Discrete Cosine Transform

Nuno Vasconcelos
UCSD

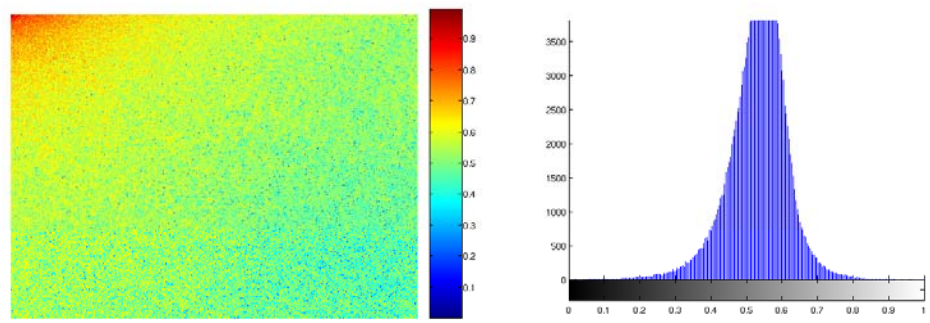
Discrete Fourier Transform

- last classes, we have studied the DFT
- due to its computational efficiency the DFT is very popular
- however, it has strong disadvantages for some applications
  - it is complex
  - it has poor energy compaction
- energy compaction
  - is the ability to pack the energy of the spatial sequence into as few frequency coefficients as possible
  - this is very important for image compression
  - we represent the signal in the frequency domain
  - if compaction is high, we only have to transmit a few coefficients
  - instead of the whole set of pixels

Discrete Cosine Transform

- a much better transform,
  from this point of view, is the DCT
  - in this example we see the amplitude spectra of the image above
  - under the DFT and DCT
  - note the much more concentrated histogram obtained with the DCT
- why is energy compaction important?
  - the main reason is image compression
  - turns out to be beneficial in other applications

DFT



DCT

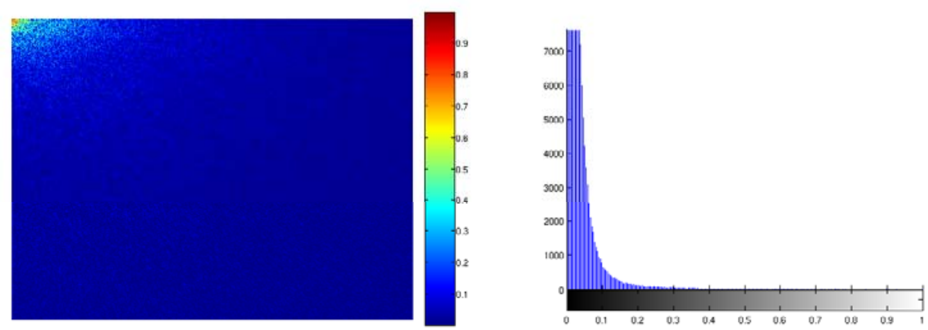
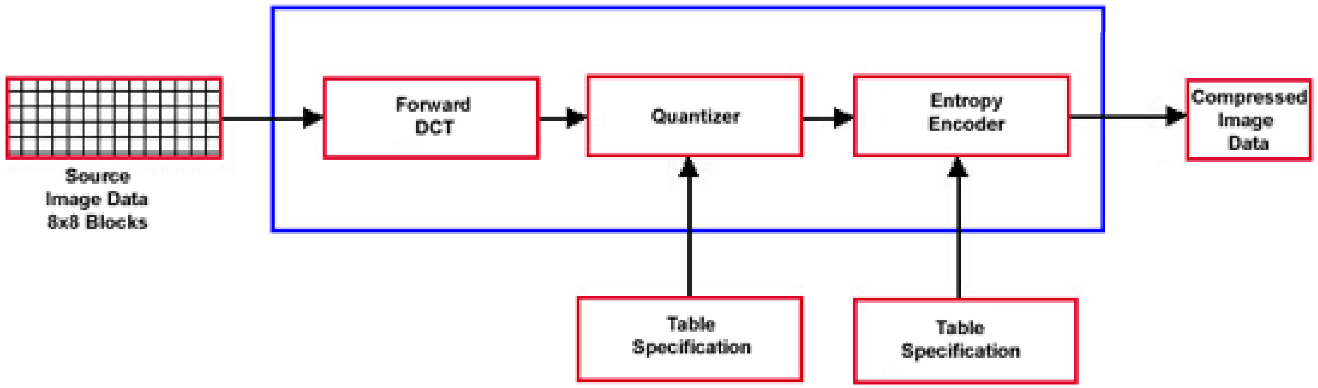


Image compression

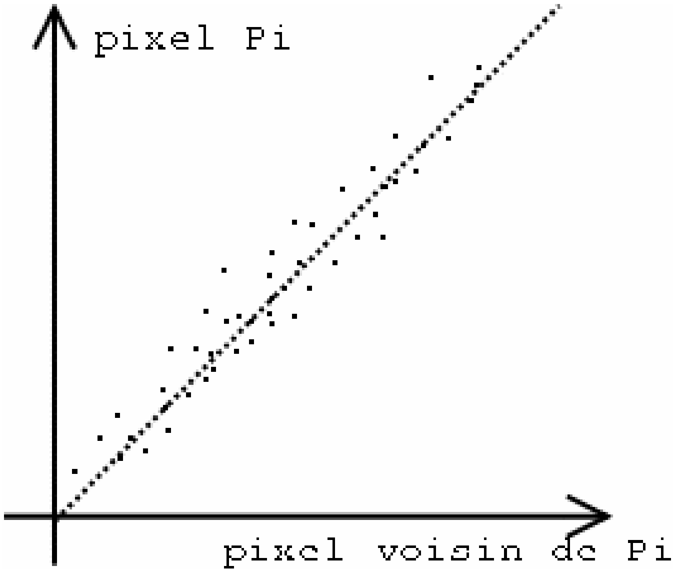
- an image compression system has three main blocks



  – a transform (usually DCT on 8x8 blocks)
  – a quantizer
  – a lossless (entropy) coder
- each tries to throw away information which is not essential to understand the image, but costs bits

Image compression

- the transform throws away correlations
  - if you make a plot of the value of a pixel as a function of one of its neighbors



  - you will see that the pixels are highly correlated (i.e. most of the time they are very similar)
  - this is just a consequence of the fact that surfaces are smooth

Image compression

- the transform eliminates these correlations
  - this is best seen by considering the 2-pt transform
  - note that the first coefficient is always the DC-value

$$\boxed{X[0] = x[0] + x[1]}$$

  - an orthogonal transform can be written in matrix form as

$$\boxed{X = Tx, \qquad T^T T = I}$$

  - i.e. T has orthogonal columns
  - this means that

$$\boxed{X[1] = x[0] - x[1]}$$

  - note that if x[0] similar to x[1], then

$$\boxed{\begin{cases} X[0] = x[0] + x[1] \approx 2x[0] \\ X[1] = x[0] - x[1] \approx 0 \end{cases}}$$

Image compression

- the transform eliminates these correlations
  - note that if x[0] similar to x[1], the

$$\boxed{\begin{cases} X[0] = x[0] + x[1] \approx 2x[0] \\ X[1] = x[0] - x[1] \approx 0 \end{cases}}$$

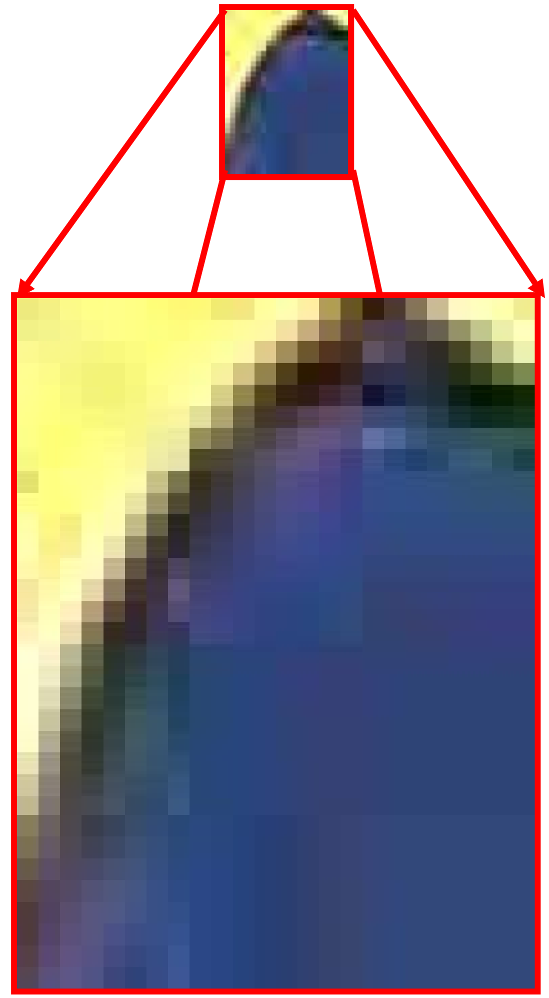
  - in the transform domain we only have to transmit one number without any significant cost in image quality
  - by "decorrelating" the signal we reduced the bit rate to ½!
  - note that an orthogonal matrix

$$\boxed{T^T T = I}$$

    applies a rotation to the pixel space
  - this aligns the data with the canonical axes

Image compression

- a second advantage of working in the frequency domain
  - is that our visual system is less sensitive to distortion around edges
  - the transition associated with the edge masks our ability to perceive the noise
  - e.g. if you blow up a compressed picture, it is likely to look like this



  - in general, the compression errors are more annoying in the smooth image regions

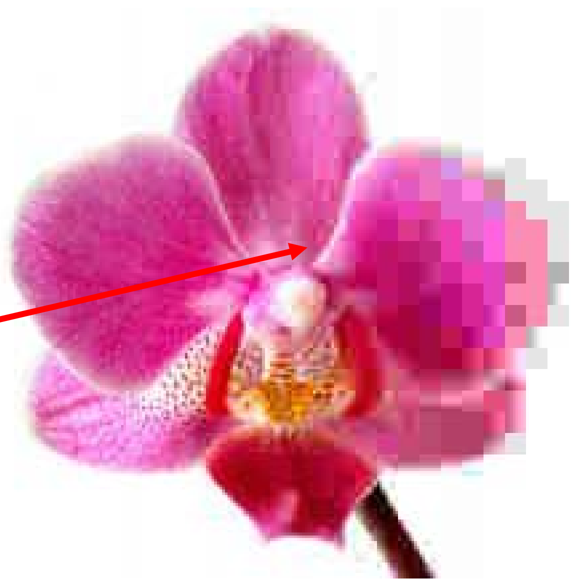


Image compression

- three JPEG examples

36KB

5.7KB

1.7KB

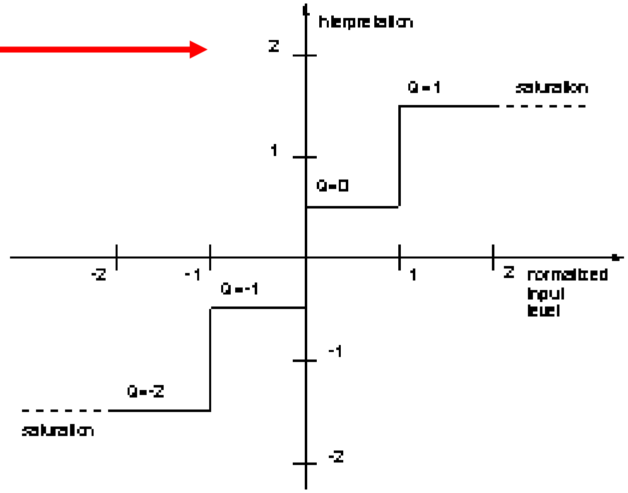
  – note that the blockiness is more visible in the torso

Image compression

- important point: by itself, the transform
  - does not save any bits
  - does not introduce any distortion
- both of these happen when we throw away information
- this is called "lossy compression" and implemented by the quantizer
- what is a quantizer?
  - think of the round() function, that rounds to the nearest integer
  - round(1) = 1; round(0.55543) = 1; round (0.0000005) = 0
  - instead of an infinite range between 0 and 1 (infinite number of bits to transmit)
  - the output is zero or one (1 bit)
  - we threw away all the stuff in between, but saved a lot of bits
  - a quantizer does this less drastically

Quantizer

- it is a function of this type
  - inputs in a given range are mapped to the same output
- to implement this, we
  - 1) define a quantizer step size Q
  - 2) apply a rounding function

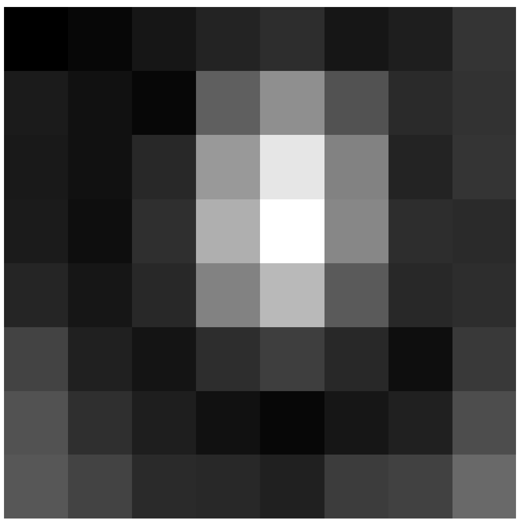


$$\boxed{x_q = round\left(\frac{x}{Q}\right)}$$

  - the larger the Q, the less reconstruction levels we have
  - more compression at the cost of larger distortion
  - e.g. for x in [0,255], we need 8 bits and have 256 color values
  - with Q = 64, we only have 4 levels and only need 2 bits

Quantizer

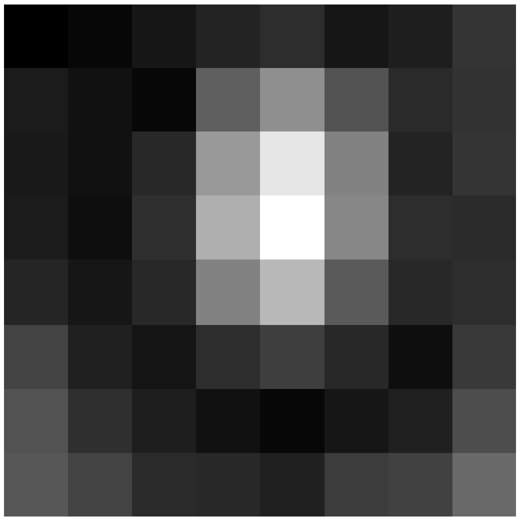
- note that we can quantize some frequency coefficients more heavily than others by simply increasing Q
- this leads to the idea of a quantization matrix
- we start with an image block (e.g. 8x8 pixels)



$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

Quantizer

- next we apply a transform (e.g. 8x8 DCT)



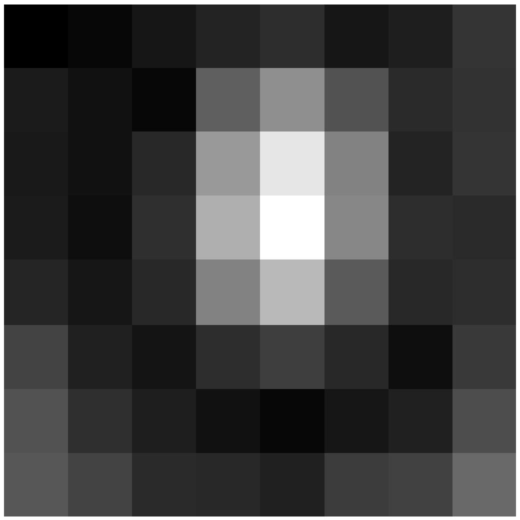
$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

DCT

$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix}$$

Quantizer

- and quantize with a varying Q



$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

DCT

$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix}$$

Q mtx

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Quantizer

- note that higher frequencies are quantized more heavily

Q mtx

$$
\begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 1\text{[illegible]} & \text{[illegible]} & \text{[illegible]} & \text{[illegible]} & \text{[illegible]} \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99
\end{bmatrix}
$$

increasing frequency

  - in result, many high frequency coefficients are simply wiped out

DCT $\longrightarrow$ **quantized DCT**

$$
\begin{bmatrix}
-415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\
4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\
-47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\
-49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\
12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\
-8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\
-1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\
0 & 0 & -1 & -4 & -1 & 0 & 1 & 2
\end{bmatrix}
\longrightarrow
\begin{bmatrix}
-26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\
0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\
-4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Quantizer

- this saves a lot of bits, but we no longer have an exact replica of original image block

DCT

$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix} \longrightarrow$$

quantized DCT

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

inverse DCT

$$\begin{bmatrix} 60 & 63 & 55 & 58 & 70 & 61 & 58 & 80 \\ 58 & 56 & 56 & 83 & 108 & 88 & 63 & 71 \\ 60 & 52 & 62 & 113 & 150 & 116 & 70 & 67 \\ 66 & 56 & 68 & 122 & 156 & 116 & 69 & 72 \\ 69 & 62 & 65 & 100 & 120 & 86 & 59 & 76 \\ 68 & 68 & 61 & 68 & 78 & 60 & 53 & 78 \\ 74 & 82 & 67 & 54 & 63 & 64 & 65 & 83 \\ 83 & 96 & 77 & 56 & 70 & 83 & 83 & 89 \end{bmatrix} \neq$$

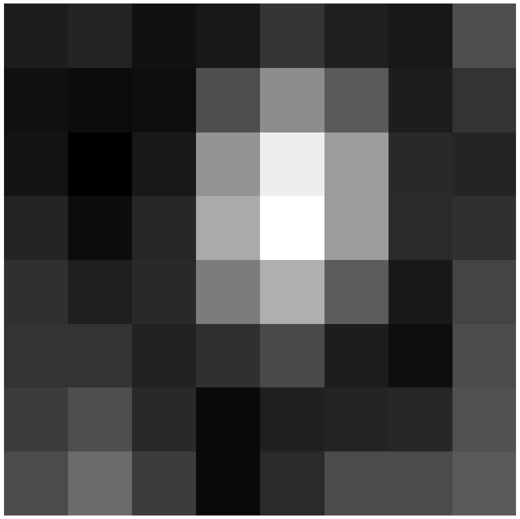
original pixels

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

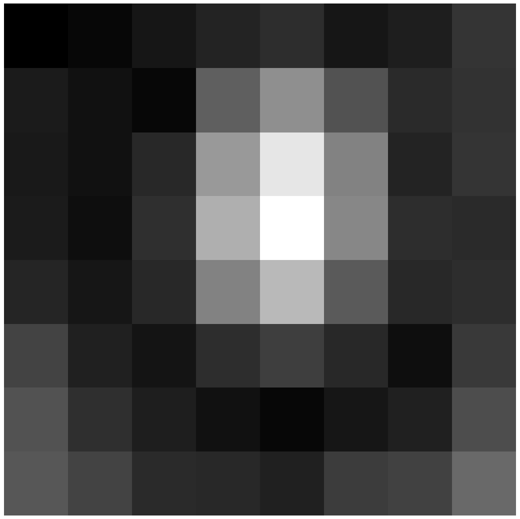
Quantizer

- note, however, that visually the blocks are not very different

original



decompressed



  - we have saved lots of bits without much “perceptual” loss
  - this is the reason why JPEG and MPEG work

Image compression

- three JPEG examples

36KB

5.7KB

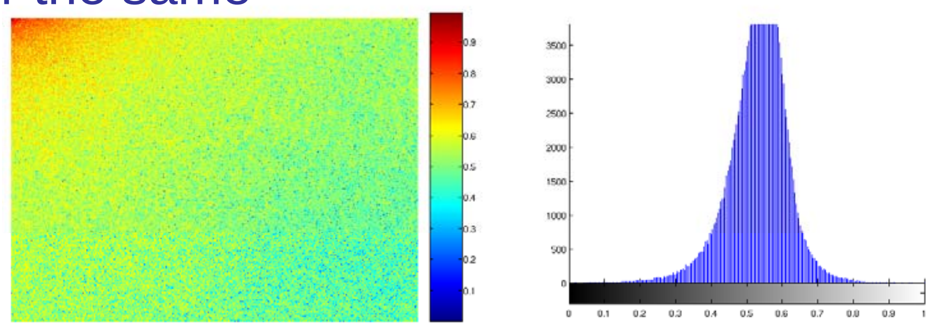
1.7KB

  - note that the two images on the left look identical
  - JPEG requires 6x less bits

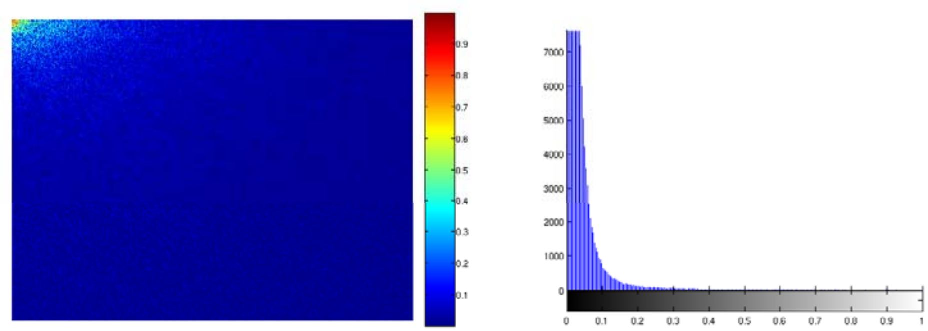
Discrete Cosine Transform

- note that
  - the better the energy compaction
  - the larger the number of coefficients that get wiped out
  - the greater the bit savings for the same loss
- this is why the DCT is important
- we will do mostly the 1D-DCT
  - the formulas are simpler the insights the same
  - as always, extension to 2D is trivial

DFT



DCT



Discrete Cosine Transform

- the first thing to note is that there are various versions of the DCT
  - these are usually known as DCT-I to DCT-IV
  - they vary in minor details
  - the most popular is the DCT-II, also known as even symmetric DCT, or as "the DCT"

$$w[k] = \begin{cases} \frac{1}{2}, & k = 0 \\ 1, & 1 \le k < N \end{cases},$$

$$C_x[k] = \begin{cases} \sum_{n=0}^{N-1} 2x[n]\cos\left(\frac{\pi}{2N}k(2n+1)\right), & 0 \le k < N \\ 0 & otherwise \end{cases}$$

$$x[n] = \begin{cases} \frac{1}{N}\sum_{k=0}^{N-1} w[k]C_x[k]\cos\left(\frac{\pi}{2N}k(2n+1)\right) & 0 \le n < N \\ 0 & otherwise \end{cases}$$

Discrete Cosine Transform

$$\boxed{C_x[k] = \begin{cases} \sum_{n=0}^{N-1} 2x[n]\cos\left(\frac{\pi}{2N}k(2n+1)\right), & 0 \le k < N \\ 0 & otherwise \end{cases}}$$

- from this equation we can immediately see that the DCT coefficients are real
- to understand the better energy compaction
  - it is interesting to compare the DCT to the DFT
  - it turns out that there is a simple relationship
- we consider a sequence x[n] which is zero outside of {0, …, N-1}
- to relate DCT to DFT we need three steps

Discrete Cosine Transform

- step 1): create a sequence

$$\boxed{\begin{aligned} y[n] &= x[n] + x[2N-n-1] \\ &= \begin{cases} x[n], & 0 \le n < N \\ x[2N-n-1], & N \le n < 2N \end{cases} \end{aligned}}$$

- step 2): compute the 2N-point DFT of y[n]

$$\boxed{Y[k] = \sum_{n=0}^{2N-1} y[n] e^{-j\frac{2\pi}{2N}kn}, \quad 0 \le k < 2N}$$

- step 3): rewrite as a function of N terms only

$$Y[k] = \sum_{n=0}^{N-1} y[n] e^{-j\frac{2\pi}{2N}kn} + \sum_{n=N}^{2N-1} y[n] e^{-j\frac{2\pi}{2N}kn}$$

Discrete Cosine Transform

- step 3): rewrite as a function of N terms only

$$Y[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{2N}kn} + \sum_{n=N}^{2N-1} x[2N-n-1] e^{-j\frac{2\pi}{2N}kn}$$

$$= \left(m = 2N-1-n, \quad n = 2N-1-m\right) =$$

$$= \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{2N}kn} + \sum_{m=0}^{N-1} x[m] e^{-j\frac{2\pi}{2N}k(2N-1-m)}$$

$$= \sum_{n=0}^{N-1} x[n] \left\{ e^{-j\frac{2\pi}{2N}kn} + e^{j\frac{2\pi}{2N}kn} \underbrace{e^{-j\frac{2\pi}{2N}k2N}}_{1} e^{j\frac{2\pi}{2N}k} \right\}$$

$$= \sum_{n=0}^{N-1} x[n] \left\{ e^{-j\frac{2\pi}{2N}kn} + e^{j\frac{2\pi}{2N}kn} e^{j\frac{2\pi}{2N}k} \right\}$$

  - to write as a cosine we need to make it into two "mirror" exponents

Discrete Cosine Transform

- step 3): rewrite as a function of N terms only

$$Y[k] = \sum_{n=0}^{N-1} x[n] \left\{ e^{-j\frac{2\pi}{2N}kn} + e^{j\frac{2\pi}{2N}kn} e^{j\frac{2\pi}{2N}k} \right\}$$

$$= \sum_{n=0}^{N-1} x[n] e^{j\frac{\pi}{2N}k} \left\{ e^{-j\frac{2\pi}{2N}kn} e^{-j\frac{\pi}{2N}k} + e^{j\frac{2\pi}{2N}kn} e^{j\frac{\pi}{2N}k} \right\}$$

$$= \sum_{n=0}^{N-1} 2x[n] e^{j\frac{\pi}{2N}k} \cos\left(\frac{\pi}{2N} k(2n+1) \right)$$

$$= e^{j\frac{\pi}{2N}k} \sum_{n=0}^{N-1} 2x[n] \cos\left(\frac{\pi}{2N} k(2n+1) \right)$$

  - from which

$$\boxed{Y[k] = e^{j\frac{\pi}{2N}k} C_x[k], \qquad 0 \le k < 2N}$$

Discrete Cosine Transform

- it follows that

$$\boxed{C_x[k] = \begin{cases} e^{-j\frac{\pi}{2N}k} Y[k], & 0 \le k < N \\ 0, & otherwise \end{cases}}$$

- in summary, we have three steps

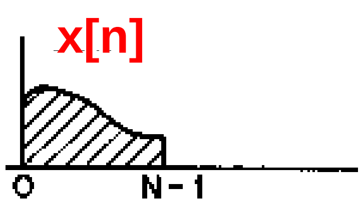
$$\boxed{\underbrace{x[n]}_{N-pt} \leftrightarrow \underbrace{y[n]}_{2N-pt} \overset{DFT}{\leftrightarrow} \underbrace{Y[k]}_{2N-pt} \leftrightarrow \underbrace{C_x[k]}_{N-pt}}$$

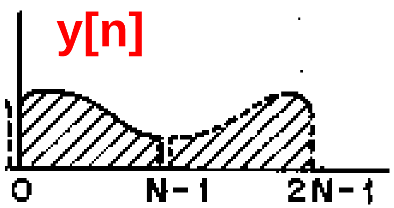
- this interpretation is useful in various ways
  - it provides insight on why the DCT has better energy compaction
  - it provides a fast algorithm for the computation of the DFT

Energy compaction

$$\underbrace{x[n]}_{N-pt} \leftrightarrow \underbrace{y[n]}_{2N-pt} \overset{DFT}{\leftrightarrow} \underbrace{Y[k]}_{2N-pt} \leftrightarrow \underbrace{C_x[k]}_{N-pt}$$

- to understand the energy compaction property
  - we start by considering the sequence y[n] = x[n]+x[2N-1-n]
  - this just consists of adding a mirrored version of x[n] to itself





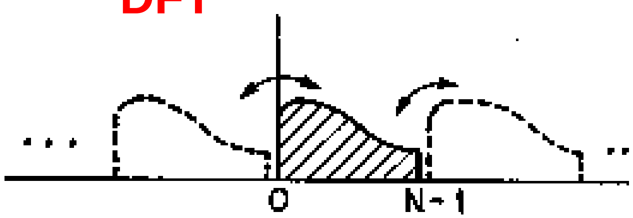
  - next we remember that the DFT is identical to the DFS of the periodic extension of the sequence
  - let's look at the periodic extensions for the two cases
    - when transform is DFT: we work with extension of x[n]
    - when transform is DCT: we work with extension of y[n]

Energy compaction

$$\boxed{\underbrace{x[n]}_{N-pt} \leftrightarrow \underbrace{y[n]}_{2N-pt} \overset{DFT}{\leftrightarrow} \underbrace{Y[k]}_{2N-pt} \leftrightarrow \underbrace{C_x[k]}_{N-pt}}$$

- the two extensions are

DFT



DCT

  - note that in the DFT case the extension introduces discontinuities
  - this does not happen for the DCT, due to the symmetry of y[n]
  - the elimination of this artificial discontinuity, which contains a lot of high frequencies,
  - is the reason why the DCT is much more efficient

Fast algorithms

- the interpretation of the DCT as

$$\boxed{\underbrace{x[n]}_{N-pt} \leftrightarrow \underbrace{y[n]}_{2N-pt} \overset{DFT}{\leftrightarrow} \underbrace{Y[k]}_{2N-pt} \leftrightarrow \underbrace{C_x[k]}_{N-pt}}$$

  - also gives us a fast algorithm for its computation
  - it consists exactly of the three steps
  - 1) y[n] = x[n]+x[2N-1-n]
  - 2) Y[k] = DFT{y[n]}

    this can be computed with a 2N-pt FFT
  - 3)

$$\boxed{C_x[k] = \begin{cases} e^{-j\frac{\pi}{2N}k} Y[k], & 0 \le k < N \\ 0, & otherwise \end{cases}}$$

  - the complexity of the N-pt DCT is that of the 2N-pt DFT

2D DCT

- the extension to 2D is trivial
- the procedure is the same

$$\underbrace{x[n_1,n_2]}_{N_1\times N_2-pt} \leftrightarrow \underbrace{y[n_1,n_2]}_{2N_1\times 2N_2-pt} \overset{2\text{D } DFT}{\leftrightarrow} \underbrace{Y[k_1,k_2]}_{2N_1\times 2N_2-pt} \leftrightarrow \underbrace{C_x[k_1,k_2]}_{N_1\times N_2-pt}$$

- with

$$\begin{aligned} y[n_1,n_2] = {} & x[n_1,n_2] + x[2N_1-1-n_1,n_2] \\ & + x[n_1,2N_2-1-n_2] + x[2N_1-1-n_1,2N_2-1-n_2] \end{aligned}$$

- and

$$C_x[k_1,k_2] = \begin{cases} e^{-j\frac{\pi}{2N_1}k_1} e^{-j\frac{\pi}{2N_2}k_2} Y[k_1,k_2], & \begin{array}{l} 0 \le k_1 < N_1 \\ 0 \le k_2 < N_2 \end{array} \\ 0, & otherwise \end{cases}$$

2D DCT

- the end result is the 2D DCT pair

$$C_x[k_1,k_2]=\begin{cases}\displaystyle\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}4x[n_1,n_2]\cos\left(\frac{\pi}{2N_1}k_1(2n_1+1)\right)\cos\left(\frac{\pi}{2N_2}k_2(2n_2+1)\right), & \begin{array}{l}0\le k_1<N_1\\ 0\le k_2<N_2\end{array}\\ 0 & \textit{otherwise}\end{cases}$$

$$x[n_1,n_2]=\begin{cases}\displaystyle\frac{1}{N_1N_2}\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1}w_1[k_1]w_2[k_2]C_x[k_1,k_2]\cos\left(\frac{\pi}{2N_1}k_1(2n_1+1)\right)\cos\left(\frac{\pi}{2N_2}k_2(2n_2+1)\right) & \begin{array}{l}0\le n_1<N_1\\ 0\le n_2<N_2\end{array}\\ 0 & \textit{otherwise}\end{cases}$$

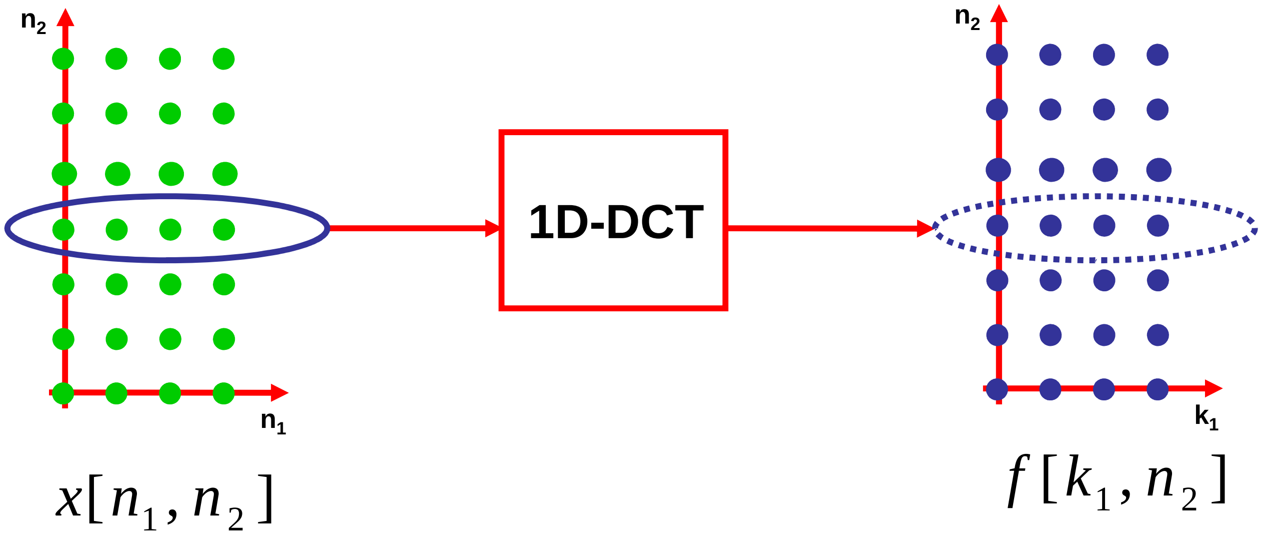
with

$$w_1[k_1]=\begin{cases}1/2, & k_1=0\\ 1, & 1\le k_1<N_1\end{cases},\qquad w_2[k_2]=\begin{cases}1/2, & k_2=0\\ 1, & 1\le k_2<N_2\end{cases}$$

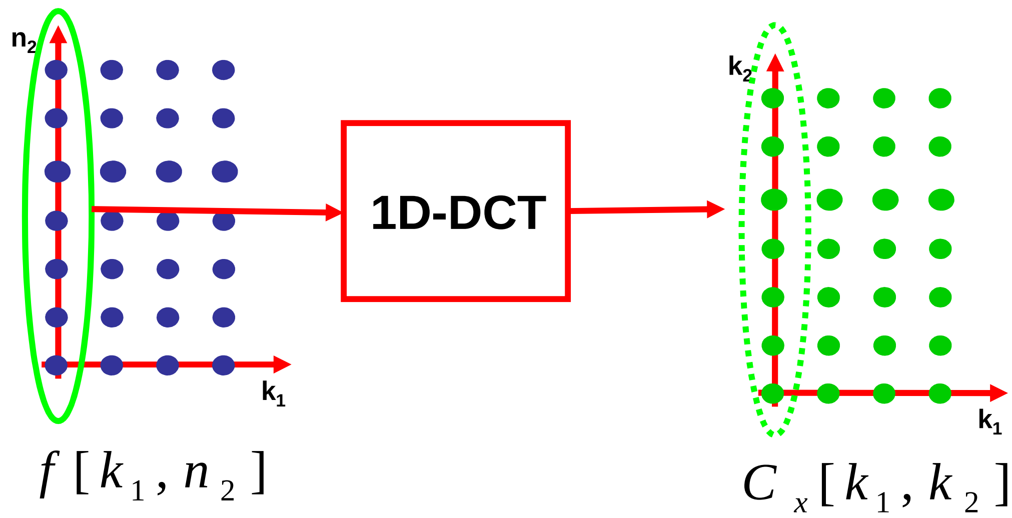
- it is possible to show that the 2DCT can be computed with the row-column decomposition (homework)

2D-DCT

- 1) create intermediate sequence by computing 1D-DCT of rows



- 2) compute 1D-DCT of columns



Any questions?