Image Processing - two-dim. DSP

Simple Property of two-dim. FFT:

IF Special case: \( Y_{mn} = Y_m \cdot Y_n \) (image factors into
fctn. of \( x \) times fctn. of \( y \))

Then transform is

\[
Y_{kl} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} (Y_m \cdot Y_n) e^{-j \frac{2\pi km}{N}} e^{-j \frac{2\pi ln}{N}}
\]

\[
= \left( \sum_{m=0}^{N-1} Y_m e^{-j \frac{2\pi km}{N}} \right) \ast \left( \sum_{n=0}^{N-1} Y_n e^{-j \frac{2\pi ln}{N}} \right)
\]

(then transform also factors)

\[
= Y_{k} \ast Y_{l}
\]

Transform of \( m \)-fctn.

Transform of \( n \)-fctn.

**EXAMPLE:**

Sinusoid in horizontal direction (\( n \))

Constant in vertical direction (\( m \))

Transform of sinusoid:

\[
\begin{align*}
\text{Transform of constant:} & \quad & \\
\end{align*}
\]

"DC" Component - because intensity is always nonnegative

**TOTAL TRANSFORM**
void
main()
{
#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");
}
washboard image
Another Important Property of two-dim. FFT:

\[
Y_{kl} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} y_{mn} e^{-j \frac{2\pi}{N} (mk + nl)}
\]

This is dot product between two vectors
\[
\left[ \begin{array}{c} n \\ m \end{array} \right] \cdot \left[ \begin{array}{c} \ell \\ k \end{array} \right] = \text{length}(m) \cdot \text{length}(k) \cdot \cos \theta \text{ between}
\]

- invariant with respect to coordinate system rotation

\[
\hat{m} \hat{k} + \hat{n} \hat{l} = m \hat{k} + n \hat{l} \quad (m, n; \hat{k}, \hat{l} \text{ as rotated})
\]

\[
Y_{\hat{kl}} = \sum_{\hat{m}=0}^{N-1} \sum_{\hat{n}=0}^{N-1} y_{\hat{mn}} e^{-j \frac{2\pi}{N} (\hat{m} \hat{k} + \hat{n} \hat{l})}
\]
Figure 3.3. Some two-dimensional functions and their Fourier spectra.
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"

Filtering ↔ multiplication

Guideline: concentrated in space domain ↔ broad in freq. domain & vice-versa

"blur" in photoshop

Template Gaussian \( \ast \text{Gaussian} \)

Transform of template Gaussian \( e^{-\beta(x^2+y^2)} \).

Gaussian is its own transform
JPEG BASELINE IMAGE CODING ALGORITHM [Kov '95]

Use 8x8 blocks

8x8 block → FAST DISCRETE COSINE TRANSFORM (DCT) → QUANTIZE (Lossy) → ENCODE → .jpg

Decompression:

Restored image ← DCT⁻¹ ← DEQUANTIZE ← DECODE ←

DCT:

$$\hat{S}_{uv} = \frac{1}{4} \sum_{x=0}^{7} \sum_{y=0}^{7} N_{x,y} \cos \left(\frac{(2x+1)u\pi}{16}\right) \cos \left(\frac{(2y+1)v\pi}{16}\right)$$

IDCT:

$$a_{x,y} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} \hat{S}_{uv} \cos \left(\frac{(2x+1)u\pi}{16}\right) \cos \left(\frac{(2y+1)v\pi}{16}\right)$$

Where

$$c_m = \begin{cases} \sqrt{1/12} & m = 0 \\ 1 & \text{else} \end{cases}$$

Like FFT - has fast recursive algorithms that have been worked to death

Yields a block of size 8x8 holding the transform Svuv.

DC Value

Each transform element is quantized by

$$\hat{S}_{uv} = \text{round} \left[ \frac{S_{uv}}{Q_{uv}} \right] \text{ (integer)}$$
\[ G = \begin{bmatrix} \vdots & \vdots \\ \vdots & \vdots \end{bmatrix} \] is an 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, \( \hat{S}_{\text{nu}} = Q_{\text{nu}} \hat{Q}_{\text{nu}} \)

\[ \text{not perfectly restored} \quad \rightarrow \text{"lossy" compression} \]

This is the first source of compression —
most information is contained in low-frequency components, so we make \( Q \) small for low frequencies, large for high frequencies.

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

To encode non-DC components, use Zig-Zag ordering in frequency domain:

![Zig-Zag Diagram]

This puts low-frequency information first.
We then use run-length encoding, looking for runs of consecutive zeros.

Special code values:

\[
\begin{align*}
\text{EOB} = \text{"0000000" (byte)} &= \text{rest of block is zero} \\
\text{ZRL} = \text{"11110000" (byte)} &= \text{run of 16 zeros}
\end{align*}
\]

Runs of zeros of length less than 16 are encoded as

```
00000  73
   \_   \_  \
  14  \_  \_
   \_   \_  \
00000  14
```

```
\# zeros in\n binary
4 bits \( (4 \text{ bits of } 73) \)
```

```
\text{these bytes are encoded in}\n\text{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 third source of compression

\text{BTW, JPEG = Joint Photographic Experts Group}

Many elaborations - JPEG baseline is simplest \& most common