad g_k \qquad u_k \qquad q_k$$

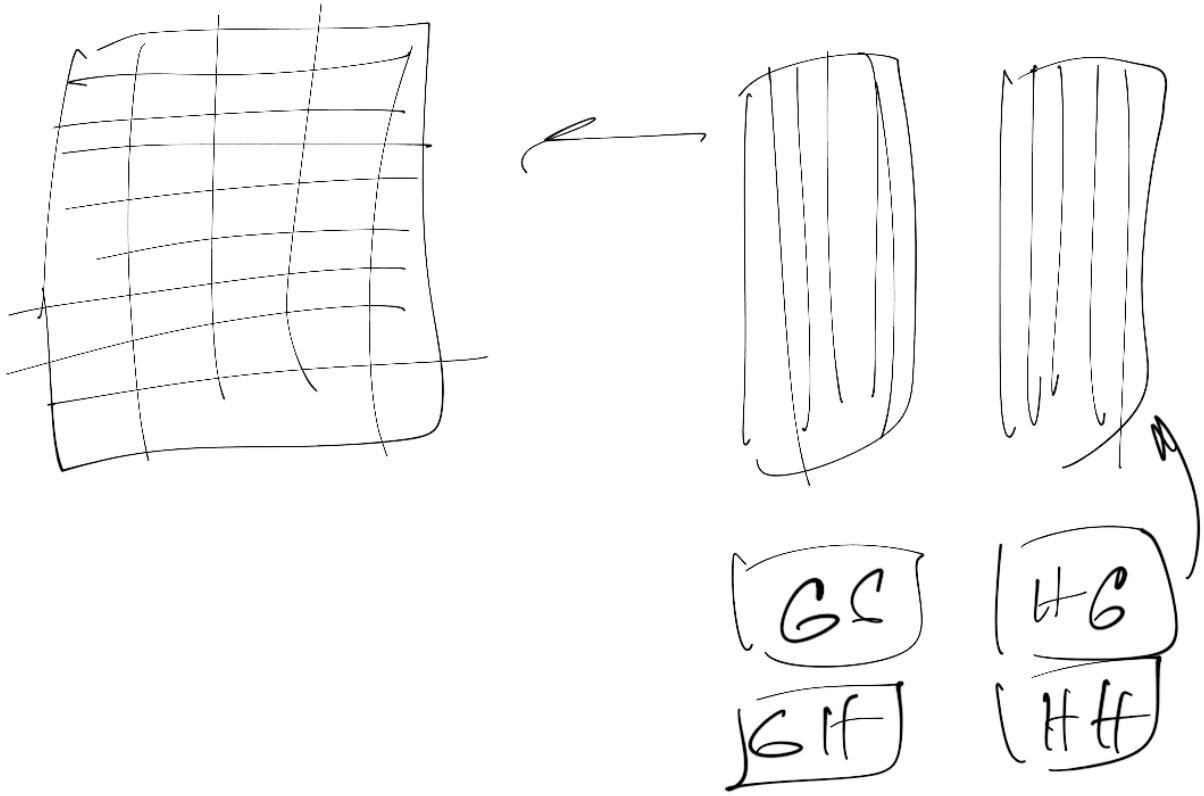
$$\frac{1}{\sqrt{2}}\left(\frac{5-3}{\sqrt{2}}\right) \qquad \frac{1}{\sqrt{2}}\left(\frac{5+3}{\sqrt{2}}\right) \qquad \frac{1}{\sqrt{2}}\left(\frac{-2+2}{\sqrt{2}}\right) \qquad \frac{1}{\sqrt{2}}\left(\frac{-2-2}{\sqrt{2}}\right)$$

$$\frac{2}{2} \qquad \frac{8}{2} \qquad \frac{0}{2} \qquad \frac{-4}{2}$$

c^2 : $\boxed{1 \mid 4 \mid \emptyset \mid -2}$

$\overset{\shortparallel}{f^2}$ ✓

- in 2D :



- DWT code snippets
- main() is similar to sobel

```
pgm_dwt2D(ging, levels)
// visualization
ning = pgm_copy(ging)
pgm_normalize(ning)
// output ning
// optional: do something interesting with ging
```

DWT

e.g. "simulate compression"

pgm_idwt2D(ging, levels) // inverse transform

// note: you could copy ging, do MSE — is there a diff between orig. & IDWT(DWT(orig))

$$MSE = \frac{1}{mn} \sum_{i,j}^{m,n} \left(I_1(i,j) - I_2(i,j) \right)^2$$

- per-pixel squared difference

- pgm_dwt2D (PGM * img, int levels)

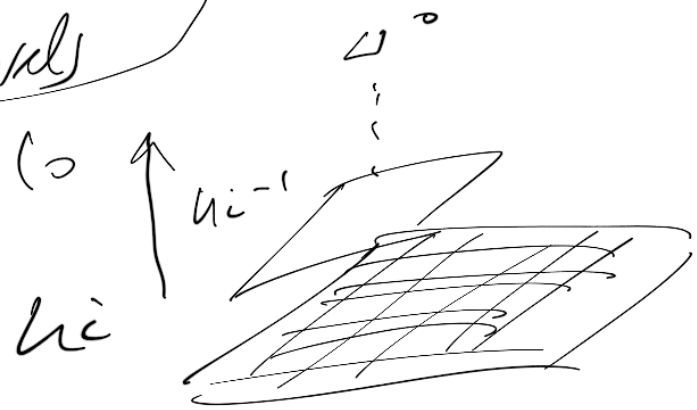
int $\underline{\underline{hi}}$ = ilog2

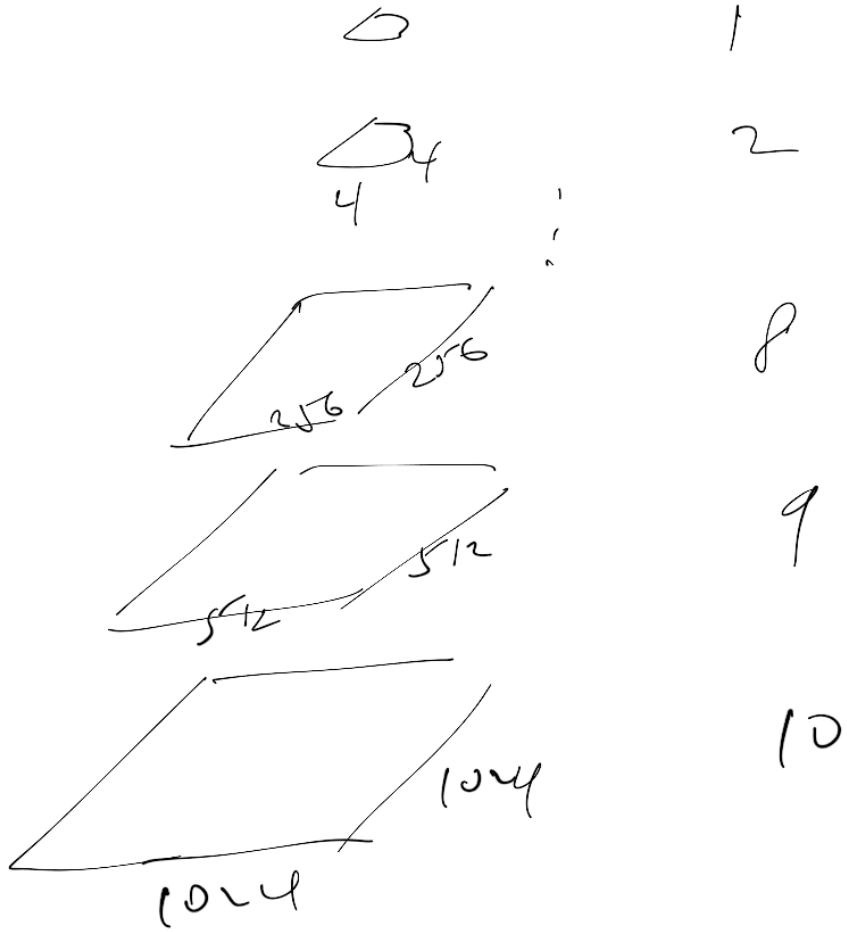
pgm_dwt2D(PGM * img, int levels)

int hi = ilog2(img->rows) { assuming square, POT image

$$\frac{\log(\text{rows})}{\log(2.0)} = \log_2(\text{rows})$$

int lo = hi - levels




1
2
4
4
8
256
256
9
512
512
10
1024
1024

int hi, lo are levels of the $n \times n$ POT image <u>pyramid</u> (DWT <u>is</u> a pyramidal image transform)

int levels = $\emptyset$

means to do it once

(1-level decomp)

computer graphics

$\rightarrow$ MIP-mapping

WVLT *wf = make_wavelet(HAAR)

wavelet browser — Haar, Biorthogonal, symlet

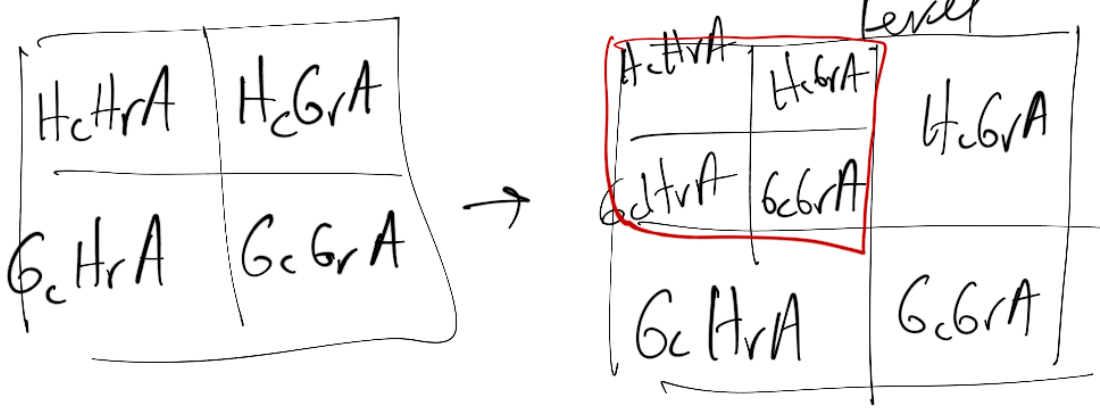
dwt2D(img → pix, img → rows, img → cols, wf, hi, lo)

- dwt2D() , idwt2D() functions

dwt2D(double *I, int rows, int cols,
WVLT *wp, fnlvl, cslvl)

fine level

coarse level



recursively decompose upper-left 0,0 at upper left

- some helper indices:

int $\underbrace{i, j}_{\text{pixel indices}}$, $\underset{\text{filter}}{m}$, $\underbrace{r, c}_{\text{row, col}}$, $\underbrace{K}_{\text{res. level}}$

int roff, coff; row, col <u>offsets</u> (+P)

int i2, j2, r2, c2; // i/2, j/2, r/2, c/2

int ii, jj; // 2i, 2j

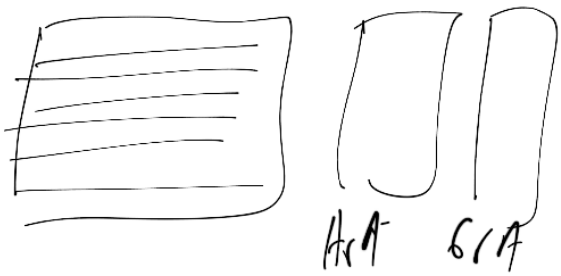
```
r = rows;  roff = r >>1;      // r/2
c = cols;  coff = c >>1;      // c/2

for k = fnlvl; k >= cslvl; k--
        r2 = r >>1;
        c2 = c >>1;
tmp
images ( HrA : [r, c2] array set to ∅
         GrA : [r, c2]  "    "   "   "
```

//pass/

for $i = 0$; $i < r$; $i{+}{+}$

for $j = 0$; $\boxed{j < c2}$; $j{+}{+}$



for $m = 0$; $m <$ cap→len; $m{+}{+}$

$jj = j << 1$;

$HrA(i,j) = \sum_m h_m I(i, \boxed{jj+m})$

same as $2j+m$

$GrA(i,j) = \sum_m g_m I(i, jj+m)$

(fill in K loop) pass 2

```
for i=0; i < r2 ; i++
    for j=0; j < c2; j++
        I(i      , j      ) = 0.0;
        I(i+roff , j      ) = 0.0;
        I(i      , j+coff) = 0.0
        I(i+roff , j+coff) = 0.0.
```

for $m = 0$; $m < wp \to len$, m ++

$ii = i << 1$; // $2i$

$$I(i, j) = \sum_m h_m HrA(ii+m, j)$$

$$I(i+roff, j) = \sum_m g_m HrA(ii+m, j)$$

$$I(i, j+coff) = \sum_m h_m GrA(ii+m, j)$$

$$I(i+roff, j+coff) = \sum_m g_m GrA(ii+m, j)$$

+= in code

- bottom of main k loop:

```
free(H,A)
free(G,A)
r >>= 1;    roff >>= 1;
c >>= 1;    coff >>= 1;
}
```

- idct2D() I leave to you.

it's similar, BUT you need to interleave rows, cols in between

(extra subtop)

- otherwise, idct2D() works backwards
- I still use tmpA, GrA temp images
- I also use rA, cA as interleave temp images

- for asg03:
  - implement idwt2D
  - show you can reconstruct image
  - optional: try MSE

- Moving on to OpenCV, video processing
  some image processing

- mainly in Python 3 (3.7)
  ↳ I have this installed

py37-numpy on mac via macports
" - scipy (need XCode + Command Line Tools)
" - opencv? (X11 Quartz)

- ffmpeg — command-line tool for transcoding video
- what is video?
  - .mpg — MPEG : Motion Pictures Expert Grp
  - .mp4 — MPEG 4, most popular today
  - .avi — ?? Window favorite
  - .mov — Mac popular

- movie file extensions: not super meaningful
- because they're all just <u>containers</u>
- really it's codec stuff inside that matters

coder – decoder

h.264 is one of the most popular

.avi, .mp4, ...

- video stream — video codec
- audio stream — audio codec
- subtitle stream — "burned in" or not
- AD stream
  - Audio Description — for the blind

- what matters are synchronization (frame rate, bit rate), <u>codecs</u>

- subtitles: - record video, speak slowly, clearly
  - upload to YouTube
  - get YT to transcribe audio ⟹ subtitles
  - download subs (.srt file) (edit)
  - render into video

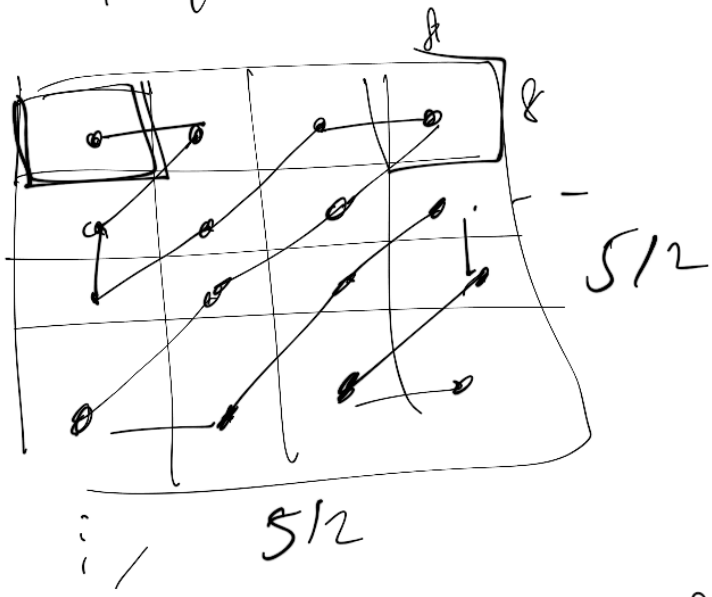
- H.264 : still DCT-based (?)

DCT: Discrete Cosine Transform

(not quite Fourier $\begin{pmatrix} \cos \\ \sin \end{pmatrix}$)

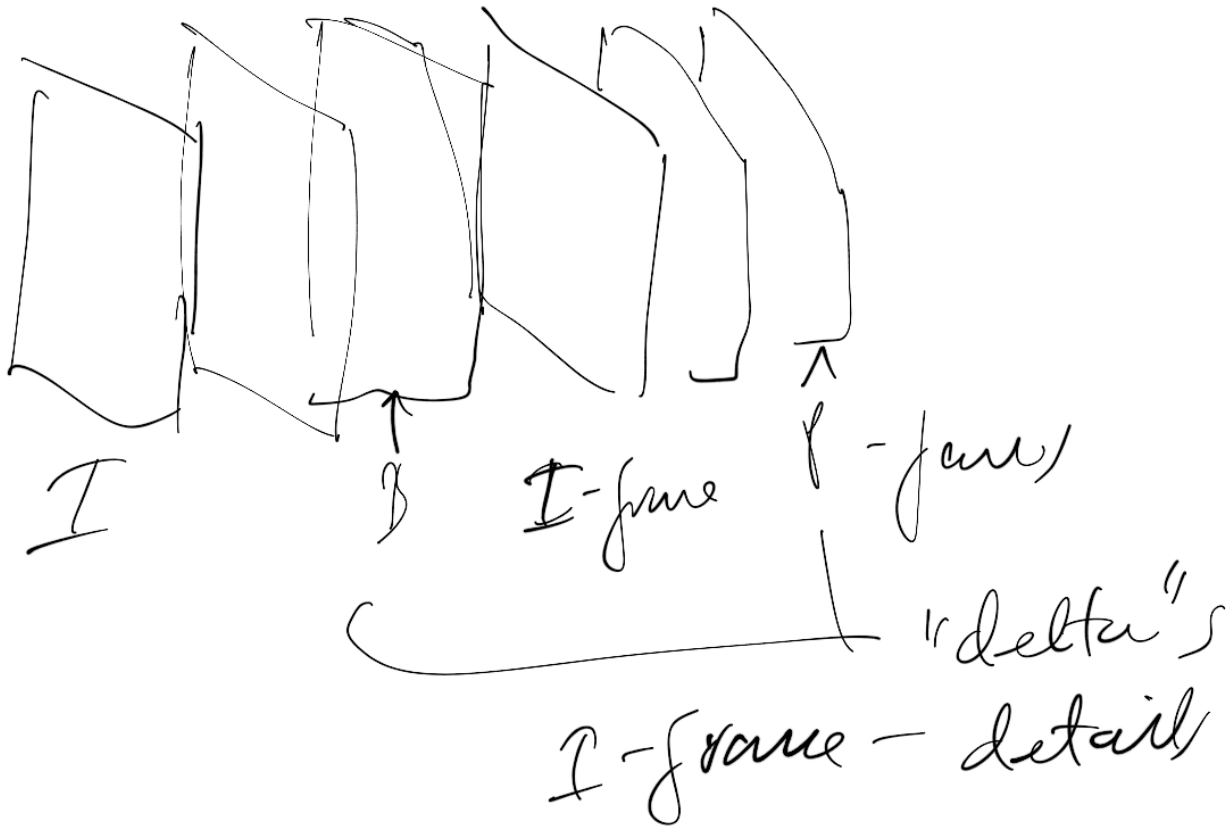
cos as the basis function.

- every I-frame fully encoded in "space"



- zig-zag pattern on <u>blocks</u>, e.g., 8x8

- MPEG: $\longrightarrow t$



- why the zig-zag pattern? DCT
- because longer runs of <u>zero coefficients</u>

"DC"

lowest freq.

197	10	0	0
23	5	0	0
0	1	0	0
0	0	0	0

run-length encode

$\Rightarrow$ 197, 10, (2x0), 23, 5, (10x0), EOB

END OF BLOCK

- ffmpeg: install from ffmpeg.org
  (or via macports or apt-get)
- get ffplay, ffprobe
- lot of programs built on top of this:
  VLC, Handbrake

- OpenCV :

  install : source code probably best

  want : contrib modules (aruco lib)

- latest version ? 4.5.1 ?

  4.4.0

- source code install:
  - see PyImageSearch
- on Mac, often need to do this as root (sudo) in /usr/local/src
- get opencv.tar.gz & contrib.
- need <u>ccmake</u>

- gunzip, untar, put into
  /usr/local/src/opencv
- contrib go into /modules/contrib
- mkdir build
- cd build
- cmake ..

asg03 : extra stuff:

- MSE
- wavelet decimation $\Longrightarrow$ compression "sim."
- edge detection via detection of

$\boxed{\text{modulus maxima}}$

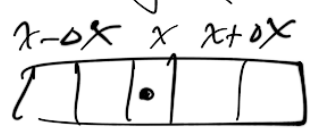
(in my notes – PDF)

- modulus maxima: wavelet coefficients larger or equal to neighbours: (1D)

$$Wf(x-\Delta x) \leq \underline{\underline{Wf(x)}} \geq Wf(x+\Delta x)$$

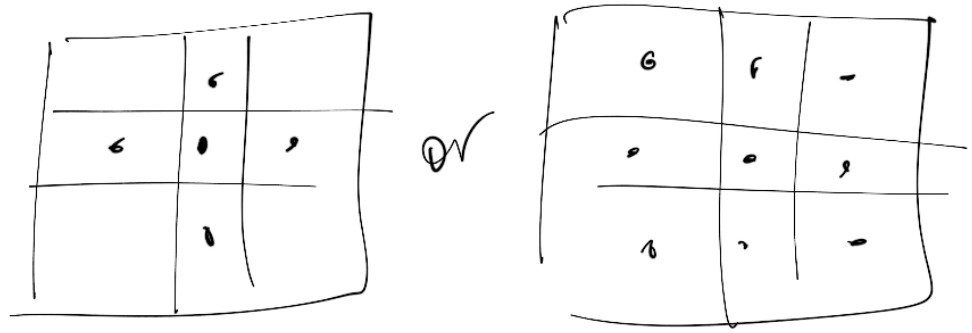
AND

$$\begin{cases} Wf(x) > Wf(x+\Delta x) \\ Wf(x) > Wf(x-\Delta x) \end{cases} \quad \underline{\underline{OR}}$$

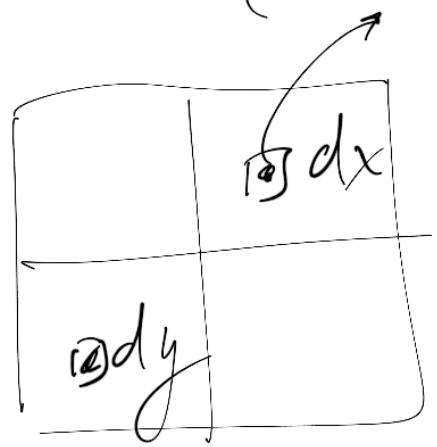


" $\nabla f(x_0)$ is greater than or equal to both of (2) (1) neighbors, and strictly greater than one of them."

in 2D:



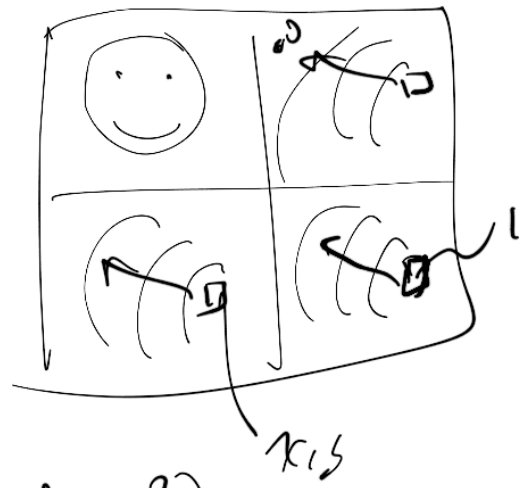
- can also get θ : $\operatorname{atan2}(dx, dy)$

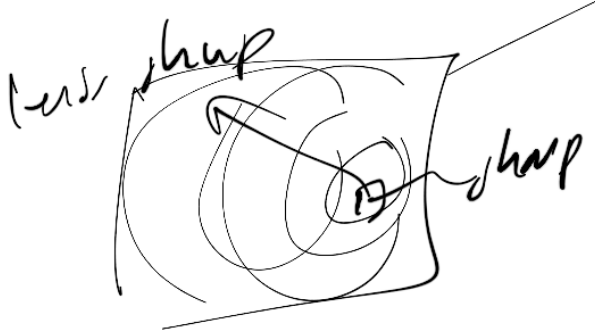


- get another interesting DWT filter:

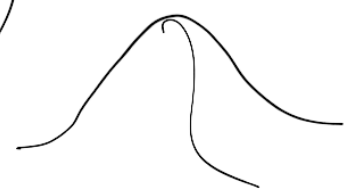
<u>anisotropic filtering</u>

not the same everywhere





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



- back to OpenCV (& Python)
- once you have OpenCV & python libs installed:

```
import sys, math
import numpy as np
import cv2
```

img.py

- Quick example: display image

```
img = cv2.imread(argv[1], 0)
cv2.imshow('image', img)
k = cv2.waitKey(0)
if k == 27:
    cv2.destroyAllWindows()
```

- can use cv2.imwrite(...)

- Next thing: display video:

```
cap = cv2.VideoCapture(0)
```

laptop camera

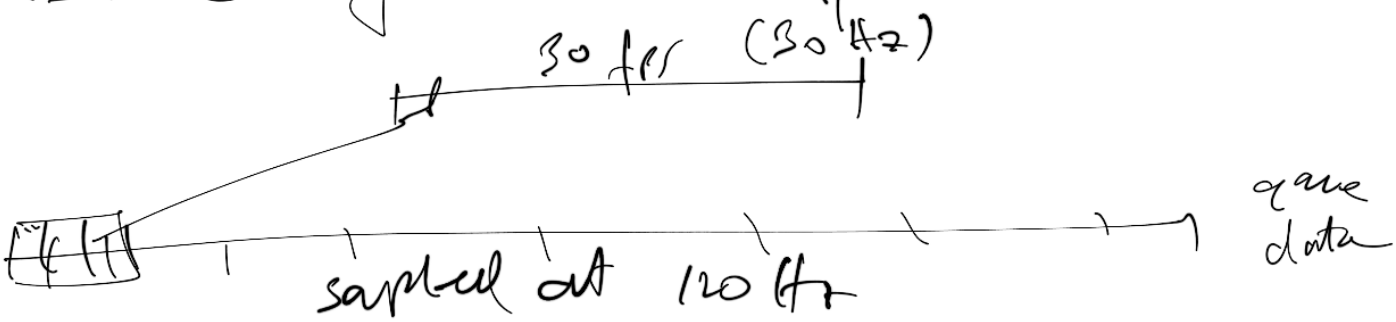
or 'file' e.g. 'file.mp4'

- with cap, interesting stuff:

```
size = (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
```

```
fps = cap.get(cv2.CAP_PROP_FPS)
```
(might not be correct)

```
frame_rate = 1./fps
frames = cap.get(cv2.CAP_PROP_FRAME_COUNT)
total_time = frame_rate * frames
```



- I've used this before:

$$t_{dilation} = \frac{\frac{fps}{data.duration()}}{\underbrace{(frame\_rate * frames)}_{video\ duration}}$$

- display video : need frames

```
if cap.isOpened():
    ret, frame = cap.read()
    if ret is True:
        frame = cv2.resize(frame, size, 0.0,
                    cv2.INTER_CUBIC)
        grey = cv2.cvtColor(frame,
                    cv2.COLOR_BGR2GRAY)
        cv2.imshow('vid', grey)
```

- in the loop also:

ms = cap.get(cv2.CAP_PROP_POS_MSEC *
fps / tdilation)

for timestamp at bottom left

asg04

- video copy assignment using OpenCV
- Python3 (C++ ok) — email me
- CHOOSE A VIDEO — short, not too big dim. (640 x 480 ...)
  (.mp4)

- to output video:

.mp4

```
⋮
cap = cv2.VideoCapture(infile)
size = (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
if output:
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    fps = 25.0  # why did I do this?
                # should be same as input
    out = cv2.VideoWriter('output.mov',
            fourcc, fps, size, True)
```

```
while (cap.isOpened():
    ret, frame = cap.read()
    if ret == True:
        # do stuff — RGB -> GREY ... ...   face detection

        if output:
            out.write(frame)

        cv2.imshow('frame', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

```
# end of while
cap.release()
if out:
    out.release()
cv2.destroyAllWindows()
```

(asg 05 – face detection) (& tracking)

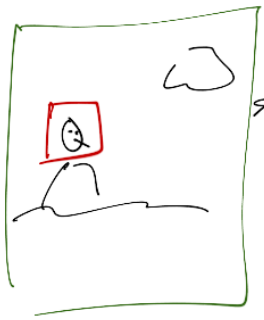
"face is there"

"history"

simple approach

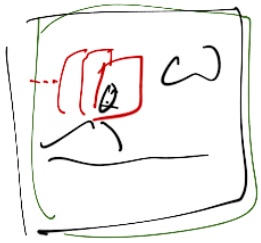
+ opencv

↳ zoom filter



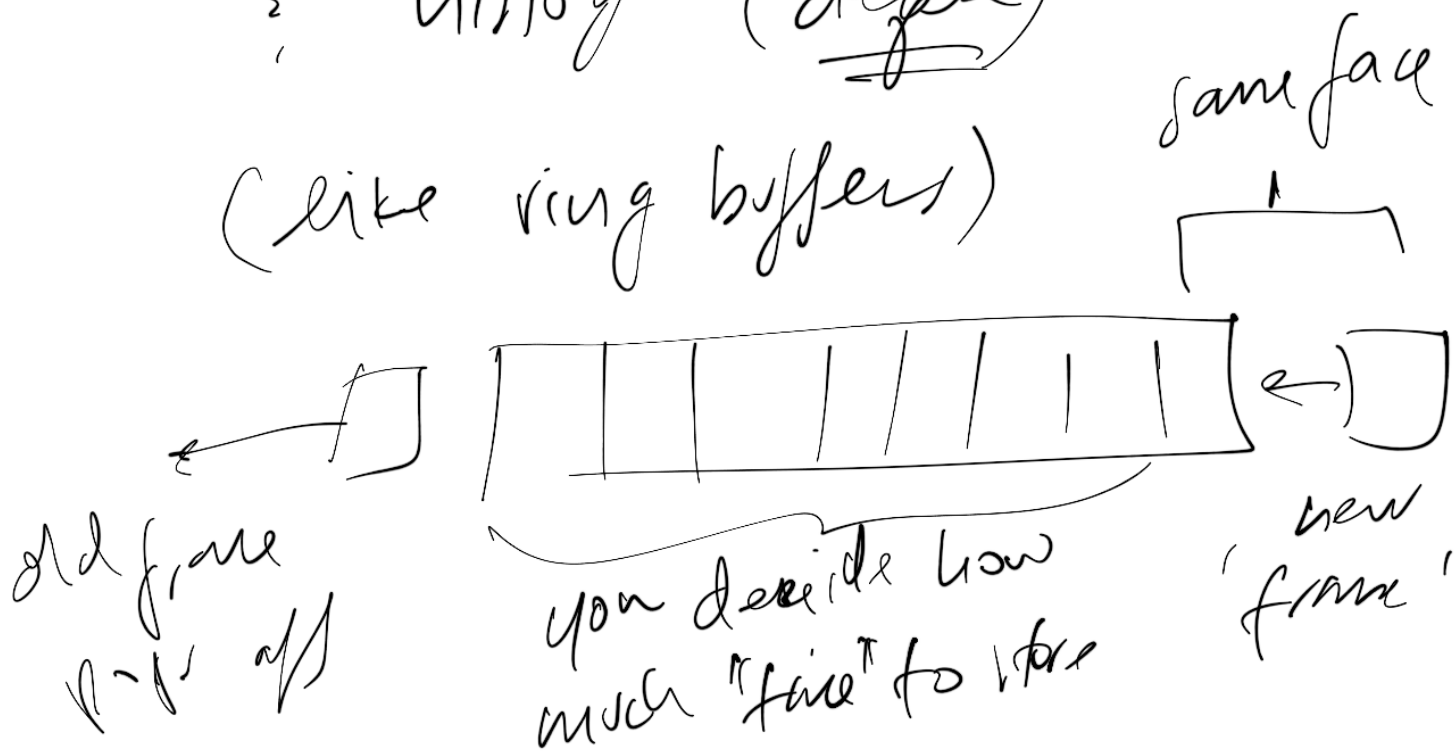
"frame coherence"

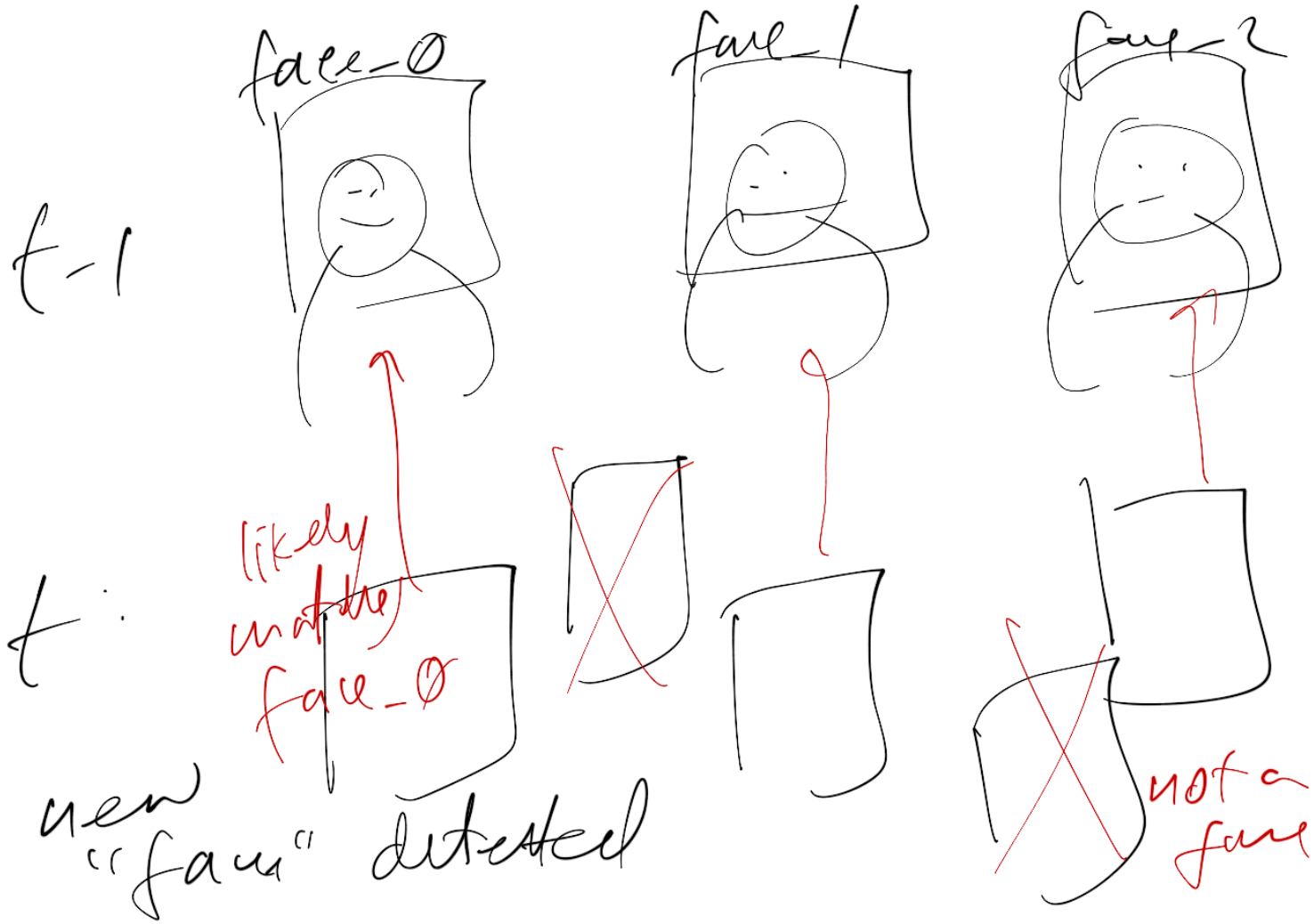
→ remembering that we had face on previous frames



- face tracking: use object for face & "history" (degree)

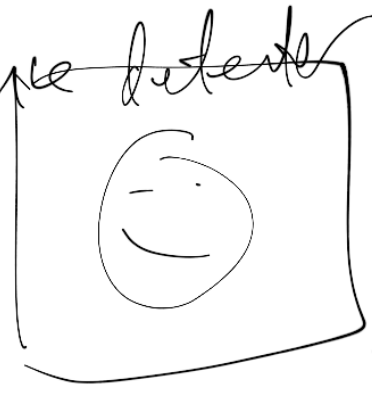
(like ring buffers)




face_0
face_1
face_2
t-1
t:
likely matches face_0
new "face" detected
not a face

- asgOr:

a) face detector

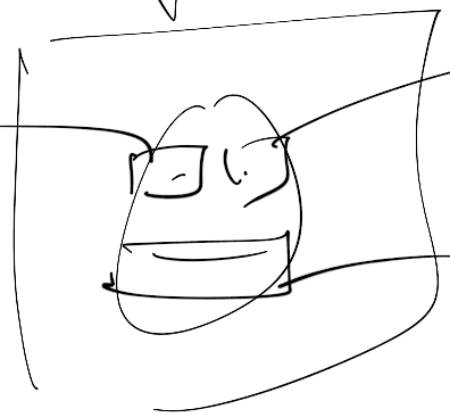


face

Haar cascading filters

b) facial features

individual detectors for each feature



eye_l

eye_r

mouth

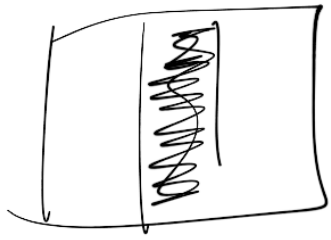
- Haar cascading filters
- Viola-Jones 2001 paper
- machine-learning approach
  - trains detectors for face, eyes, nose, mouth → .xml files in OpenCV distro

- Viola-Jones
- 3 advancements
  1. use an integral image
     allows for fast computation
  2. learning algorithm based on AdaBoost
     → selects small no. of features
  3. combination of classifiers into a cascade

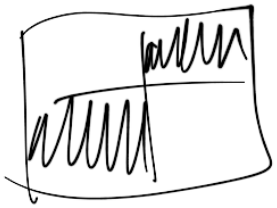
1. Integral image : based on
Haar basis functions
- computed at many scales
- sound familiar ?
- Haar cascading filters
- Haar basis <u>features</u> — not pixels

- 3 kinds of features
  - 2 rectangle

  - 3-rect



  - 4-rect

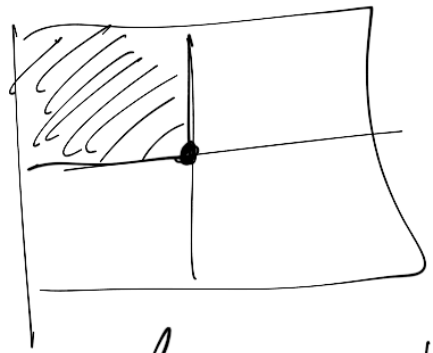


} sum of pixels

- detector is 24×24
- unlike wavelet Haar bases, these <u>overlap</u> ← over-complete

- Compute Haar-like sums, use integral image:

$$ii(x,y) = \sum i(x',y'), \quad x' \le x, \; y' \le y$$



- similar to summed area tables in graphics

2. learning classification functions

- need a feature set: +ve, -ve training images

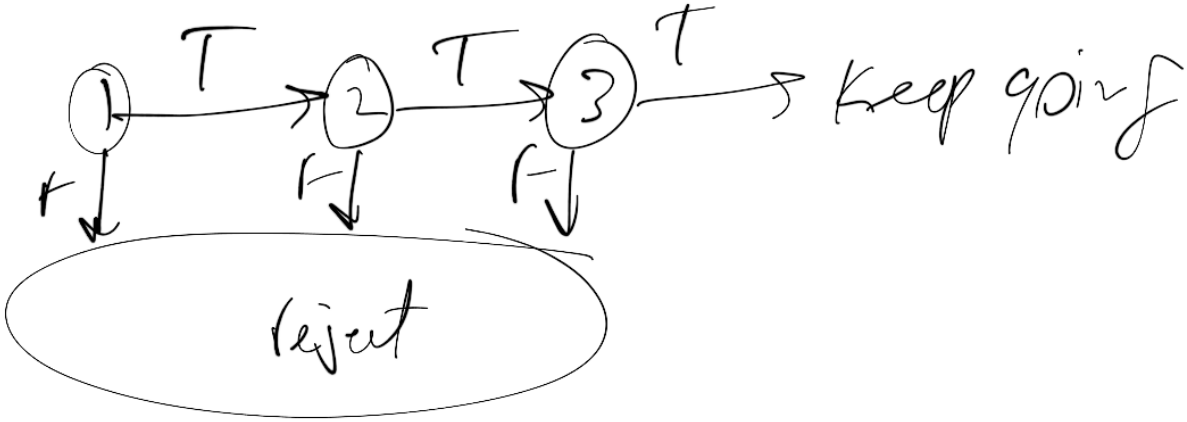
- a weak classifier is used

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$

- need AdaBoost to <u>learn</u> (classifier)

3. <u>Attentional</u> cascade

- cascade is a decision tree
- cascaded triggering of classifier



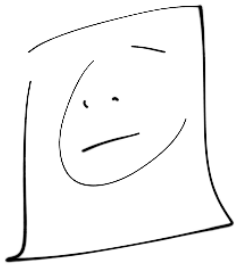
- so where are those cascaded filters?

/usr/local/src/opencv/data/

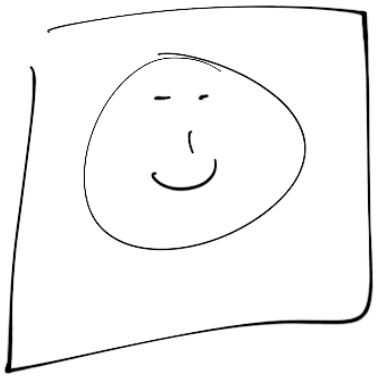
haarcascades/*.xml

- for asg05 — use Haar cascades for face detection — you set a box

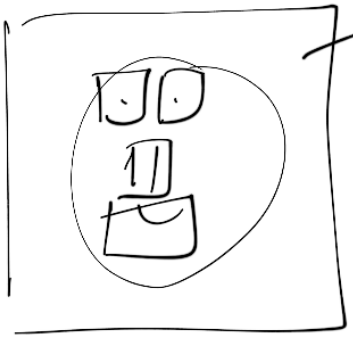
- then, within this box, use cascaded filters for eyes, mouth, nose



- 1.



- 2.



you will get a bunch of false +ves

- expect a <u>list</u> of rectangles for <u>each</u> feature

0+

- code looks like this:

```
face_cascade = cv2.cascadeClassifier(
    "haarcascade_frontalface_alt2.xml")
lefe_cascade = cv2. . . .
    "haarcascade_lefteye_2splits.xml"

reye_cascade = . . .
nose_cascade = . . .
mouth_cascade = . . .
```

- then use classifier

Use gray

faces = face_cascade.detectMultiScale(~~frame~~, 1.3, 5)

I forget what these are

for (x, y, w, h) in faces: (roi = region of interest)

    roi_gray = gray[y:y+h, x:x+w]

reduced search space { leye = leye_cascade.detectMultiScale(roi_gray, ...)

... same for reye, nose, mouth

→ you get lots of noses, mouths, lewer, reye)

— this is done <u>per frame</u>

(the <u>detection</u> part of algos)