Image Processing – two-dim. DSP

Simple Property of two-dim. FFT:

IF **Special case** $y_{mn} = y_m * y_n$ (image factors into fctn. of x times fctn. of y)

then transform is

$$Y_{kl} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} (y_m * y_n)\, e^{-j\frac{2\pi k m}{N}}\, e^{-j\frac{2\pi l}{N} n}$$

$$= \left(\sum_{m=0}^{N-1} y_m\, e^{-j\frac{2\pi k m}{N}} \right) * \left(\sum_{n=0}^{N-1} y_n\, e^{-j\frac{2\pi l n}{N}} \right)$$

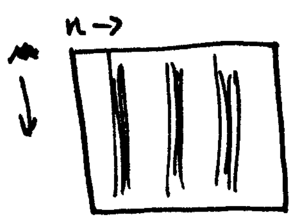
$$= Y_k * Y_l$$

Y_k ↑ transform of m-fctn. Y_l ↑ transform of n-fctn.

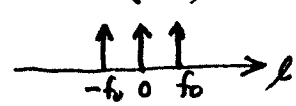
(then transform also factors)

EXAMPLE: ("washboard")

Sinusoid in horizontal direction (n)
constant in vertical direction (m)

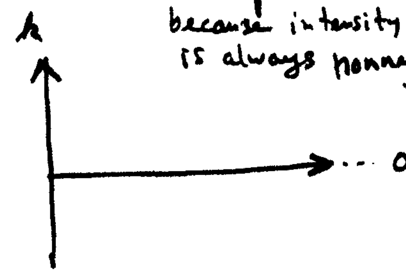


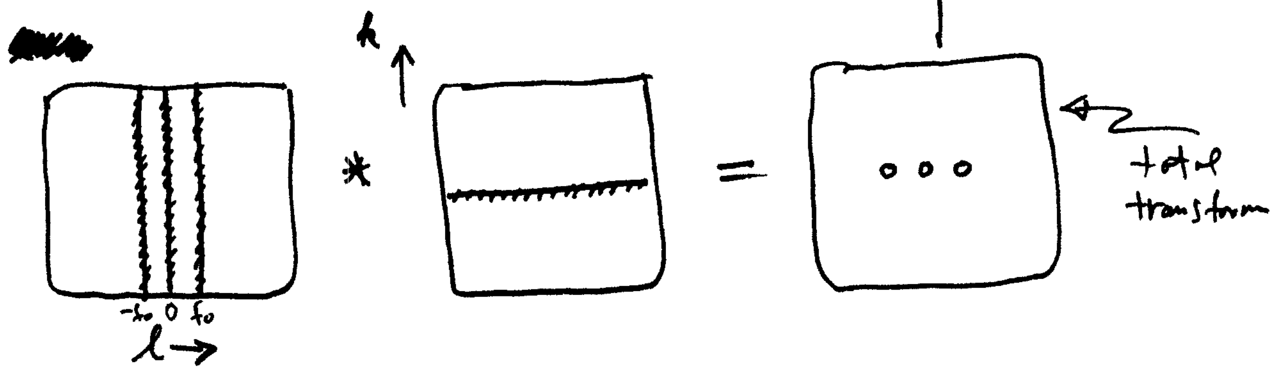
transform of sinusoid:



"DC" component – because intensity is always nonnegative

transform of constant:





washboard.c 5.3.2

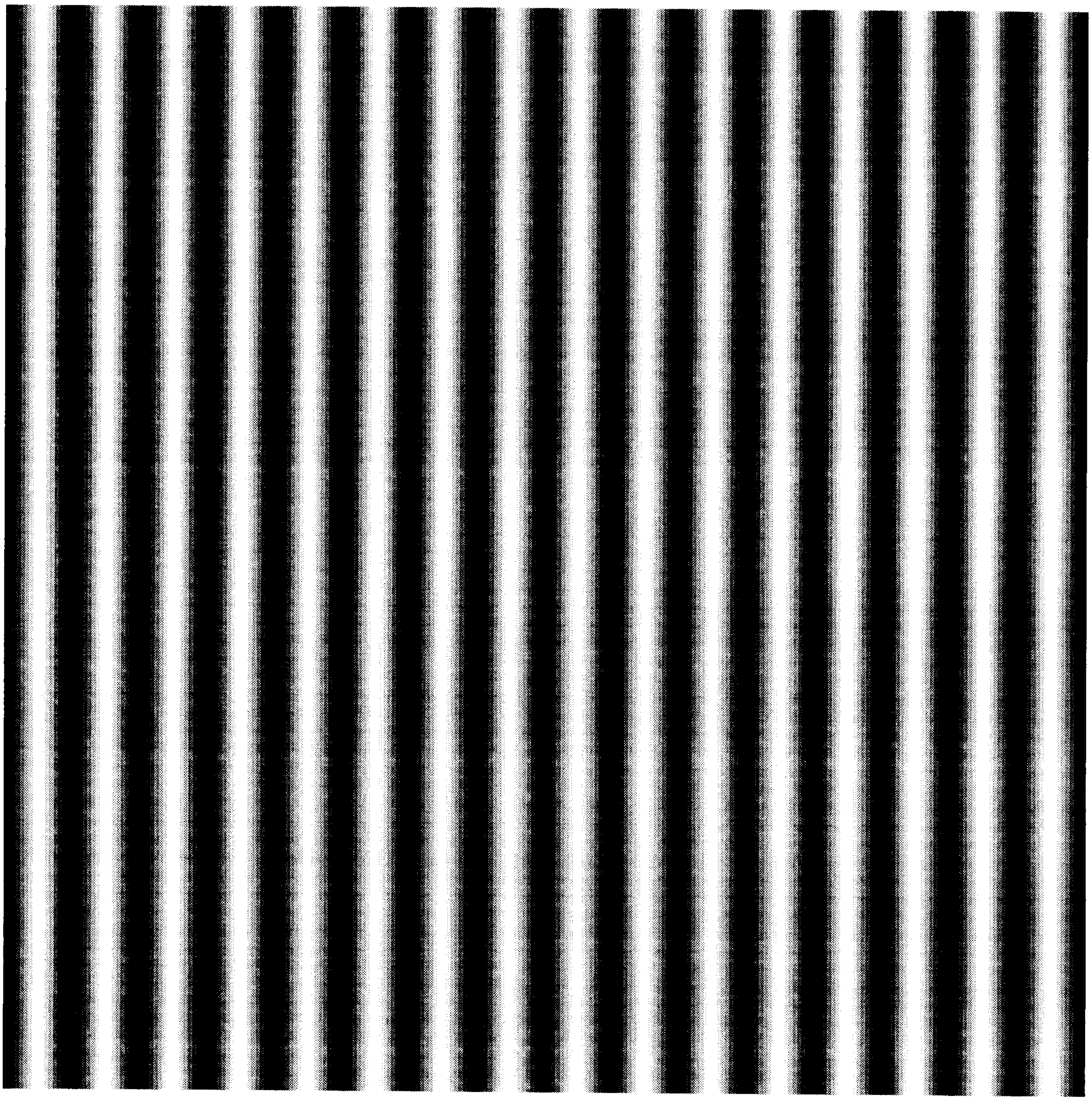
```c
#include "ppm.h"
#include <math.h>

void
main()
{
int    x, y, width, height;
double tx, ty, temp;
#define pi (3.14159265358979)
#define f (8.)
#define window 0
Image *image1;

image1 = ImageCreate(256,256);
ImageClear(image1, 255,255,255); /* all white */

width  = ImageWidth(image1);
height = ImageHeight(image1);

for (y = 0; y < height; y++){
 for (x = 0; x < width; x++){
  if(1){
   temp = sin(2.*f*pi*((double)x/(width-1)));
   tx = 255.*temp*temp;
   if(window==1) tx *= (0.54 + 0.46*cos(2.*pi*x/(width-1)-pi));
   ImageSetPixel(image1, x, y, 0, tx );
   ImageSetPixel(image1, x, y, 1, tx );
   ImageSetPixel(image1, x, y, 2, tx ); }}}

ImageWrite(image1, "washboard.ppm");
}
```

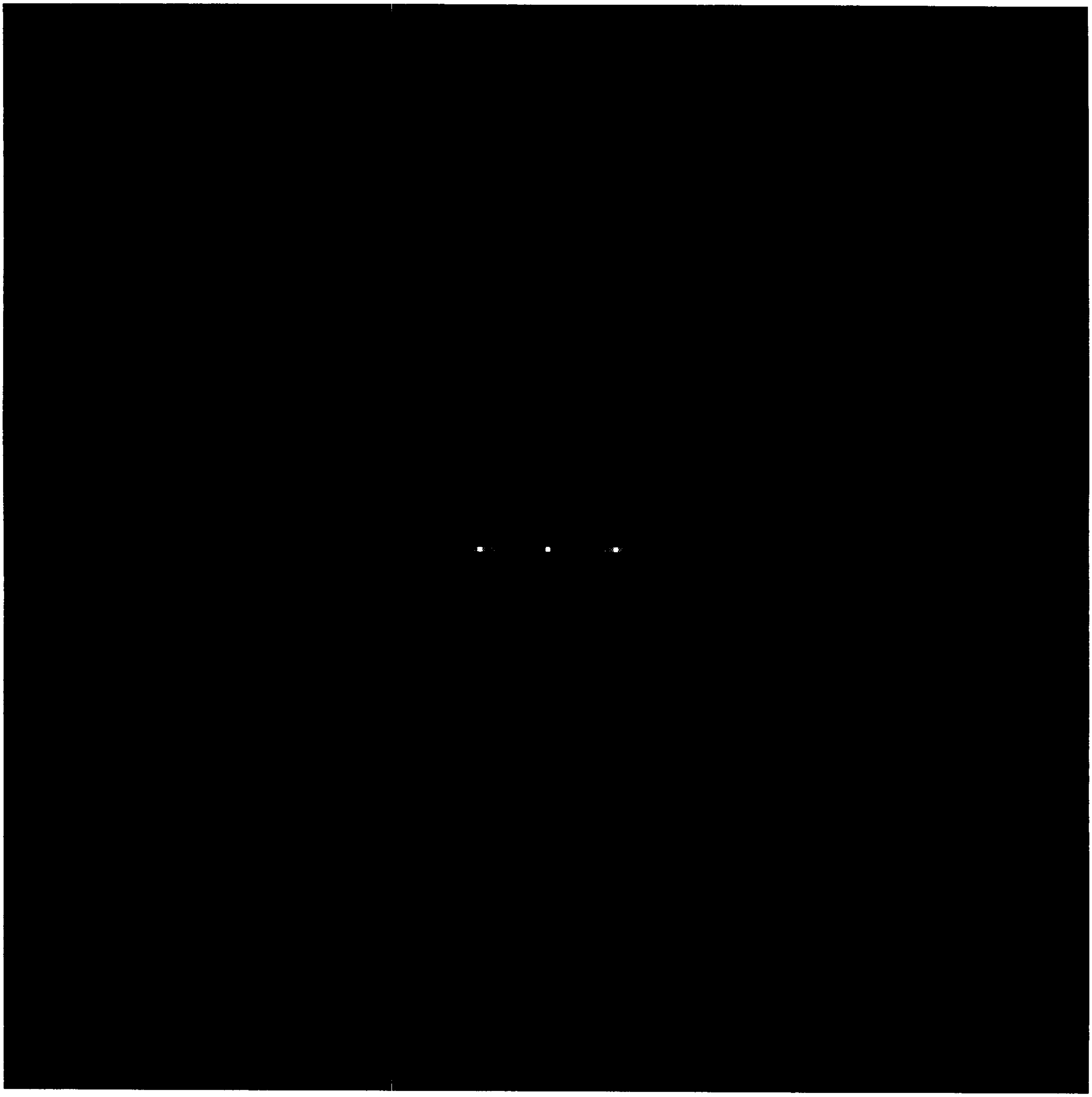
sinusoid in x-direction

0-freq.

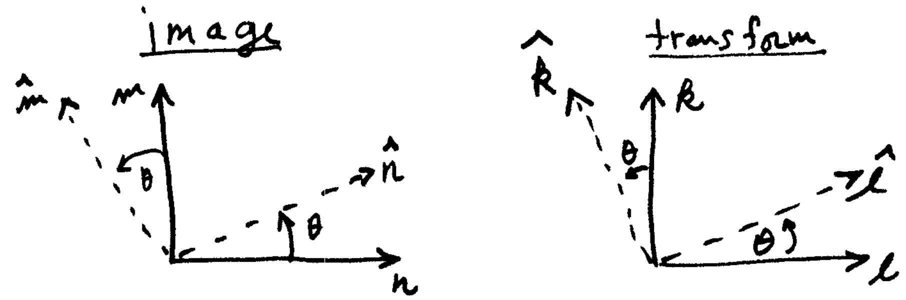
$$\sin^2\theta = \frac{1}{2}(1-\cos 2\theta)$$

Washboard image 5.3.3





Another Important Property of two-dim. FFT:



$$\boxed{\text{rotate image by } \theta \;\Rightarrow\; \text{rotate transform by } \theta}$$

Proof

transform $\quad Y_{k\ell} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} y_{mn}\, e^{-j \frac{2\pi}{N} (mk + n\ell)}$

this is dot product between two vectors

$$\begin{bmatrix} n \\ m \end{bmatrix} \cdot \begin{bmatrix} \ell \\ k \end{bmatrix} = \text{length}\begin{pmatrix} n \\ m \end{pmatrix} \cdot \text{length}\begin{pmatrix} \ell \\ k \end{pmatrix} \cdot \cos \measuredangle_{\text{between}}$$

– invariant with respect to coordinate system rotation

$\therefore \quad \hat{m}\hat{k} + \hat{n}\hat{\ell} = mk + n\ell \qquad (\hat{m}, \hat{n};\ \hat{k}, \hat{\ell} \text{ are rotated})$

$$\therefore \quad \underset{\substack{\uparrow \\ \text{rotated transform}}}{Y_{k\ell}} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \underset{\substack{\uparrow \\ \text{rotated image}}}{y_{mn}}\, e^{-j \frac{2\pi}{N} (\hat{m}\hat{k} + \hat{n}\hat{\ell})}$$

■

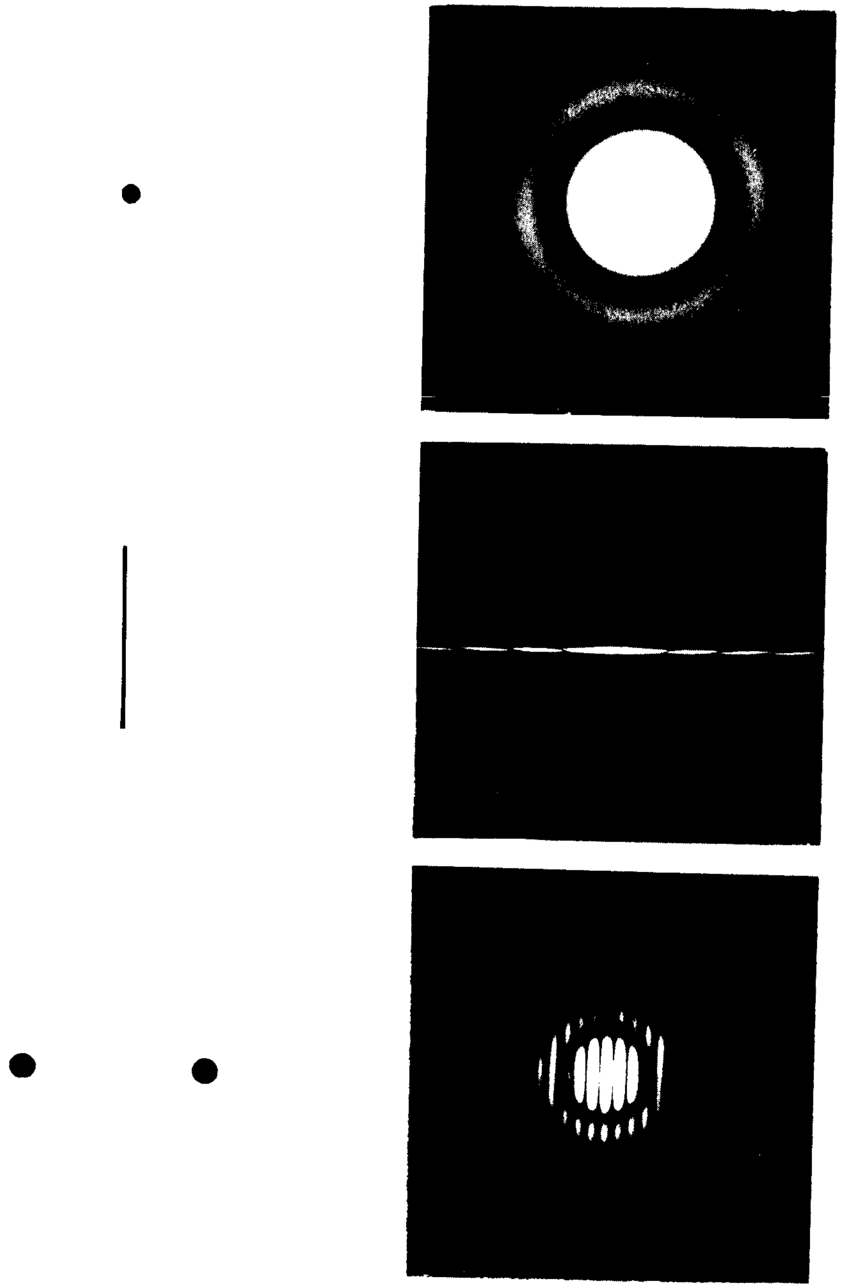


Figure 3.3 Some two-dimensional functions and their Fourier spectra.

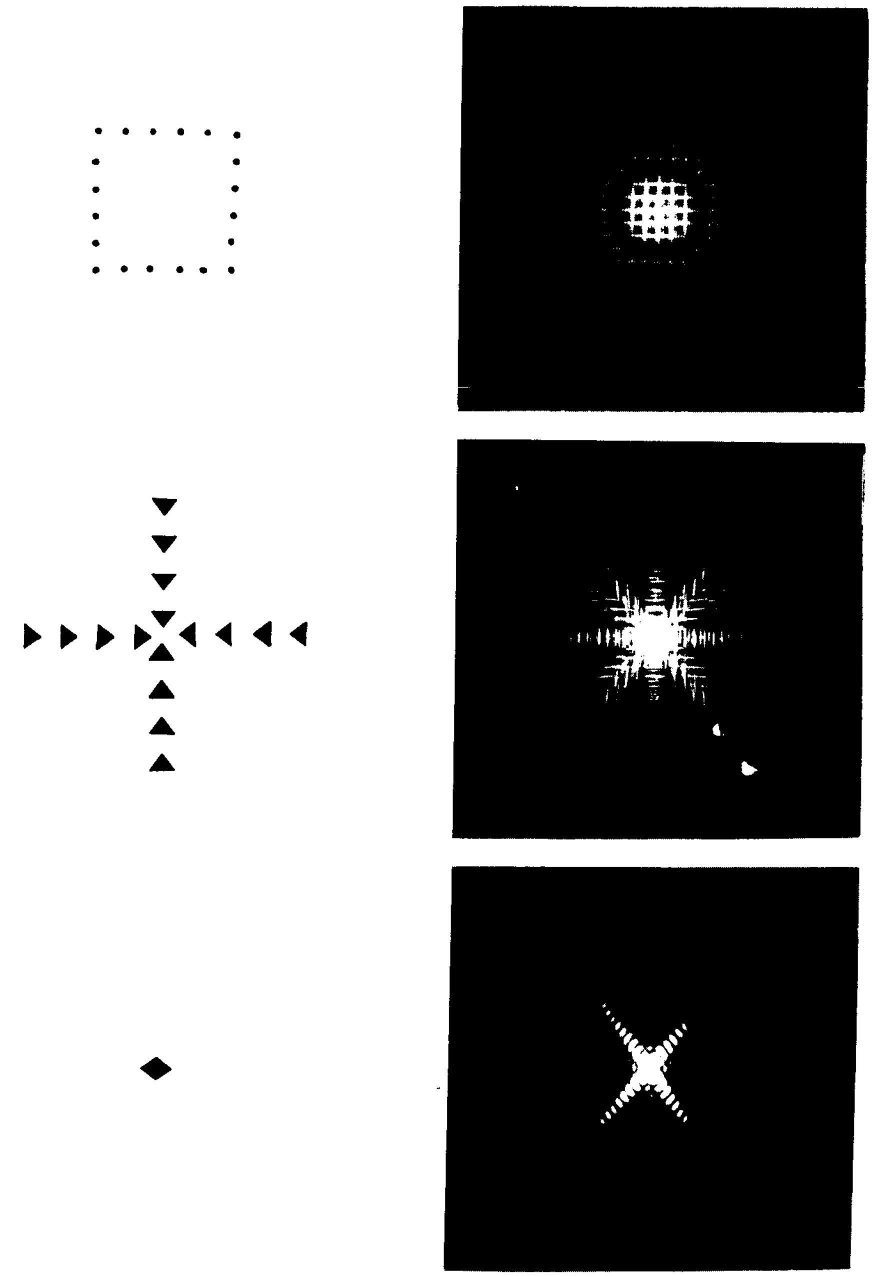
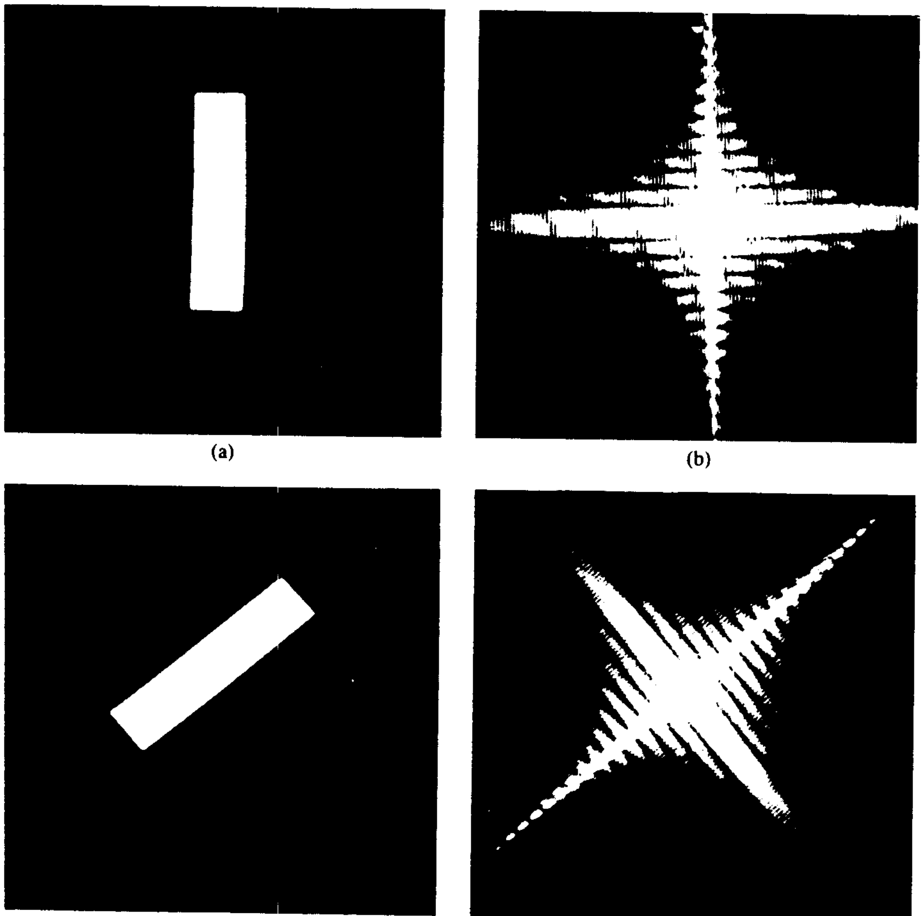


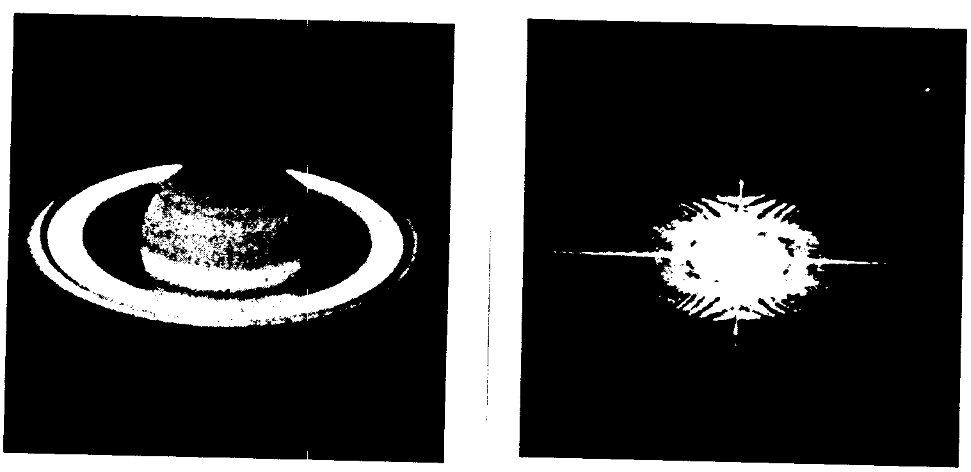
Figure 3.3 (Continued.)



(a) (b)

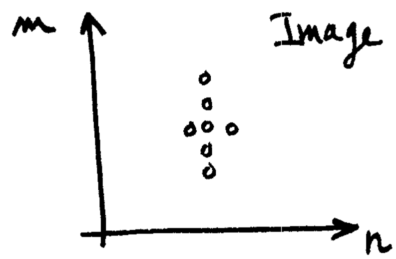
(c) (d)

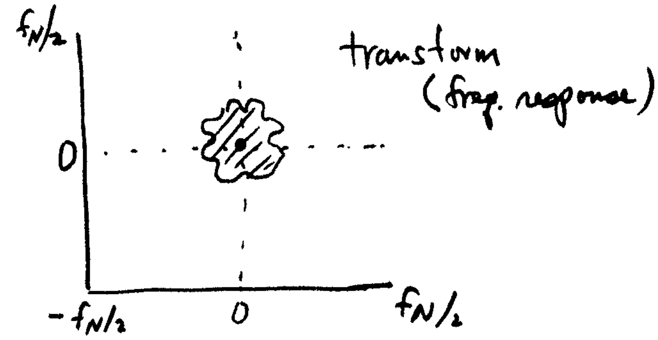
ILLUSTRATING ROTATION



third Property of two-dim. FFT:

BACK to filtering





template, molecule:
replace each point by weighted average of neighbors
called "convolution" or "filtering"

$\Longleftrightarrow$

multiply by transform of template

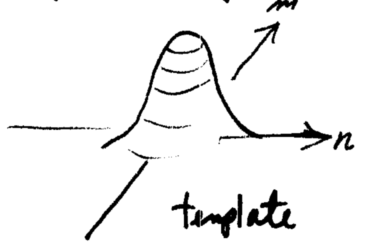
$$\boxed{\text{filtering} \Longleftrightarrow \text{multiplication}}$$

$$\begin{tikzcd} \text{image} \arrow[r, "FFT"] \arrow[d, dashed, "\text{filter}"'] & \text{transform} \arrow[d, "\text{multiply by freq. response}"] \\ {} & {} \arrow[l, "FFT^{-1}"] \end{tikzcd}$$

Guideline: concentrated in space domain $\Longleftrightarrow$ broad in freq. domain

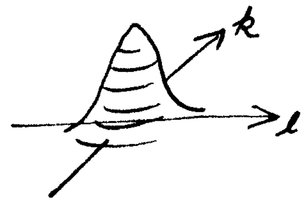
& vice-versa

"blur" in photoshop



template

GAUSSIAN $e^{-\alpha(m^2+n^2)}$



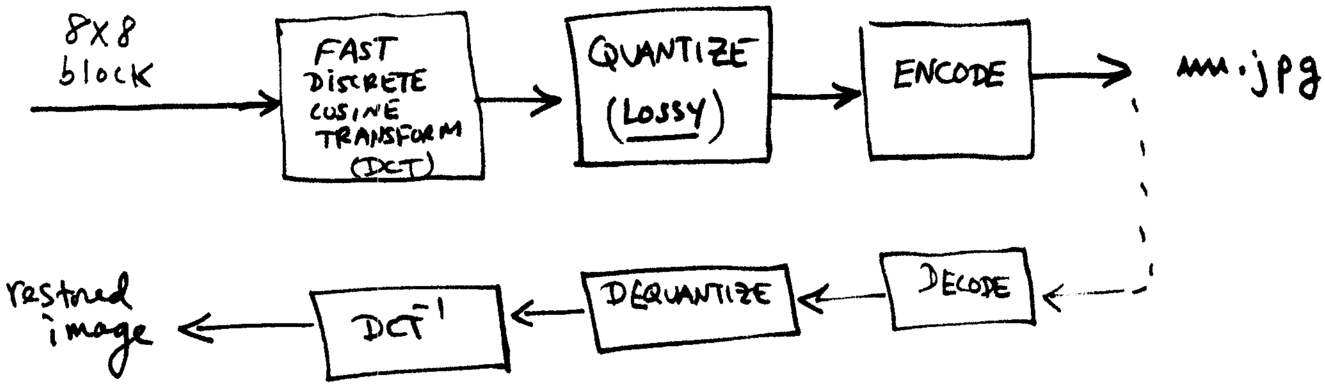
transform of template

GAUSSIAN $e^{-\beta(k^2+l^2)}$

GAUSSIAN is its own transform

JPEG BASELINE IMAGE CODING ALGORITHM [Kou 95]

Use 8x8 blocks



DCT: $$S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{yx} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16}$$

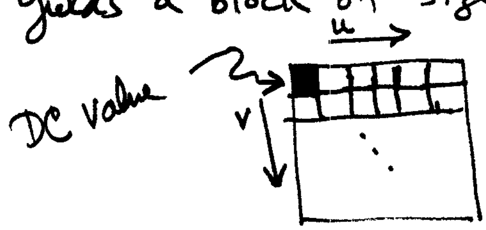
IDCT: $$s_{yx} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v S_{vu} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16}$$

$$\text{where } C_m = \begin{cases} 1/\sqrt{2} & m=0 \\ 1 & \text{else} \end{cases}$$

like FFT - has fast recursive algorithms that have been hacked to death

---

yields a block of size 8x8 holding the transform S_{vu}



Each transform element is quantized by

$$\hat{S}_{vu} = \text{round}\left[\frac{S_{vu}}{Q_{vu}}\right] \quad \text{(integer)}$$

$Q = \begin{bmatrix} \cdots \end{bmatrix}$ is 8x8 fixed matrix provided as a plug-in

small Q $\Rightarrow$ many levels possible, accurate quantization
large Q $\Rightarrow$ few levels possible, rough quantization

to (approximately) restore, $S_{vu} \cong Q_{vu} \hat{S}_{vu}$

not perfectly restored
$\longrightarrow$ "lossy" compaction

---

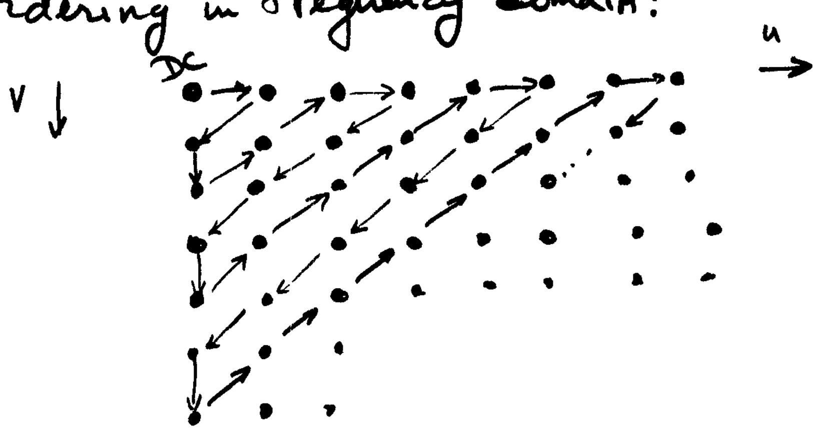
this is first source of compression –
most information is contained in low-frequency components, so we make Q ~~large~~ small for low frequencies, large for high frequencies

---

the DC component [0,0] element is especially important, and is treated separately, using change from previous block. (Second source of compression)

---

to encode Non-DC components, use Zig-Zag ordering in frequency domain:

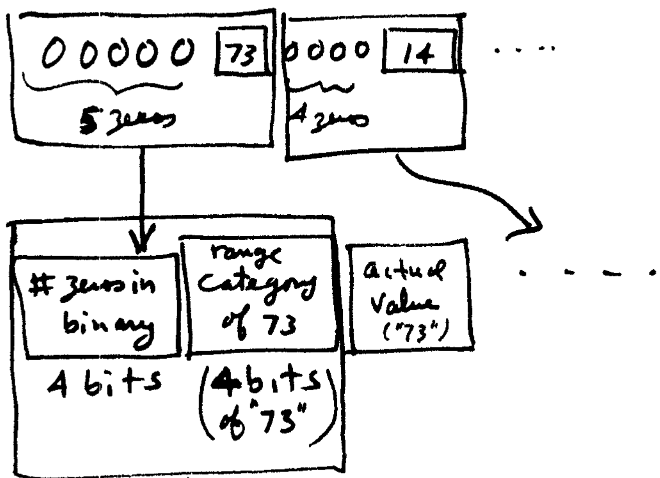


this puts low-frequency information first.

We then use <u>run-length coding</u>, looking for runs of consecutive zeros.

Special code values: $\begin{cases} \text{EOB} = \text{"00000000" (byte)} = \text{rest of block is zero} \\ \text{ZRL} = \text{"11110000" (byte)} = \text{run of 16 zeros} \end{cases}$

<u>Runs of zeros</u> of length less than 16 are encoded as



these bytes are encoded in hex (base 16) as two hex digits

---

the fact that the zig-zag ordering may result in many zeros at the end, & in general produces runs of zeros, is a <u>third</u> <u>source</u> of compression

---

BTW, JPEG = <u>J</u>oint <u>P</u>hotographic <u>E</u>xperts <u>G</u>roup

Many elaborations - JPEG baseline is simplest & most common