

CPSc 4820/6820

- Basic image info, I/O

- what's an image?

(or video frame)

magic number

P5

comment

width height

max

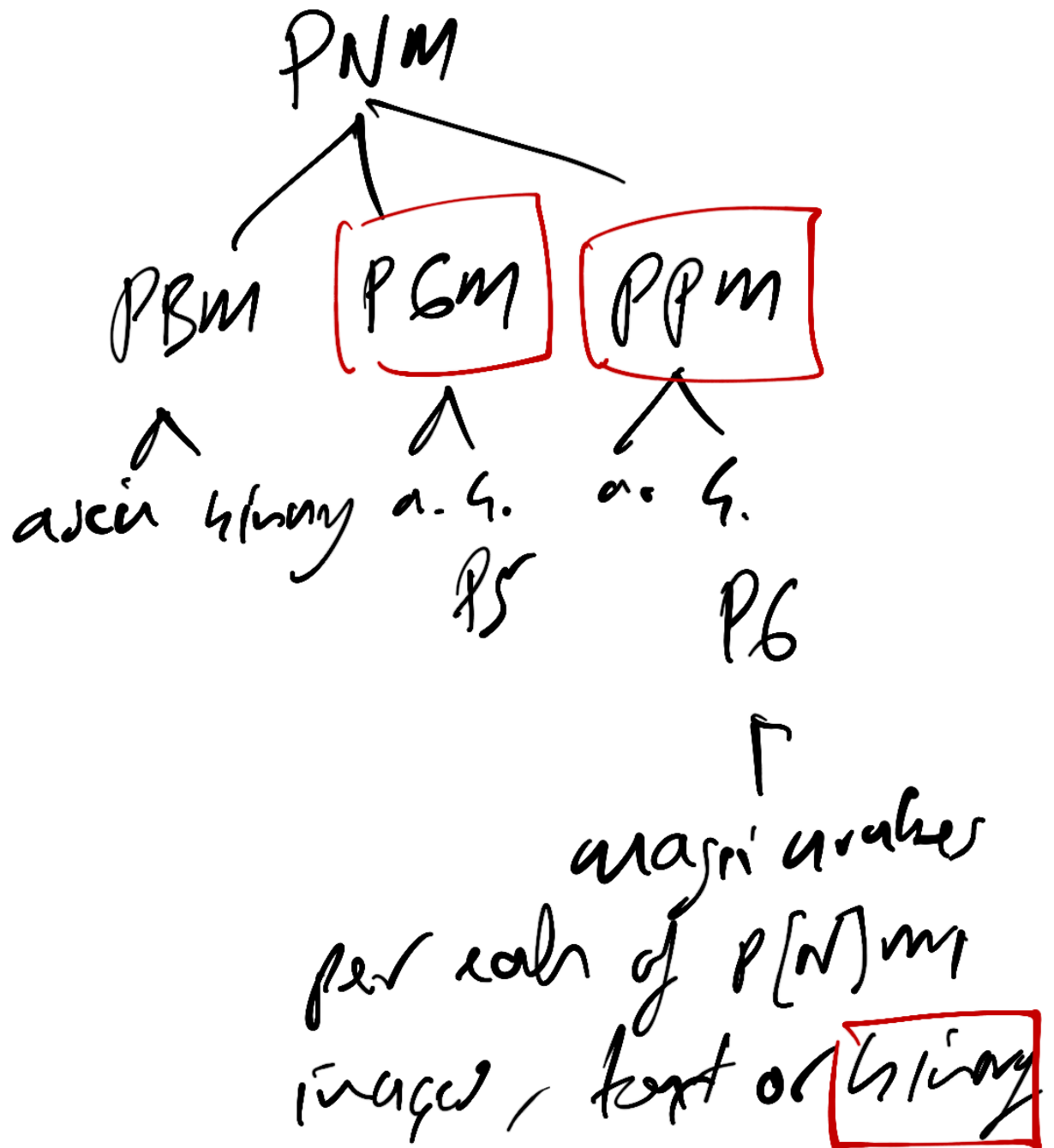
99999...

} header

} payload /
data
(pixels)

2D array / matrix

- Netpbm (old pbmplus)



PC
 # comment
 c r
 max # usually 255
 r g b r g b . . .

aj.

PC
 # mandril
 252 252
 255
 r g b r g b . . .

- what we want is :

- a) Conceptually a 2D array (float)
- b) implemented a 1D array

- 2D access to pixels:

```
for (i = 0; i < rows; i++)
```

```
    for (j = 0; j < cols; j++)
```

```
        img → data[ $i \times \text{cols} + j$ ]
```

2D access into 1D

- we can allocate image payload
float *data =

(float *) malloc(r * c * sizeof(float));

- when reading in, read data
as unsigned char (8 bits)
per channel

- simple way (i.e. C) :

fread (&data, sizeof(unsigned char), 1, in)
in & data[i * cols + j] = (float) data / (float) max

- max is usually 255
8 bits per channel

0000 0000 : black

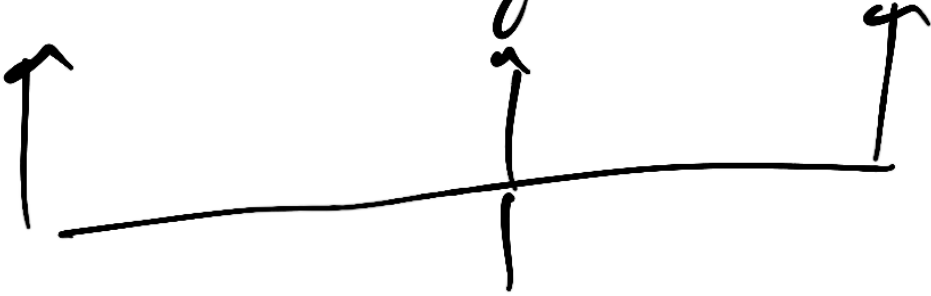
1111 1111 : white (or red, gr. & blue)

- $3 \times 8 = 24$ bits per pixel
(color ppm)

- other formats: lots of them

ex. 422 : 4 b Y
2 b U
2 b V

- for asypt, $hbb \rightarrow G$
3 channels to 1:

$$g = 0.21 * r + 0.72 * j + 0.07 * s;$$


$e [0,1]$

- when writing
unsigned grey;

grey = (unsigned char)
(img → data[i * cols + j] * max)

image
scaling

$(0,1] * max \in [0,255]$
fwrite(&grey, sizeof(unsigned char), 1, out);

- 2D $\rightarrow$ 1D macro on steroids

define $I(i, r) ((r) + (i)) \% (r)$

define $J(j, c) ((c) + (j)) \% (c)$

define $M2A(i, i, r, c)$

$(I(i, r) * (c) + J(i, c))$

same as $k \times r$

$i * c + j$

- why the $()'$

- So we can do stuff like

$(i+1), (i+2), \dots, (i+k)$
↑

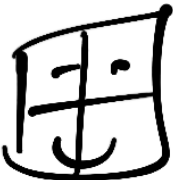
$(i-1)$
also ok.

can exceed
image boundaries

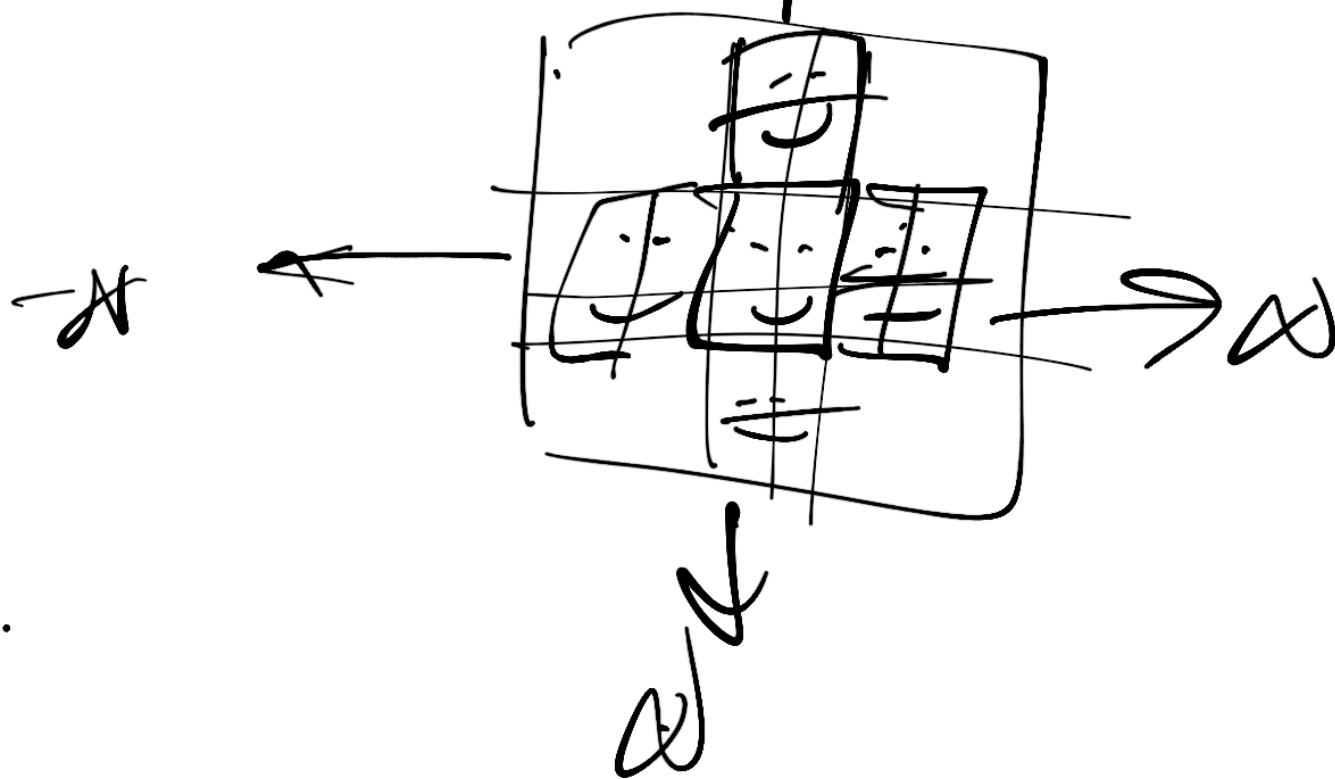
- like 19 Apr's $[-1, k]$
↑

negative index.

- "wrap around indices"

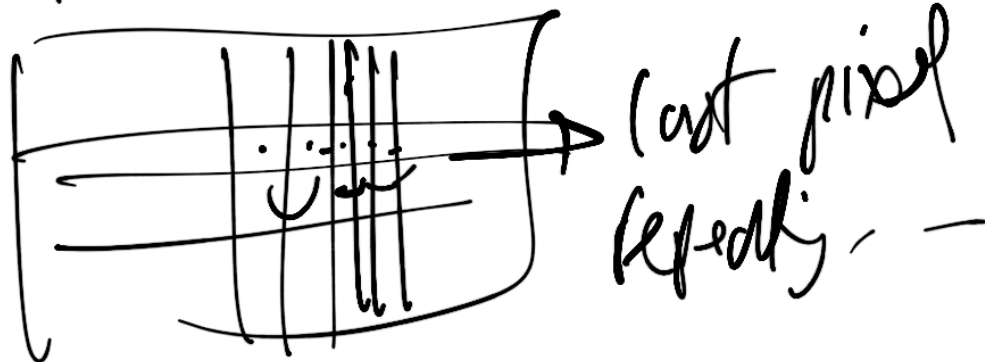
 2x2 image

↑ - 10



periodic extension to ∞

v.s. clamp



Jan. 14

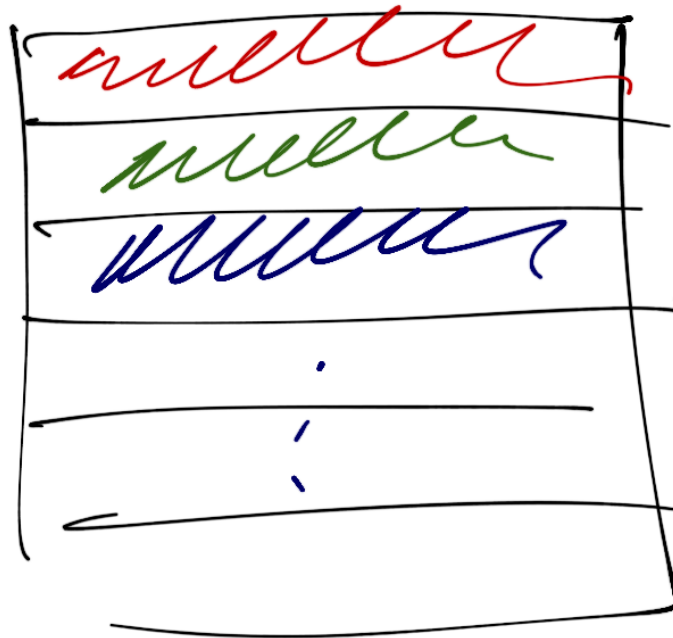
- OpenMP :

making use of multi-core
architecture

- omp.tar.gz

code produces color, ppm

Color-
banded
image



} each
band
executed
by a
core

— "normally" you 'touch pixels'
via nested for loop:

```
for (int y = h - 1; y >= 0; y--) {  
    for (int x = 0; x < w; x++) {
```

$R[y][x] = 1.0;$

$G[y][x] = 0.2;$

$B[y][x] = 0.2;$

$\vdots$

}

|

A 6 B
0 0 0 — black

0 0 1 — blue

0 1 0 — green

0 1 1 — cyan

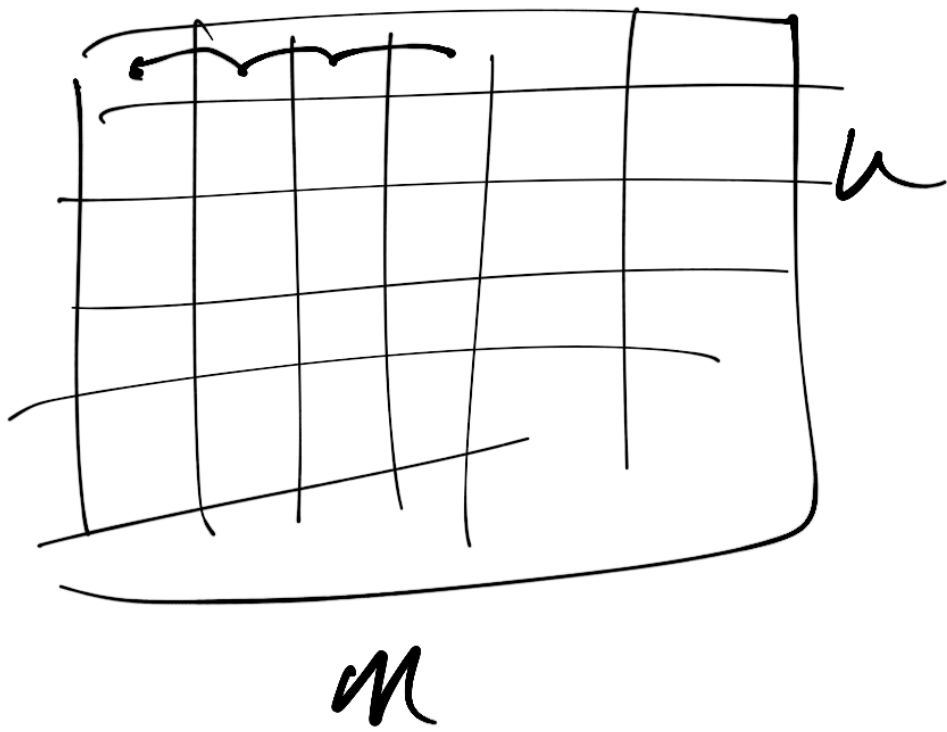
1 0 0 — red

1 0 1 — magenta

1 1 0 — yellow

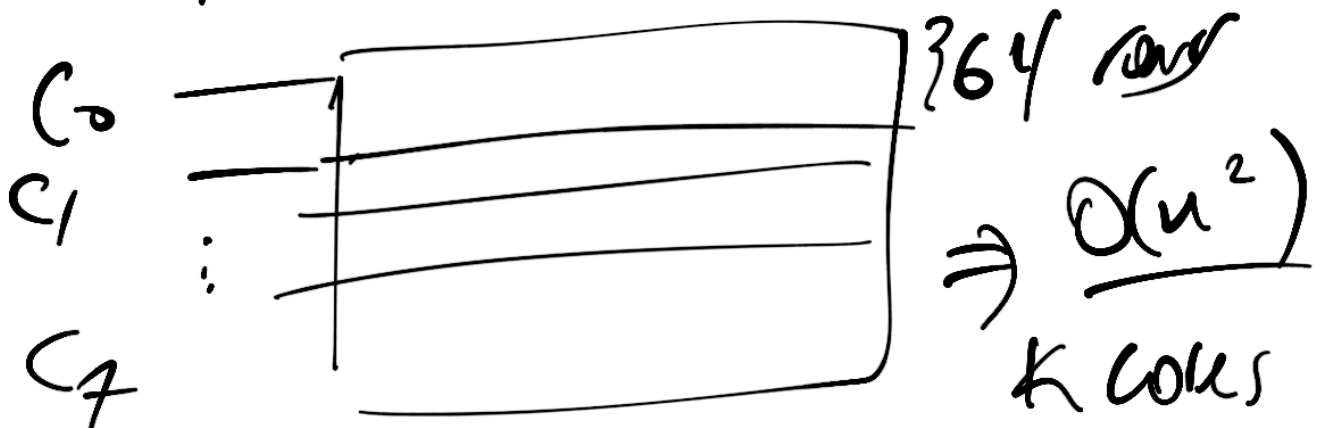
1 1 1 — white

255 0 0



$$= O(n^2)$$

- but, say $n = 512$
(POT: power of Two)
- with 8 cores (8 CPUs)
split up work into bands;



- parallel execution :
each core runs indep.
and at the same time
(concurrent processing)

- SIMD execution

Single instruction Multiple Data

- key problem : access to
shared memory.

- multiple reads OK.

- must have mutual exclusion for
writers

- need mutual exclusion for
writing into shared memory
(our image array)

- luckily OpenMP does this
for you, you just need
to: (a) specify what memory
is shared and (b) compile
with openmp

nowait

pragma omp for schedule (static, chunk)

for (int y = h-1; ---) {

for (int x = 0; ---) {

what this does:
"unrolls" the for loop
& sends it to each

core
"inside loop":

tid = omp_get_thread_num();

switch (tid % 7) {

⋮

- chose the for loop:

#pragma omp parallel

shared(w, h, R, G, B)

private(tid)

shared
image
arrays

- and chose this:

#pragma omp parallel private(tid)

{
if ((tid = omp_get_thread_num()) == 0)
ncore = omp_get_num_threads();
}

- before that:

```
#include <omp.h>
```

```
int main()
```

```
{    int tid, ncores = 1, chunk;
```

```
    :
```

```
    chunk = h / ncores; // rows per  
    :                  color band
```

```
}
```


- pixel transforms

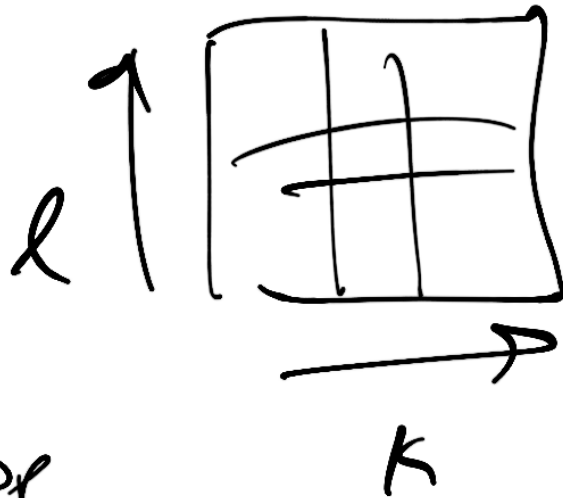
- Szuliski Sect. 3.2, 3.2.1

- let 2D image be represented as

$$\vec{x} = (\underbrace{i, j}_{\text{pixel locations}})^{(c)} \quad \begin{array}{l} i \in M, \\ j \in a(r) \end{array} \quad \begin{array}{l} m \times n \\ \text{image} \end{array}$$

$$\begin{array}{ccc} \nearrow \text{output image} & \uparrow \text{operator} & \underbrace{f(i, j)}_{\text{source image}} \\ g(i, j) = h(f(i, j)) & & \end{array}$$

- linear operator $h(k, l)$



e.g. 3×3
operator

"mask"

for loop

$$\underline{g(i, j)} = \sum f(\underline{i+k}, \underline{j+l}) \underline{h(k, l)}$$

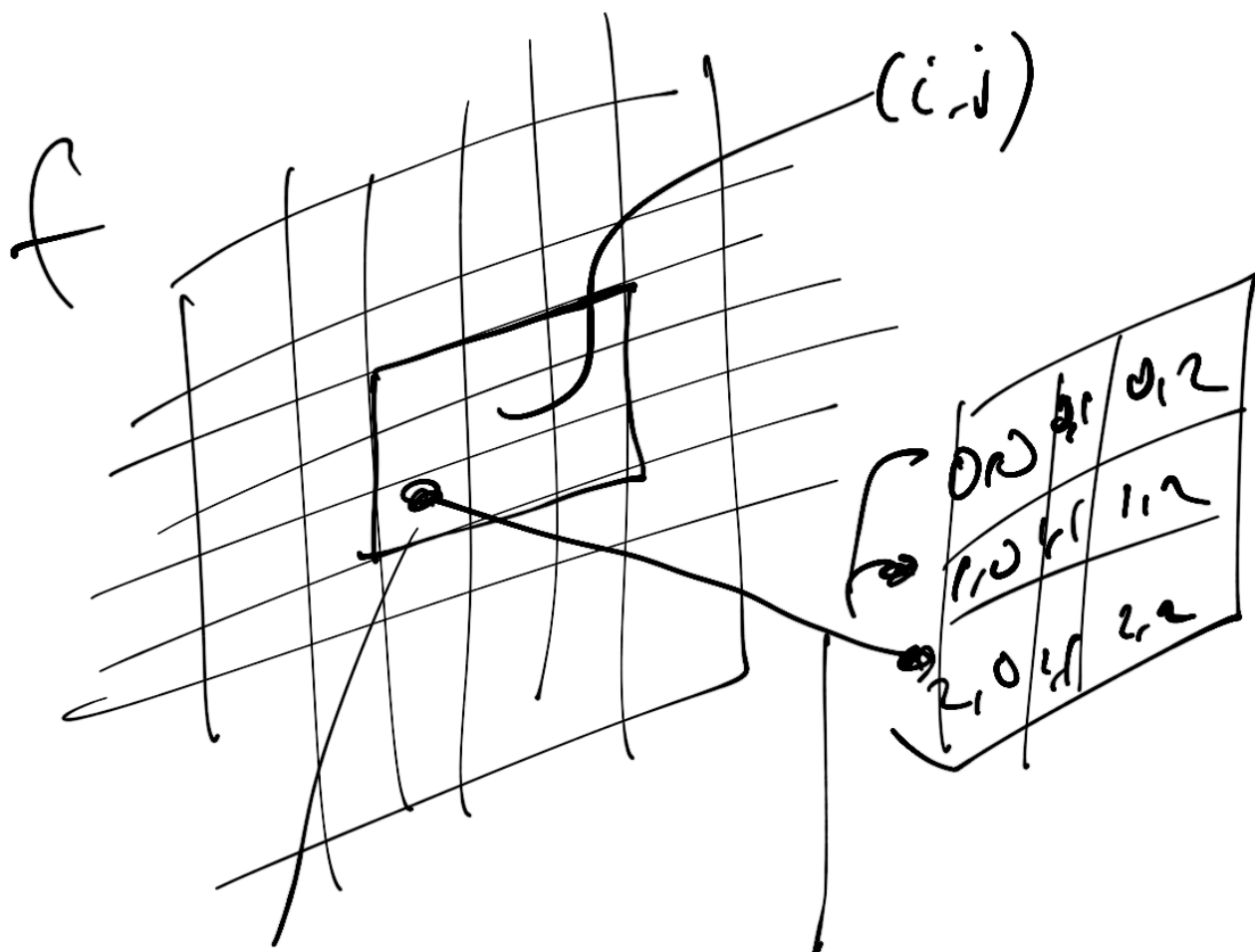
(k, l)

$k \in [0, 2], l \in [0, 2]$

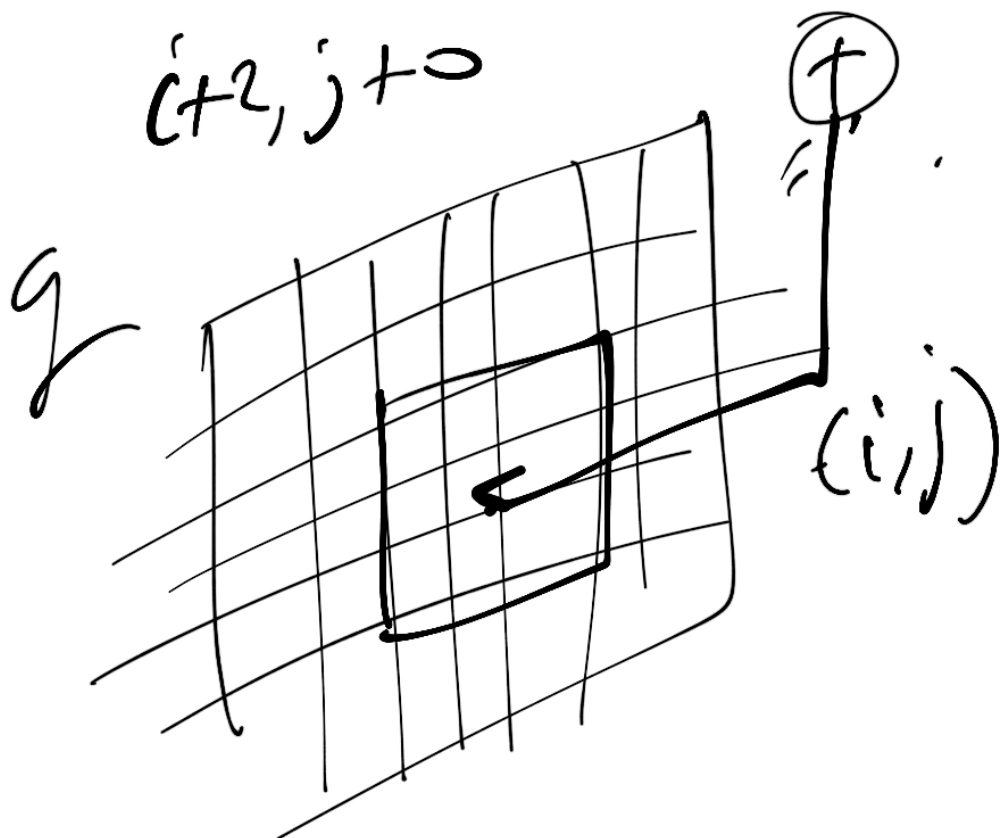
image

operator

offset



$(i+2, j+0)$



$$- \sum_{k,l};$$

$$\text{for } (k=0; k < K; k++) \\ \text{for } (l=0; l < L; l++)$$

$$h(k,l)$$

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

$O(4^2)$ $\left(\begin{array}{l} \text{for } (i=0; i < r; \dots) \\ \text{for } (j=0; j < c; \dots) \end{array} \right)$

$\{ \times \}$
 $O(3^2)$ $\left(\begin{array}{l} \text{for } (k=0; \dots) \\ \text{for } (l=0; \dots) \end{array} \right) \{$

$$g(i, j) = f(i+k, j+l) * h(k, l)$$

"shorthand" ;

$$g = f \otimes h$$

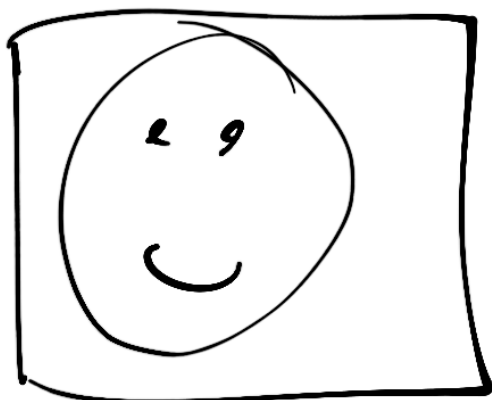
$\left\{ \begin{array}{l} \text{correlation} \\ \text{in spatial} \\ \text{domain} \end{array} \right.$

- $f \otimes h$
correlation

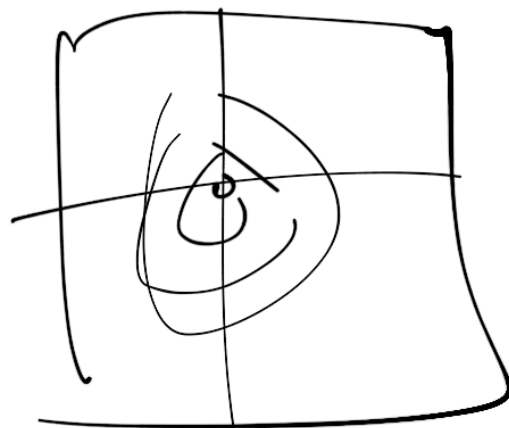
$f \times h$
convolution,
but in Fourier
(frequency)
space

$$f(\vec{x}) = \int f(\vec{x} - \vec{u}) h(\vec{u}) d\vec{u}$$

continuous integral
with $\vec{u}$, V now
depending frequency



$f(i, j)$



$F(u, v)$



same image
in Fourier domain

- $h()$: the filter, generally

linear filters are

shift-invariant

when you

shift or translate

values

does it
change

- can $h()$ be shift-variant?

Yes, e.g. median filter

- as filter shifts along image,
it can extend beyond image
boundary

- you can:

- zeros, padding borders
with 0
- wrap, wrap-around indices
- clamp - repeat last col,
row
- mirror

- what do filters look like?

- box filter

1	1	1
1	1	1

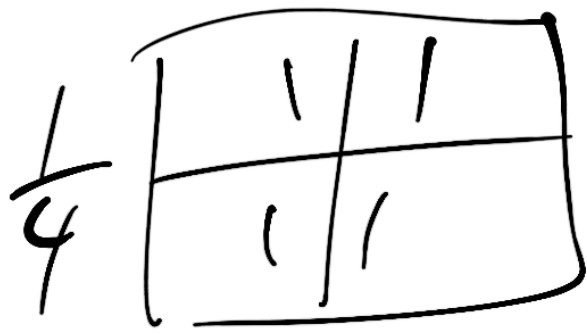
$\Rightarrow$

$\frac{1}{4}$	$\frac{1}{4}$
$\frac{1}{4}$	$\frac{1}{4}$

simple
average

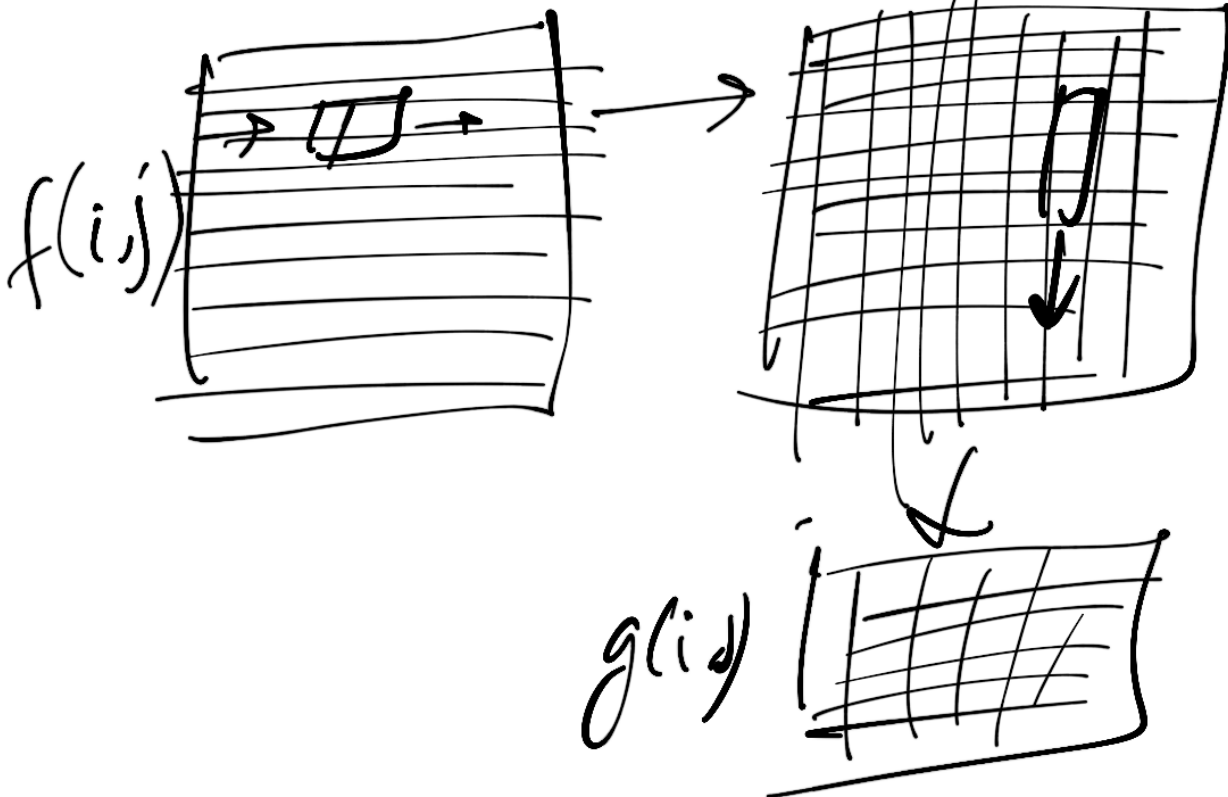
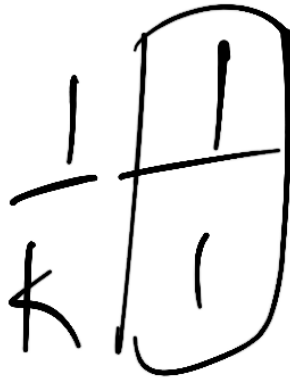
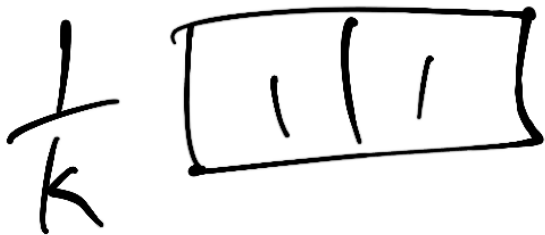
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

(i, j)



$\Rightarrow$ double-nested for loop

Separable:



- bilinear:

$$\frac{1}{16} \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

Separable:

$$\frac{1}{4} \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array}$$

$$\frac{1}{4} \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array}$$


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

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

outer product:

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

Jan. 16

- avg ϕ L:

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

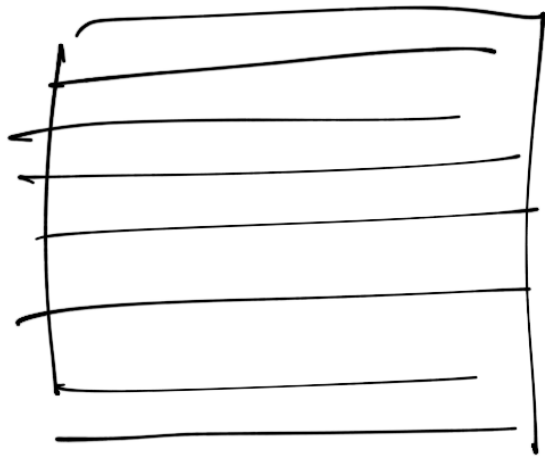
$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

$$G = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y|$$

$$G_x^2(i, j)$$

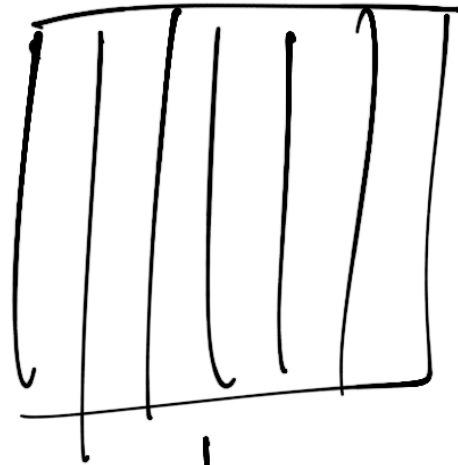
$$\theta = \tan^{-1} \left(\frac{G_y}{G_x} \right)$$

} in C:
use $\text{atan2}()$

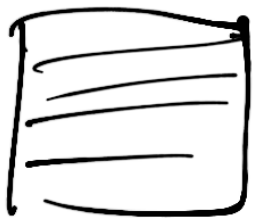


$$\rightarrow (1 \ 0 \ -1)$$

derivative
along x



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



$$\rightarrow (1 \ 2 \ 1)$$



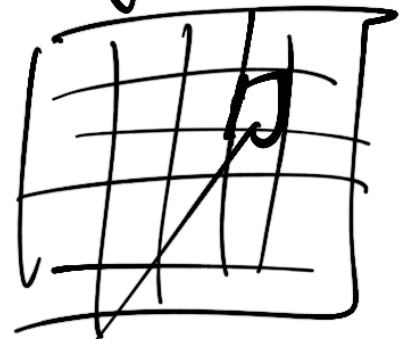
$$\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

α



ϵ_y

α



6π

$$6 = |6x| + |6y|$$

$$\theta = \tan^{-1}$$

Q: { derivative } $80^\circ \rightarrow$

N
S
E
W

84

211

$$\frac{\sqrt{x} - 150}{\pi}$$

Section 3.4 (Szeliski) Fourier

- I start here, but Glasner
(Digital Image synthesis)

- Often intro goes like this:

- given the function $h(k, l)$,
or $h(x, y)$ i.e. Schel 4)

- its Fourier Transform is: filter

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

$\uparrow$
omega, or frequency ω_x, ω_y

- discrete form:

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

- that's usually where I get lost

- what's the $e^{-j(\omega_x x + \omega_y y)}$ mean?

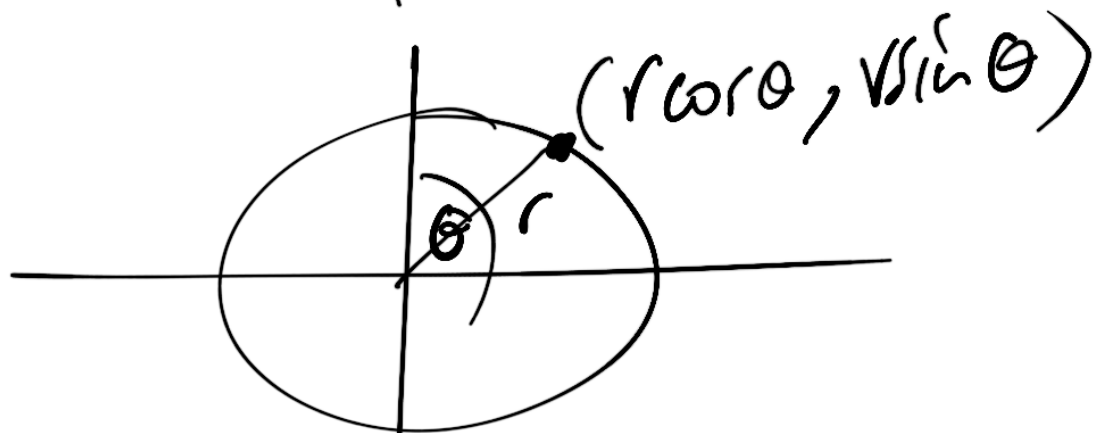
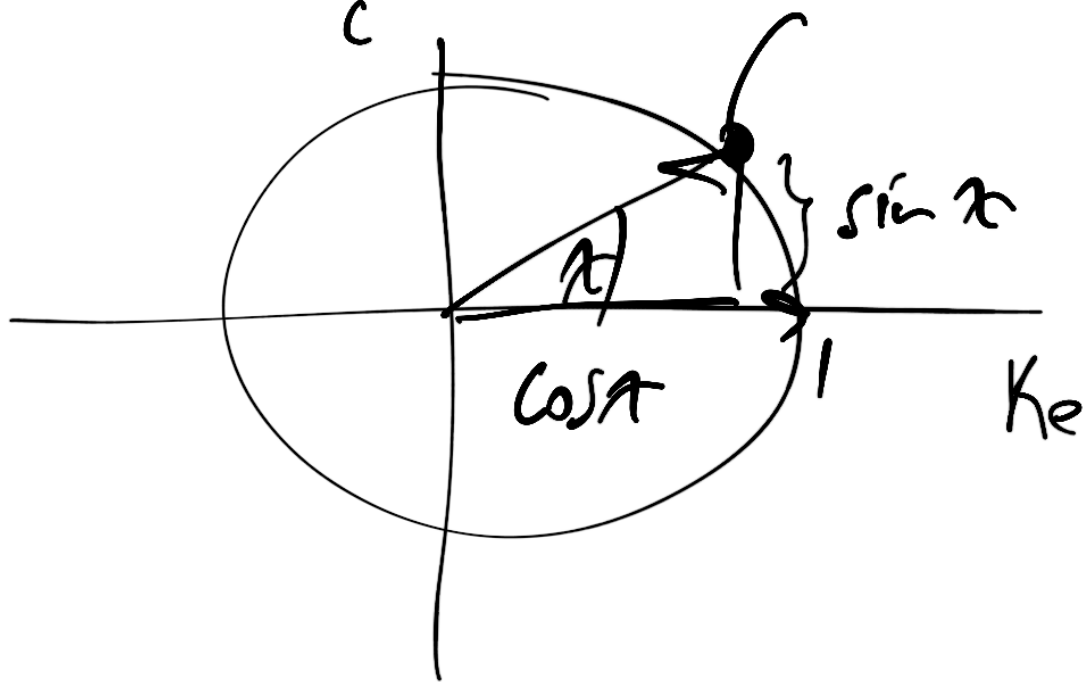
- rewrite with more familiar

$$e^{-j(\omega_x x + \omega_y y)} \quad (i \text{ vs } j)$$

OK Euler's formula:

$$e^{ix} = \cos \omega_x x + i \sin \omega_y y$$

- what it means: $e^{ix} = \cos x + i \sin x$



- analogous to polar coordinates.
just a different coordinate
system instead of Euclidean
+ x

- Fourier Transform (FT)
takes a signal (e.g. 2D image)

& represents it in frequency space

- the FT is a mathematical operator that decomposes a signal (image) into a sum of weighted sines & cosines

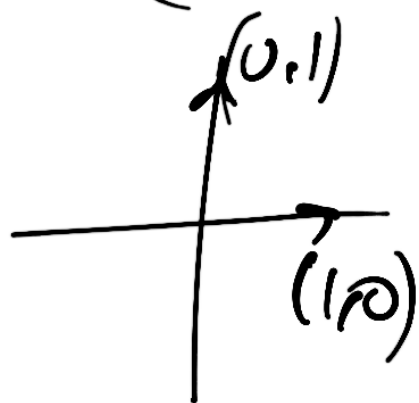
- The $\sin$ & $\cos$ form
the basis of frequency space
→ they are the basis functions

- Any signal can be
represented as combination
of basis functions

- Analogy is our familiar
Euclidean space

- In Euclidean space, need
a reference frame : origin
and orthogonal vectors

($\perp$ perpendicular)



$$V_x = [1 \ 0]$$

$$V_y = [0 \ 1]$$

- normalized vectors (length is 1)

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

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

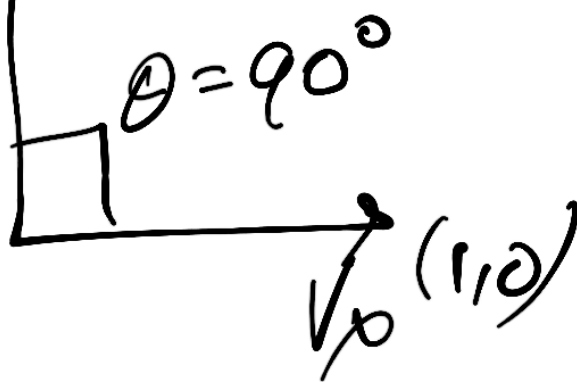
(we have orthonormal basis)

- test for orthogonality?

dot product

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

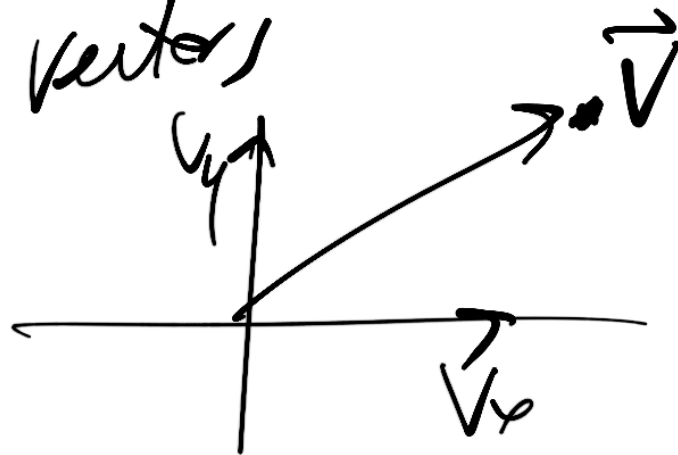
$v_y(1,0)$



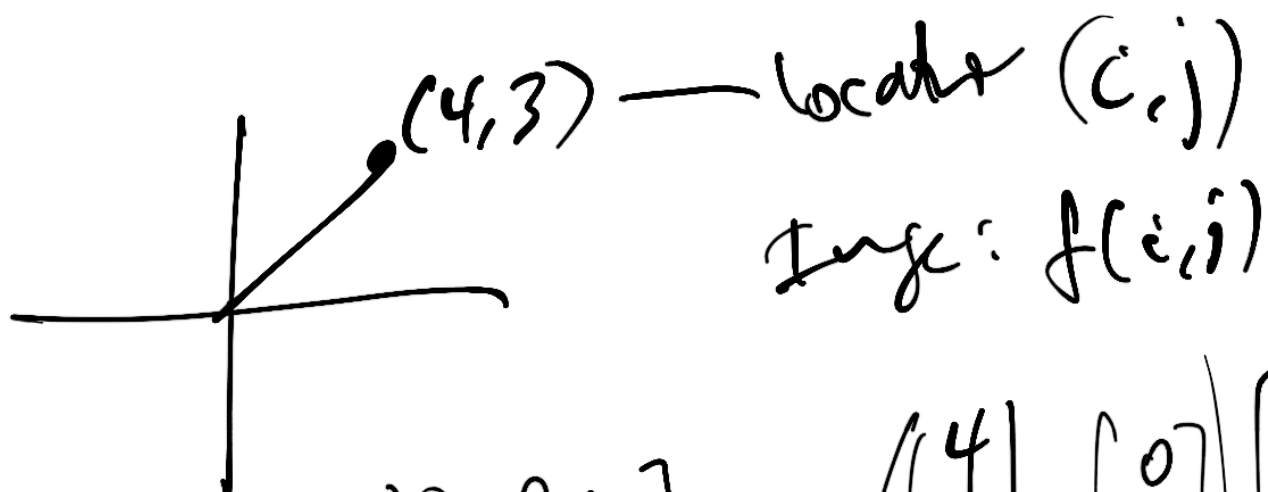
$$\frac{v_x \cdot v_y}{\|v_x\| \|v_y\|} = \cos(\theta) = \cos(90) = 0$$

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

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



$$\vec{V} = [x \ y] = \underbrace{(\vec{V} \cdot \vec{V}_x)}_{\text{dot product (a scalar)}} \vec{V}_x + \underbrace{(\vec{V} \cdot \vec{V}_y)}_{\text{a scalar}} \vec{V}_y$$



$$\left(\begin{pmatrix} 4 \\ 3 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right) \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \left(\begin{pmatrix} 4 \\ 3 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right) \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

v_0 v_1

$$\left((4)(1) + (3)(0) \right) \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \left((4)(0) + (3)(1) \right) \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

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

$$= \begin{pmatrix} 4 \\ 3 \end{pmatrix}$$

- dot product: $\vec{v} \cdot \vec{w} =$

$$\sum_{i=1}^n v_i w_i = \cos(\theta)$$

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

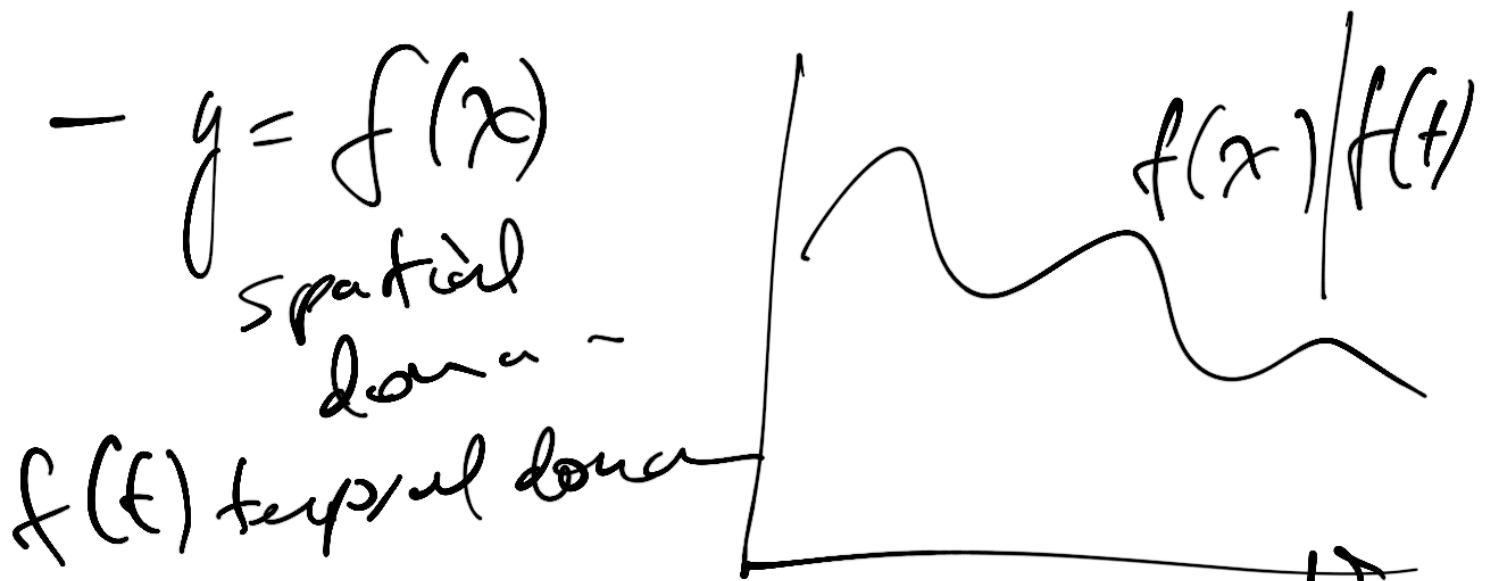
ex. $[1 \ 3 \ -5] \begin{pmatrix} 4 \\ -2 \\ -1 \end{pmatrix} = 4 - 6 + 5$
 $= 3$

- Key observation is that basis vectors were scaled

$$V = \underbrace{(V \cdot V_x)}_{\text{scalar}} V_x + \underbrace{(V \cdot V_y)}_{\text{scalar}} V_y$$

projection of V onto space defined by basis functions

- In general, for FA, discussion starts with 1D function $y = f(x)$



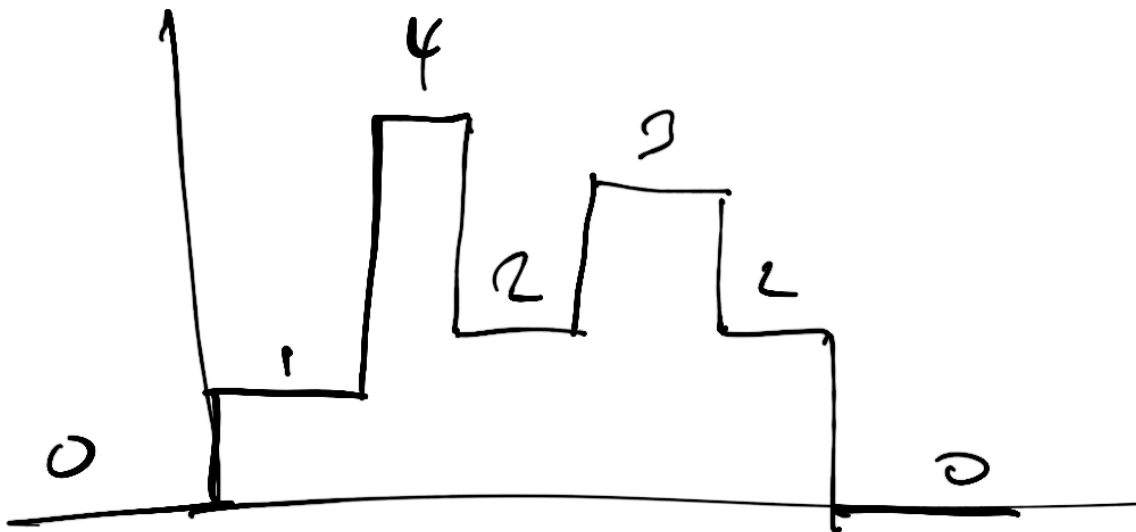
can we project onto
basis functions $\phi_i(t)$

Yes,

$$f(t) = c_1 \phi_1(t) + c_2 \phi_2(t) + \dots + c_n \phi_n(t)$$

linear combination of weighted
basis functions

- example: "bar chart" function



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

$$t: 1 \ 2 \ 3 \ 4 \ 5$$

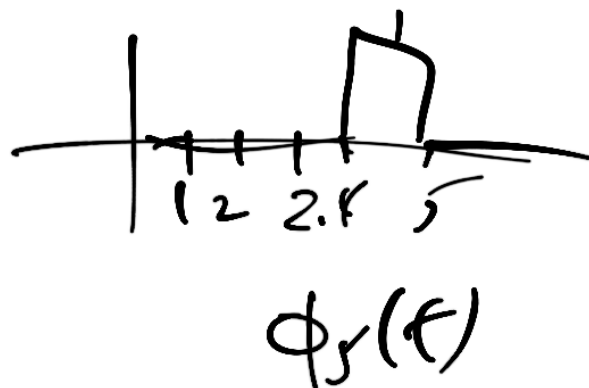
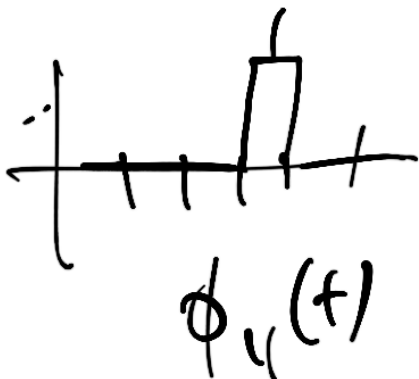
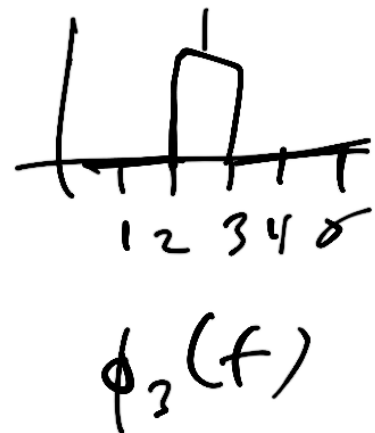
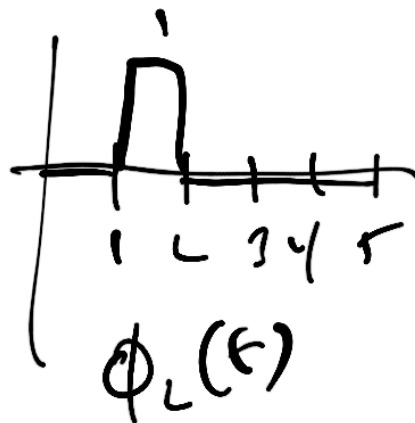
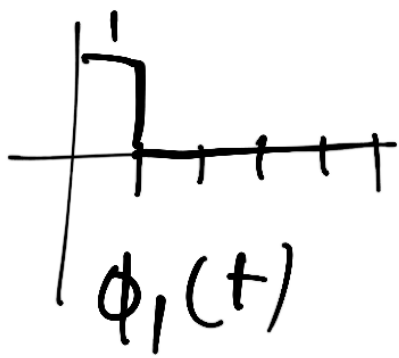
$$f(t) \in [1, 5]$$

$$f(t) \in \begin{cases} f(t), & 1 < t < 5 \\ 0, & \text{elsewhere} \end{cases}$$

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

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

$\uparrow$
 i



- bar chart $p(t)$

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

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

coefficients basis function(s)

- normally, we'd just use
basis function for this example

and shift to the right by 1 (by t)



- properties of basis functions:

- for 5D, need 5
- for 3D, need 3
etc.

} how many?
one
for each
dim.

- functions (or vectors)
need to be mutually
orthogonal to span
the space

} completeness

- In general

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

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

like the generalized dot product

$$\text{if } i=j, \quad \int_{t_1}^{t_2} \overline{\phi_i(t)} \phi_i(t) dt \neq 0$$

- for vectors, $i \neq j$

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

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

$$= 0 + 0 = 0$$

(no overlap)

$$i = j$$

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

- system is normalized if

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

orthonormal system

- one can normalize by dividing
by the norm or length =

$$\langle \phi_i, \phi_i \rangle = \|\phi_i\|^2$$

- if $\|\phi_i\| = 1$, then system is
normalized

- Now, recall,

$$f(t) = c_1 \phi_1(t) + c_2 \phi_2(t) + \dots + c_n \phi_n(t)$$

$$= \sum_i c_i \phi_i(t)$$

- key thing: how to find c_i
for a given basis $\in f(t)$?

wavelet coefficients

- a math digression -

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

is just an approximation, 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 the interval $[t_1, t_2]$

- rewrite in long form:

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

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

$$(a-b)(a-b) = a^2 - 2ab + b^2$$

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

where

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

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

- the $c_1^2 k_1 + c_2^2 k_2 + \dots + c_n^2 k_n -$

$$2c_1 \gamma_1 - c_2 \gamma_2 - \dots - 2c_n \gamma_n$$

can be written as

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

to complete the square

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

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

- plug back in:

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

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

for each i

so the squared error goes to 0

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

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

- in the end, we basically

convolve $f(t)$ with basis functions $\phi_i(t)$

- Basically like or

$$(V \cdot v_x) v_x + (V \cdot v_y) v_y$$

- In other words, we just
project $f(t)$ onto space
defined by $\phi_i(t)$

- basically a change of
coordinate space for

pixels
(space, time) onto frequencies
spatial domain (frequency space)

- The FT uses $\psi_u(t) = e^{iut}$

$$u \in \mathbb{Z}$$

$\leftarrow$ integers

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

- wavelet transform projects

onto ϕ and ψ

$\nearrow$
scaling
function

$\uparrow$
wavelet
function

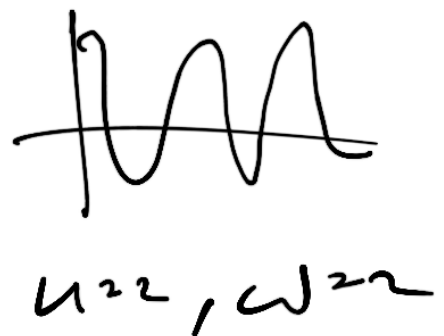
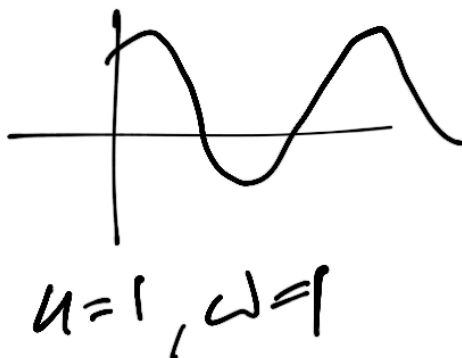
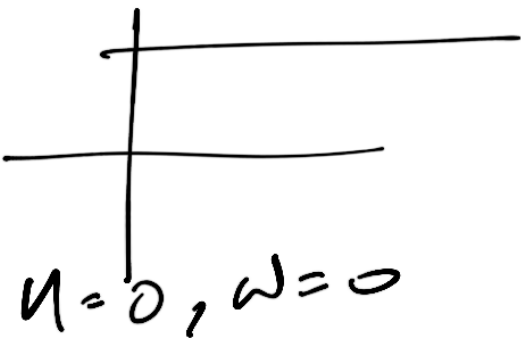
- Fourier series)

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

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

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

omega, frequency



- Fourier Transform:

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

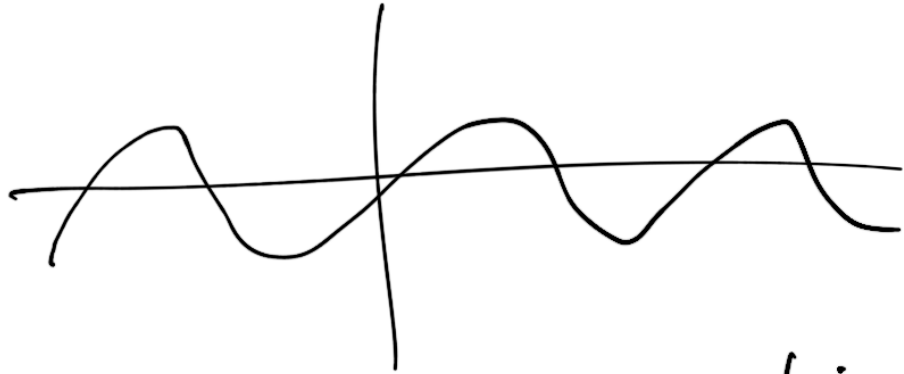
$$x(t) = \frac{1}{\sqrt{2\pi}} \int X(\omega) e^{i\omega t} d\omega = \frac{1}{\sqrt{2\pi}} \langle \bar{x}, \psi \rangle$$

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

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

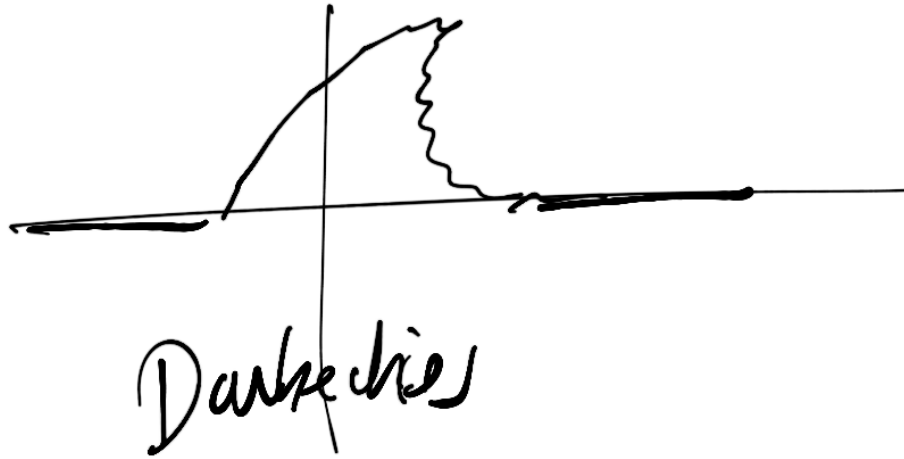
$$x(t) \xleftrightarrow{F} X(\omega)$$

- key thing: Fourier basis functions have infinite support: they exist everywhere in the signal domain



- FT integrates over entire signal
if there's a high freq. burst somewhere
FT won't indicate where it is

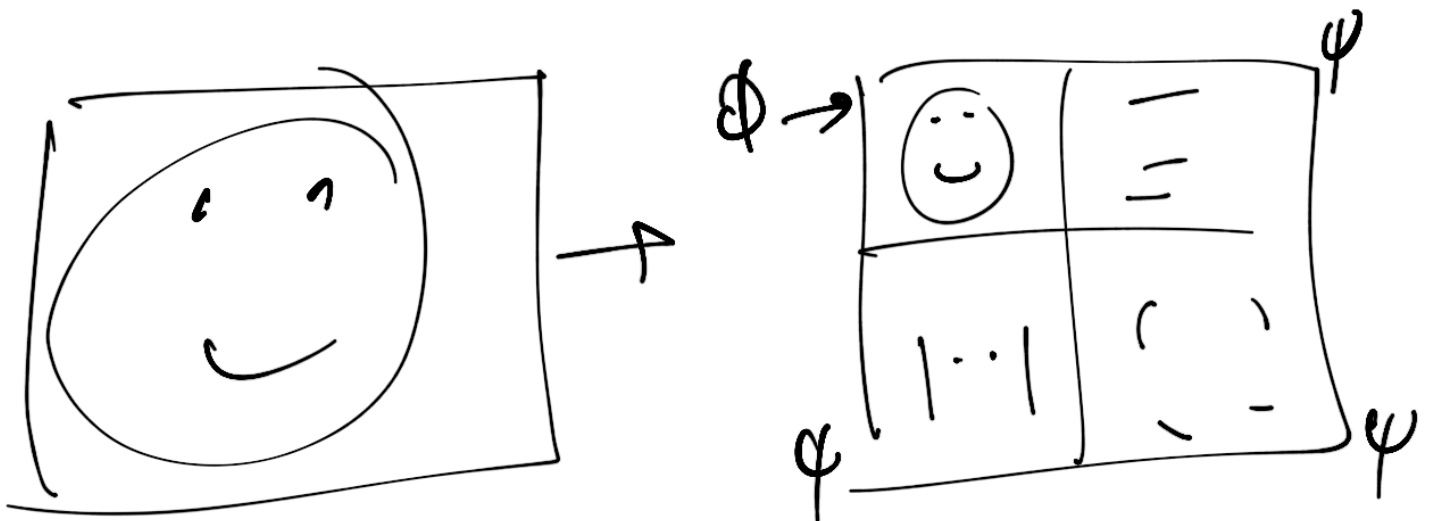
- wavelet basis function :
Compact support



- Wavelets:

- 2D family of function
derived from scaling
function ϕ

(this is the one that smooths
a values in resolution signal)



- a mother wavelet is derived,
which is then

scaled, dilated, shifted
(multiplication) (power) (translation)

$$2^j \psi(2^j t - k)$$

$$2^j \psi(2^j t - k)$$

$$j \in \mathbb{Z} \quad 0, 1, 2, \dots, \infty$$

in code, set $j=1$, then $j = j \ll 1$

	0001	1
$\ll 1$	0010	2
$\ll 2$	0100	4

- wavelets:

- starts with scaling function

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

coefficients

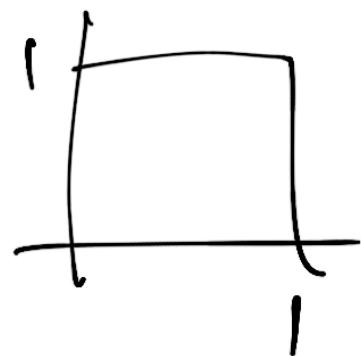
dilation

shift

- dilation is often applied recursively:

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

example: unit square



$$S(x, y, lx, ly, s) =$$

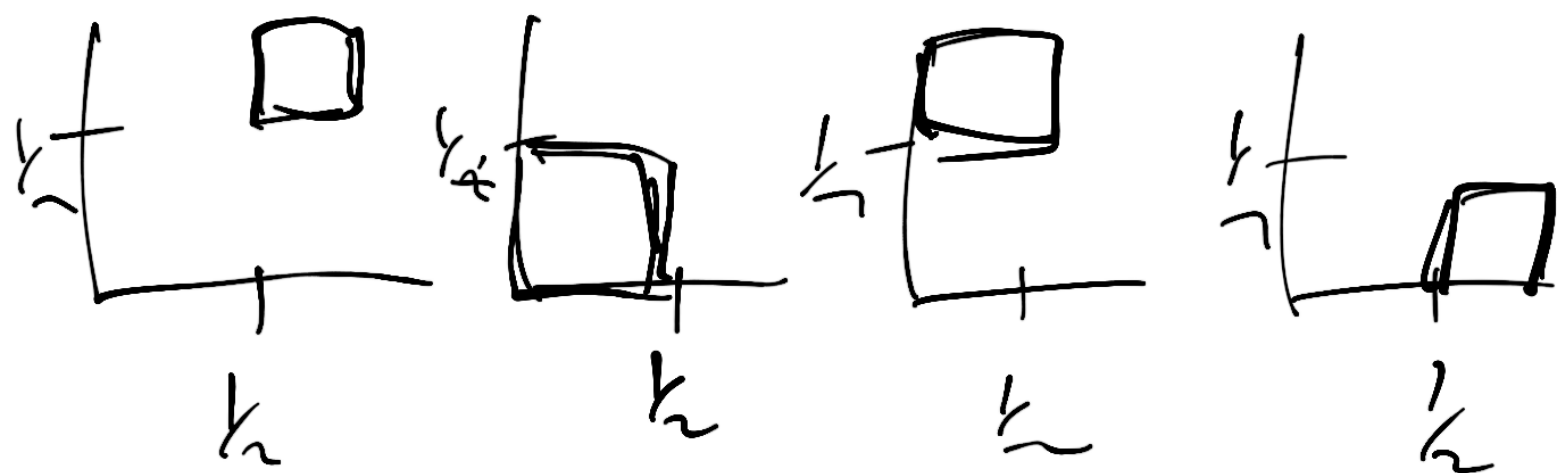
$$\begin{cases} 1, & lx \leq x < lx+s \text{ and} \\ & ly \leq y \leq ly+s \\ 0, & \text{otherwise} \end{cases}$$

$$S(x, y, lx, ly, s) = S(x, y, x - \frac{s}{2}, y - \frac{s}{2}, s) +$$

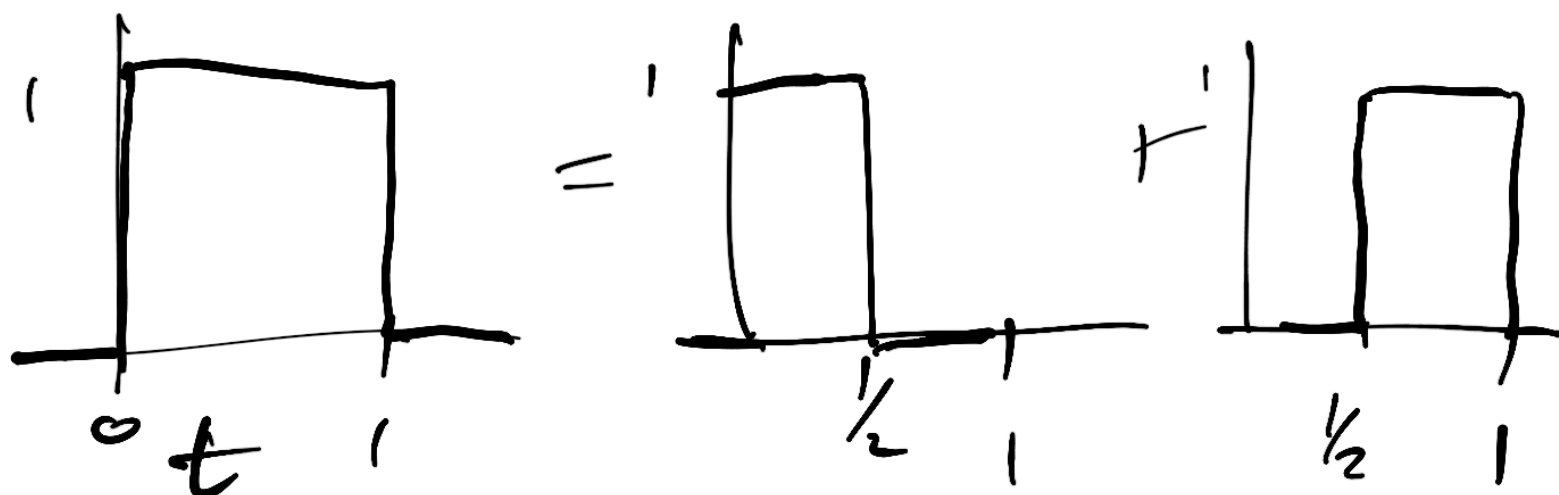
$$S(x, y, x + \frac{s}{2}, y - \frac{s}{2}, s) +$$

$$S(x, y, x - \frac{s}{2}, y + \frac{s}{2}, s) +$$

$$S(x, y, x + \frac{s}{2}, y + \frac{s}{2}, s)$$

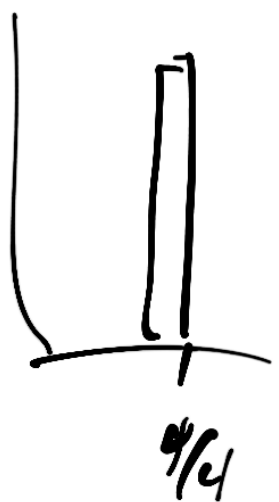
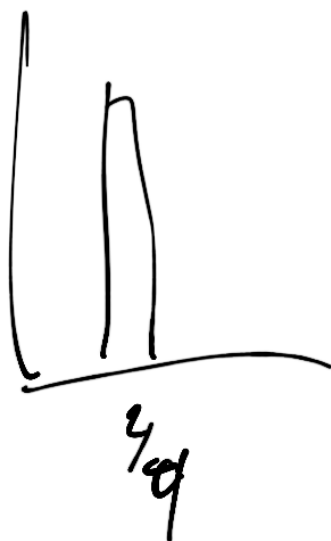
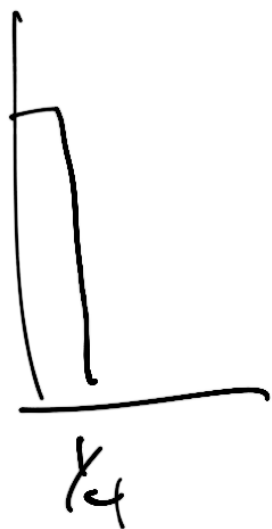


- box function:

$$y_0(t) = \begin{cases} 1, & 0 \leq t \leq 1 \\ 0, & \text{otherwise} \end{cases}$$


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

$$g_0(t) = g_0(2t) + \gamma_0(2t-1) + g_0(2t-2) + \gamma_0(2t-3)$$

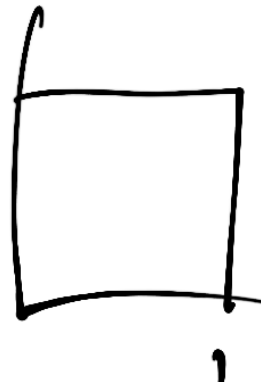


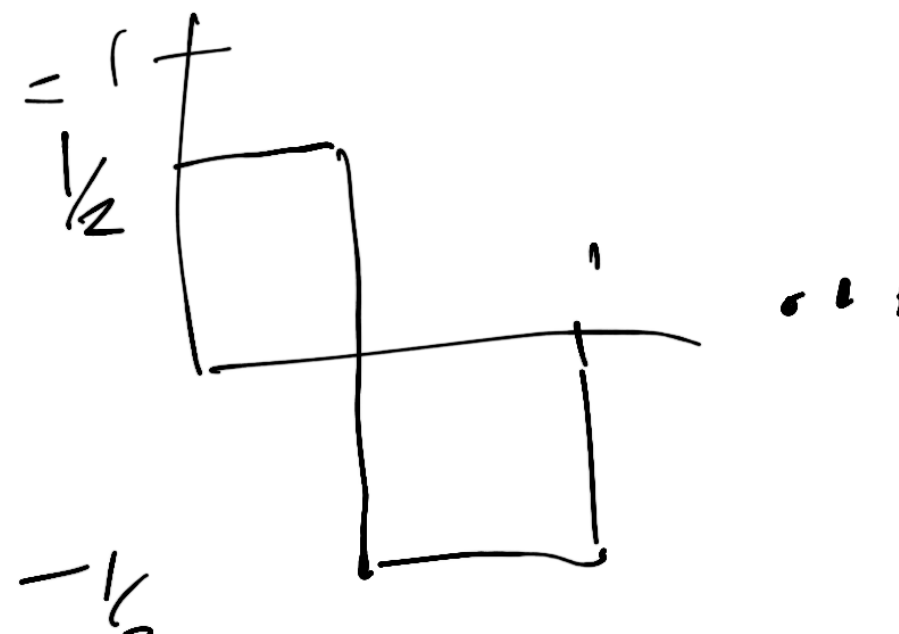
$$\Phi_j(x) = \sum_k c_k \Phi_{j-1}(2x-k)$$

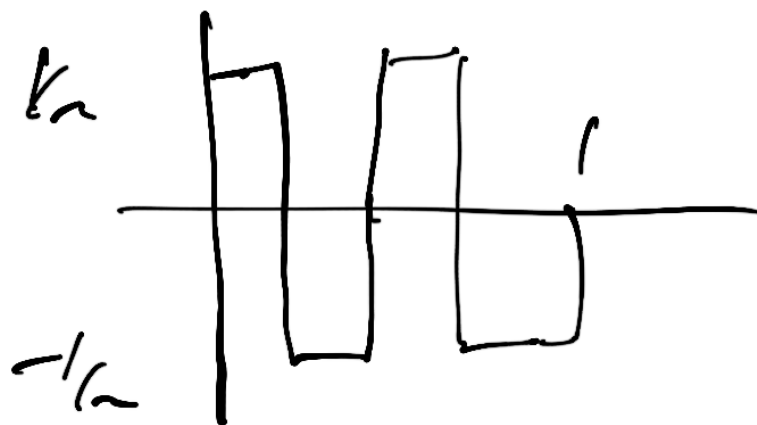
recipe for wavelet function ψ :

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

$\sum_{\text{some text}} c_{1-k}$

- with ζx : $\phi(x) =$ 

$\psi(x) =$ 



dilation also works for $\psi(x)$

and: $\langle \phi, \psi \rangle = 0$ they are
orthogonal
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 two mean dyadic
dilations (the $\ll 1$ shift
operator is well)

- how does this work?

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

where $\swarrow$ resolution level, not "power"

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

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

$\uparrow$
detail (wavelet) coefficient

- recursively

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

- "1-level" decomposition

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

"2-level":

$$\begin{aligned} f^2(x) &= g^1(x) + \underbrace{f^1(x)} \\ &= g^1(x) + g^0(x) + f^0(x) \end{aligned}$$

$$\vdots$$

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

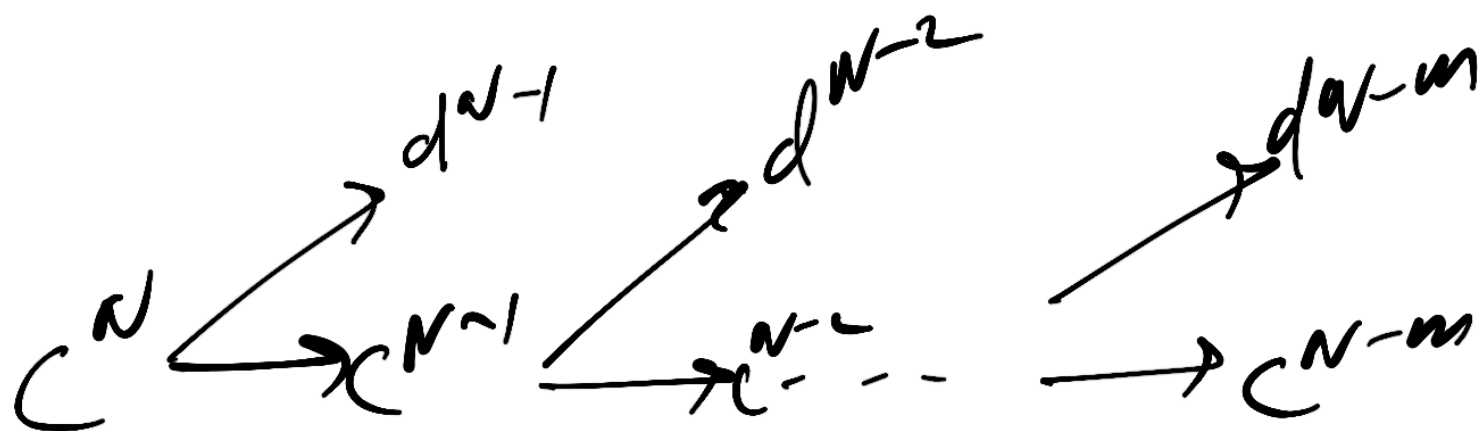
$$f^{j+1}(x) = g^j(x) + g^j(x)$$

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

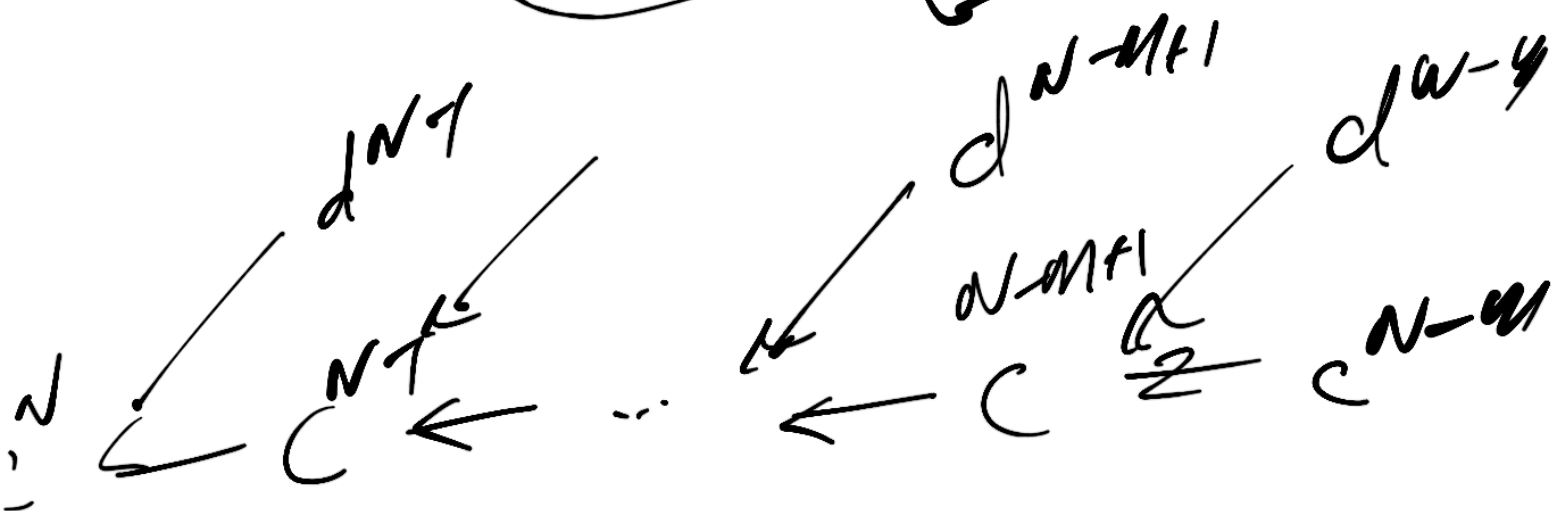
$$\sum c_{j,k} \phi_{j,k}(x),$$

$$j,k \in \mathbb{Z}$$

- schematically,



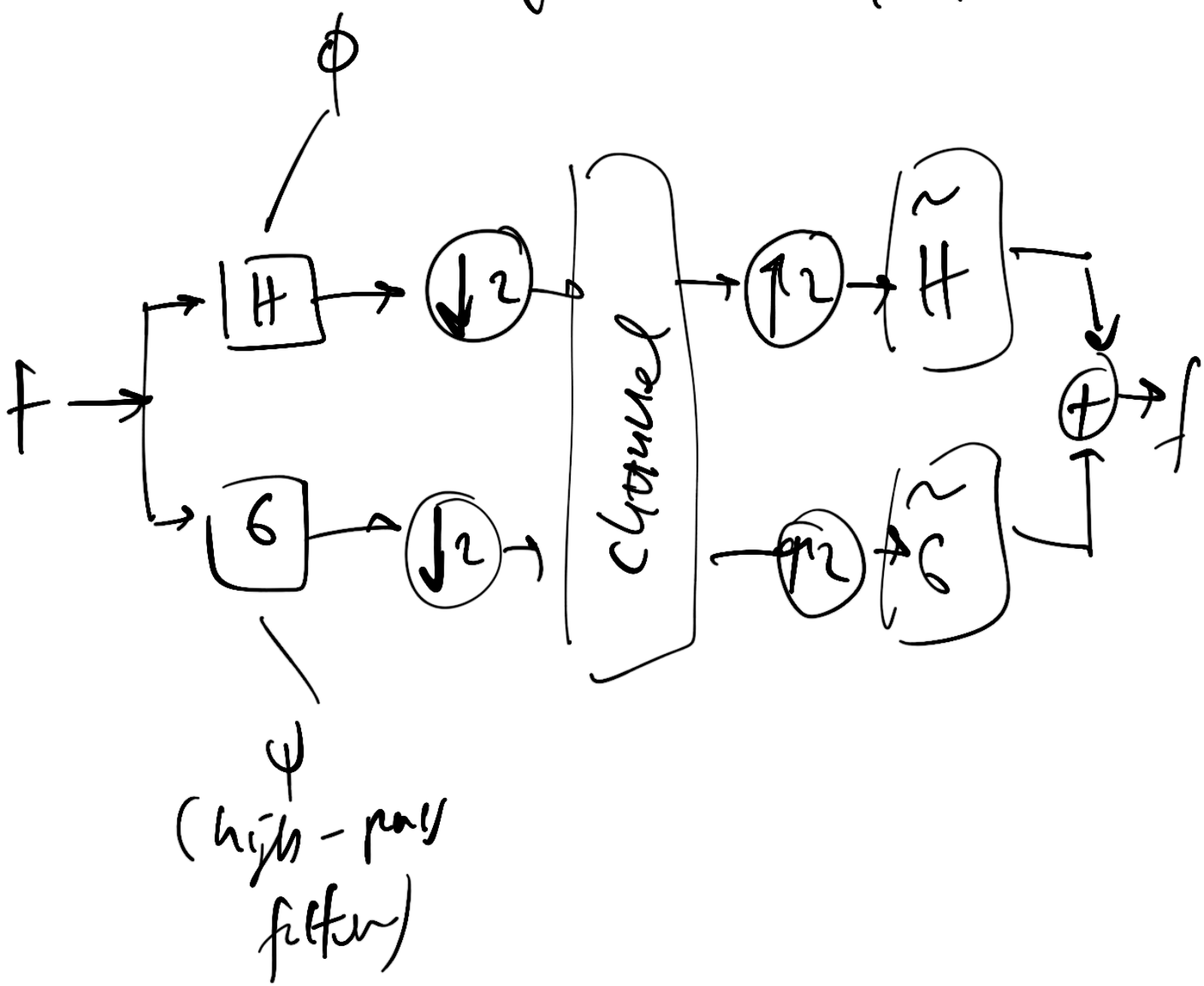
decomposition
unstructured



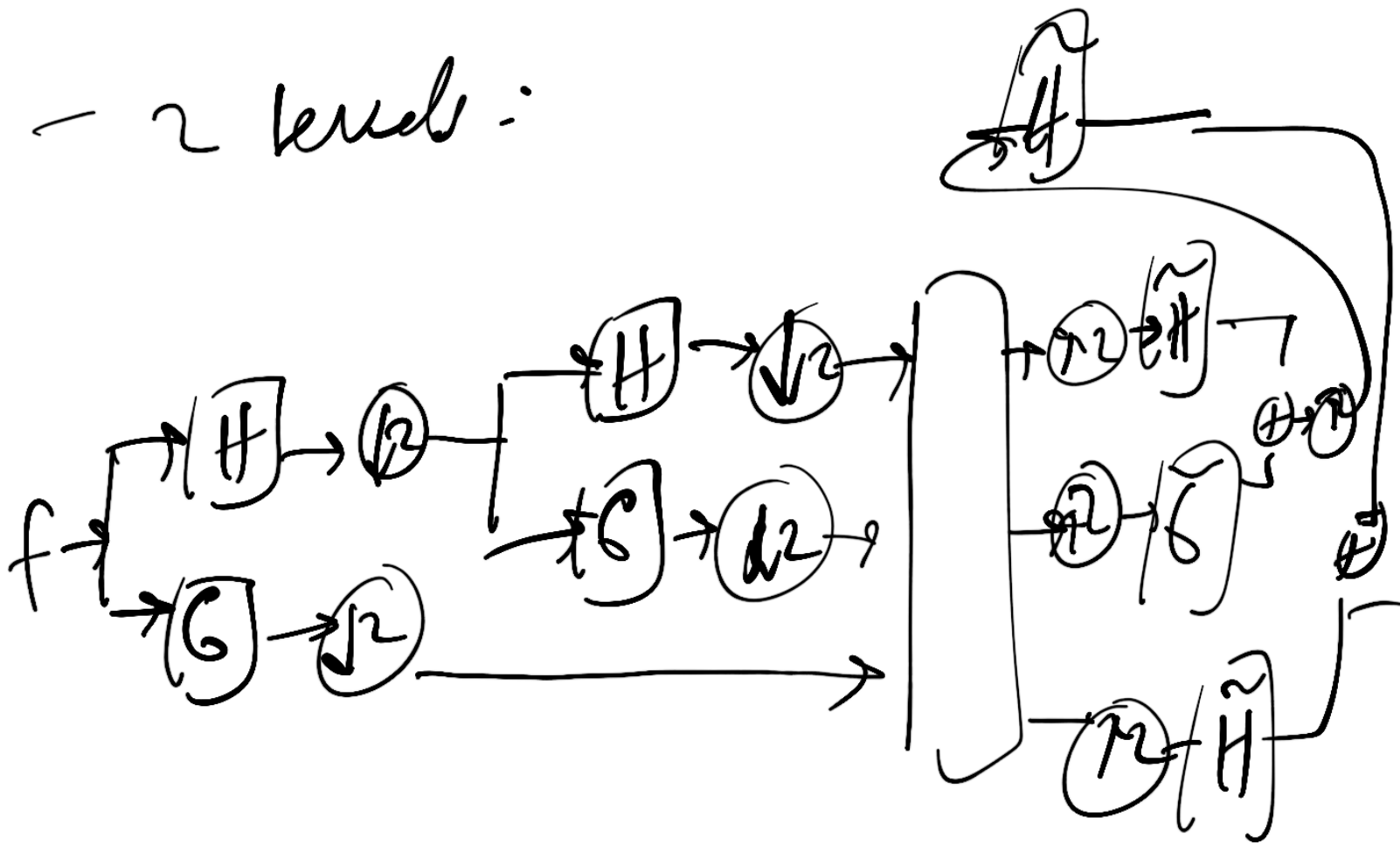
- or

(low-pass filter)

1-level



- 2 kinds:



H : low-pass filter, (ϕ)
 e.g.; Haar:

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

G : high-pass filter (Haar)

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

(ψ)

$$g_k = (-1)^k h_{1-k} \quad \left. \vphantom{g_k} \right\} \text{quadrature mirror}$$

e.g., some arbitrary random
~~number~~: filter h_k :

-1	2	5	7	4	12	-4
----	---	---	---	---	----	----

g_k :

-4	-12	4	7	5	-2	-1
----	-----	---	---	---	----	----

$$h_k: \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$g_k: \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

- mathematically, decomposition at resolution level j is:

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

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

still the low-pass
image at previous
level

h_k, g_k are the 1D filters.

- construction is a bit trickier:

$$f_{\phi \boxtimes \psi}^{j-1}(2x+p) = (1-p)f_{\phi}^{j-1}(x) + (p)f_{\psi}^{j-1}(x)$$

↑
interleave

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

- above is a pre-processing step

- the construction is then:

$$f_{\phi}^j(2x+p) = (1-p) \sum_k \tilde{h}_k f_{\phi \boxtimes \psi}^{j-1}(x-k) +$$

$$(p) \sum_k \tilde{g}_k f_{\phi \boxtimes \psi}^{j-1}(x-k)$$

just a convenient way of writing:

$$f_{\phi}^j(2\pi) = \sum_k \tilde{h}_k f_{\phi \otimes \psi}^{j-1}(\pi - k)$$

$$f_{\phi}^j(2\pi + 1) = \sum_k \tilde{g}_k f_{\phi \otimes \psi}^{j-1}(\pi - k)$$

$$h_k: \quad \frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}}$$

$$k=0 \quad k=1$$

$$\tilde{h}_k: \quad \frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}}$$

$$k=0 \quad k=1$$

$$g_k: \quad \frac{1}{\sqrt{2}} \quad -\frac{1}{\sqrt{2}}$$

$$\hat{g}_k: \quad \frac{1}{\sqrt{2}} \quad -\frac{1}{\sqrt{2}}$$

- a numerical example: decomp:

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

$$d^1: \quad \begin{array}{cc} \swarrow & \searrow \\ \frac{1}{\sqrt{2}}(1) - \frac{1}{\sqrt{2}}(4) & \frac{1}{\sqrt{2}}(0) + \frac{1}{\sqrt{2}}(2) \end{array}$$

$$\begin{array}{cc} -\frac{3}{\sqrt{2}} & \frac{2}{\sqrt{2}} \end{array} \quad (g_k)$$

$$c^1: \quad \begin{array}{cc} \frac{1}{\sqrt{2}}(1) + \frac{1}{\sqrt{2}}(4) & \frac{1}{\sqrt{2}}(0) + \frac{1}{\sqrt{2}}(-2) \end{array} \quad (h_k)$$

$$\boxed{\begin{array}{cc} \frac{5}{\sqrt{2}} & -\frac{2}{\sqrt{2}} \end{array}}$$

$$d^0: \quad \frac{1}{\sqrt{2}}\left(\frac{5}{\sqrt{2}}\right) + \frac{2}{\sqrt{2}\sqrt{2}} = \frac{7}{2}$$

$$c^0: \quad \frac{3}{2}$$

$$f^2 (= c^2) = \begin{bmatrix} 1 & 4 & 0 & -2 \end{bmatrix}$$

$$w(f) = \begin{bmatrix} \frac{3}{2} & \frac{7}{2} & -3 & \frac{2}{\sqrt{2}} \end{bmatrix}$$

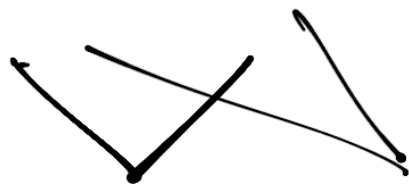
$c_0 \quad d_0 \quad d_1 \quad d_1$

restarted to get back

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

- Transformation:

$$c^0 \quad d^0: \quad \frac{3}{2} \quad \frac{7}{2}$$



$c^1:$

$$\tilde{u}_k \quad \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tilde{g}_k$$

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

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

$$\begin{pmatrix} \frac{5}{\sqrt{2}} & \frac{-2}{\sqrt{2}} \end{pmatrix}$$

$$c^1 \quad d^1: \quad \begin{pmatrix} \frac{5}{\sqrt{2}} & \frac{-3}{\sqrt{2}} & \frac{-2}{\sqrt{2}} & \frac{2}{\sqrt{2}} \end{pmatrix}$$

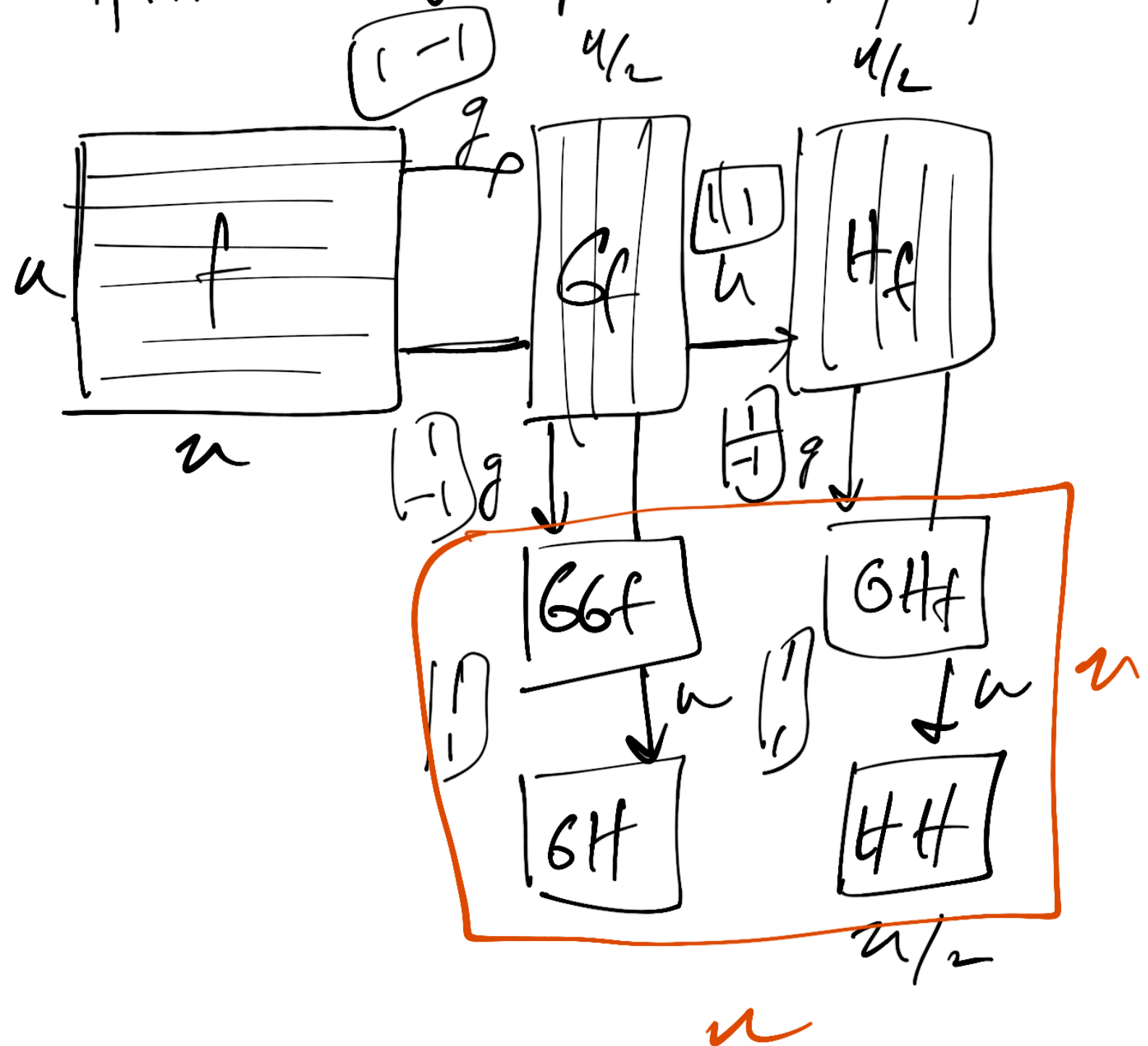
$c_1 \quad d_1 \quad c_1 \quad d_1$

$$C^1_{pd}: \quad \frac{5}{\sqrt{2}} \quad \frac{-3}{\sqrt{2}} \quad \frac{-2}{\sqrt{2}} \quad \frac{2}{\sqrt{2}}$$

$$\sim h_k: \quad \begin{array}{cc} \begin{array}{c} \text{Diagram 1: Triangle with vertices at } \frac{5}{\sqrt{2}}, \frac{-3}{\sqrt{2}}, \text{ and } \frac{-2}{\sqrt{2}} \end{array} & \begin{array}{c} \text{Diagram 2: Triangle with vertices at } \frac{-2}{\sqrt{2}}, \frac{2}{\sqrt{2}}, \text{ and } \frac{-2}{\sqrt{2}} \end{array} \\ \frac{1}{\sqrt{2}} \left(\frac{5-3}{\sqrt{2}} \right) & \frac{1}{\sqrt{2}} \left(\frac{-2+2}{\sqrt{2}} \right) \\ \frac{2}{2} & \frac{0}{2} \end{array}$$

$$C^2: \quad \begin{array}{|c|c|c|c|} \hline 1 & 4 & 0 & -2 \\ \hline \end{array}$$

- 2D implementation:
- filter using separable h, g filters



- What is the effect?

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

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

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

$$H_C: \phi \otimes \psi^T: \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} & \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} \end{pmatrix}$$

$$G_C = \psi \otimes \psi^T = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Ch. 4

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

pgm_dwt2D (gimg, levels)
// visualization

nimg = pgm_copy(gimg);

pgm_normalize (nimg):

// output nimg

// smoothing interval

pgm_idwt2D (gimg, levels)

inverse transform

If note: you could copy
giving, do the

DWT $\rightarrow$ IDWT $\hat{=}$ compute

the MSE - is there a diff?

Mean-squared Error:

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

— per-pixel squared difference

- pgen_dwt 2D (p64 * iing, int levels)

int $hi = i \log_2(iing \rightarrow rows)$

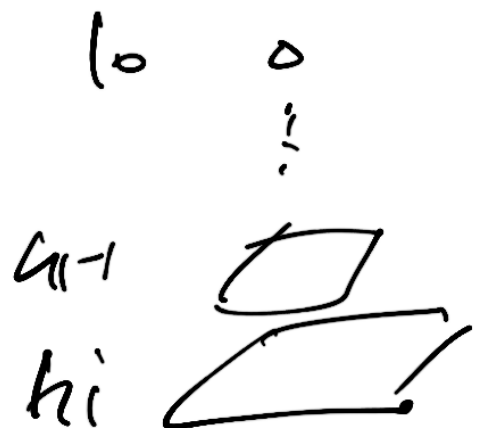
// stupid code:

$(i \& 0x8000) \text{ return } 15;$

else if $(i \& 0x0001) \text{ return } 0;$

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

int lo = hi - levels;



1 0,1

2 ~~4~~ 4
1 4

1

1

1

8



9



10



- / vs int h_i, l_i ; as the levels of the POTaxa image pyramid
- int levels = $\emptyset$; how many levels to do the DWT ($\emptyset$ means do it once)

WVLT xwt = wavelet_Wavelet(HAAR)

dw2d(int $\rightarrow$ data, int $\rightarrow$ rows, int $\rightarrow$ cols, w, h_i, l_i);

make wavelet:

$$\text{wvlt} \rightarrow h[0] = \frac{1}{\sqrt{2}}$$

$$\text{wvlt} \rightarrow h[1] = \frac{1}{\sqrt{2}}$$

$$\text{wvlt} \rightarrow g[0] = \text{wvlt} \rightarrow h[1]$$

$$\text{" } g[1] = -\text{wvlt} \rightarrow h[0]$$

$\text{wvlt} \rightarrow h_t$: same as h

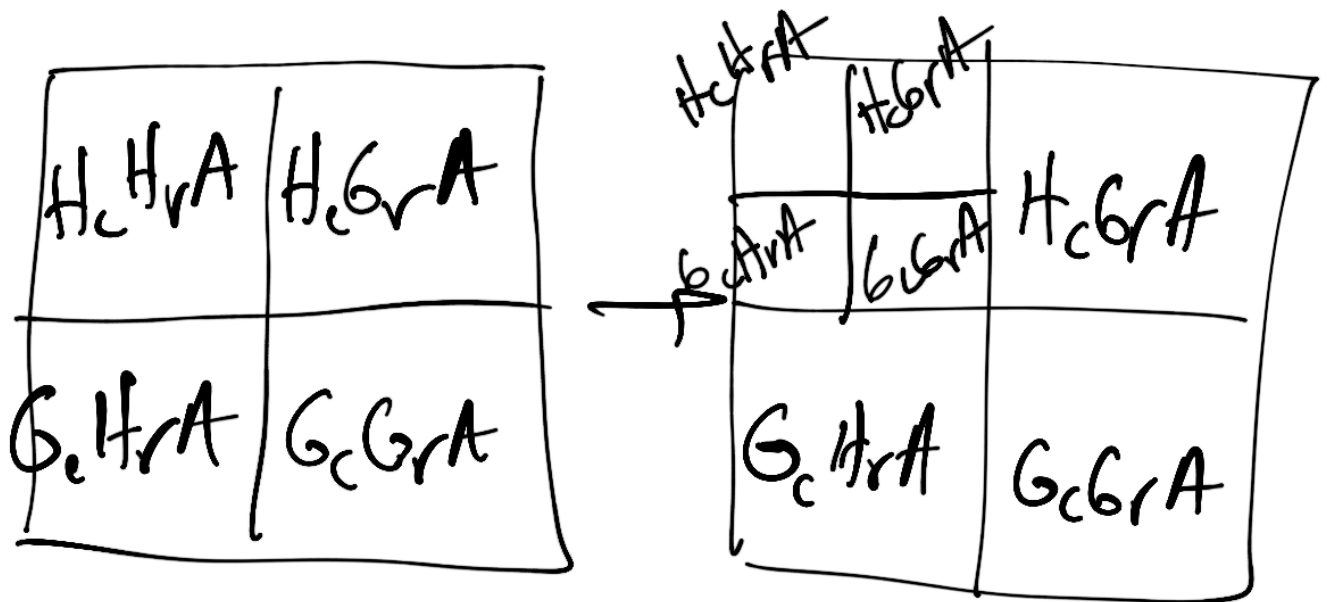
$\text{wvlt} \rightarrow g_t$: same as g

for Haar wavelets only

- `dwt2D()`, `idwt2D()` functions

↓
I'll go over this one

- `dwt2D(double *I, int rows, int cols, wVLT w, fVLT f, cVLT c)`



- $\neq$ v.k double *H_rA, *G_rA
as temp image (pixel) arrays

- also, some helper indices:

int i, j, m, r, c, k

fixed indices filter index row, col val. level

int r_{off}, c_{off}; row, col
 offsets

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

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

$r = \text{rows}; \quad \text{roff} = r >> 1; \quad // \frac{r}{2}$

$c = \text{cols}; \quad \text{coff} = c >> 1;$

for $k = \text{fulvl}; \quad k \geq \text{culvl}; \quad k--$

$r2 = r >> 1;$

$c2 = c >> 1;$

HrA : $[r, c2]$ array, set to $\emptyset$

GrA : $[r, c2]$

pass 1

for $i = \emptyset; i < r; i++$

for $j = \emptyset; j < c2; j++$



for $m = \emptyset; m < \underline{\text{up} \rightarrow \text{len}}; \text{with}$

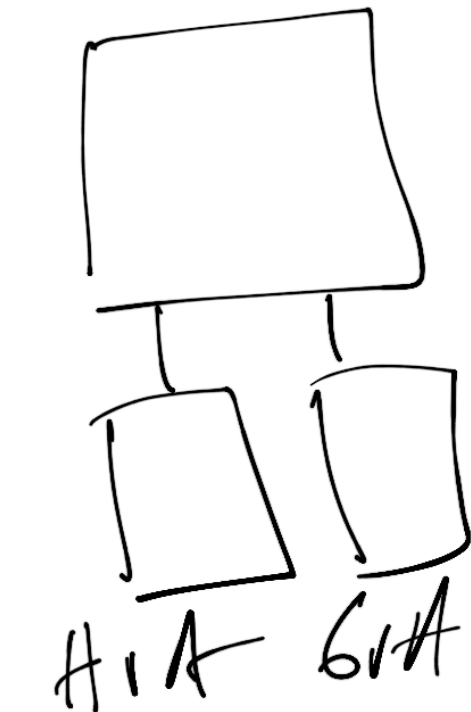
$$j = j < l;$$

$$H_A(i, j) = \sum_m h_m I(i, \underbrace{j+m})$$

same as

$$2j + m$$

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



same height,
half width

pass 2
(still in k loop)

for $i = 0; i < rL; i++$

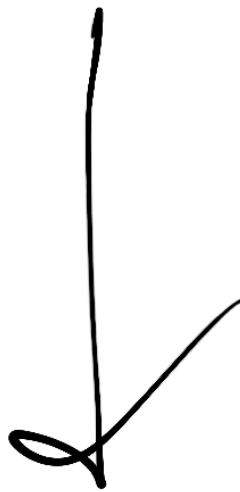
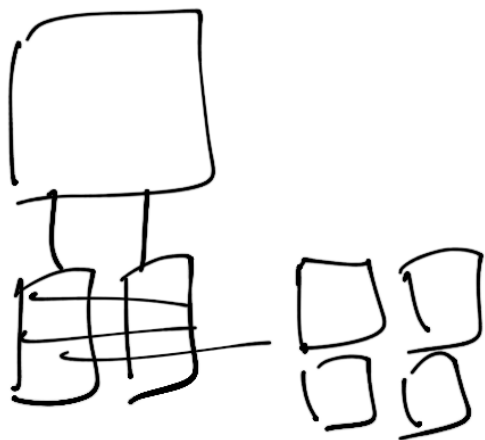
for $j = 0; j < cL; j++$

$$I(i, j) = 0$$

$$I(i + rOff, j) = 0$$

$$I(i, j + cOff) = 0$$

$$I(i + rOff, j + cOff) = 0$$



for ($m=0; m < w \rightarrow kn; m++$)
 $i = i \ll 1; (2i)$

$$I(i, j) =$$

$$\sum_m h_m H_A(i+m, j)$$

$$I(i+roff, j) =$$

$$\sum_m g_m H_A(i+m, j)$$

$$I(i, j+coff) =$$

$$\sum_m h_m G_A(i+m, j)$$

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

$+=$

- ~~body~~ of main k loop:

free (hrA)

free (orA)

r >>= 1; r~~off~~ >>= 1;

c >>= 1; c~~off~~ >>= 1;

}

- and that's it.

- idwt2() I leave to you,
it's similar, BUT you need to
interleave rows, col is ~~rather~~ (see
after)

- otherwise, idwt2) round
"backwards"

- I still use HrA, GrA
as temp images $\hat{=}$

I use rA, cA as temps
for interleaving

- for a1903:
 - implement idwt2D
 - show that you can reconstruct image
- try MSE
- extra stuff:
 - try edge detection via modvl maxima detection

- moduli maxima - see
 notes: wavelet coefficients
 larger or equal to neighbors

$$wf(x-\delta x, y-\delta y) \leq wf(x, y) \leq wf(x+\delta x, y+\delta y)$$

and

$$\left\{ \begin{array}{l} wf(x, y) > wf(x+\delta x, y+\delta y) \\ wf(x, y) > wf(x-\delta x, y-\delta y) \end{array} \right. \quad \underline{\underline{x}}$$

