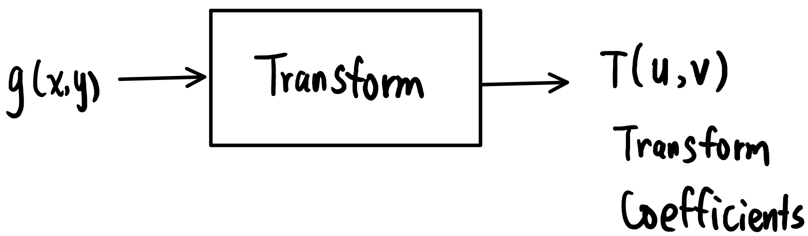
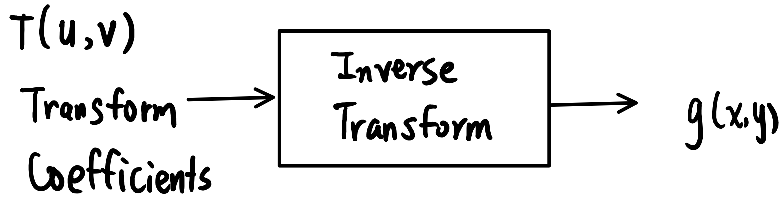
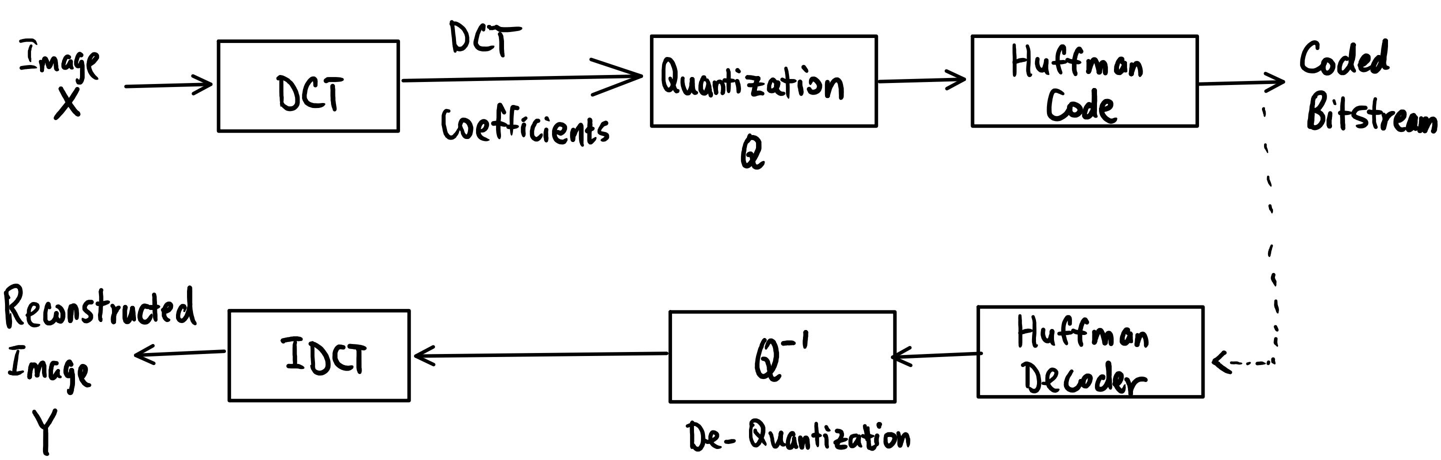
# Lecture 24

## Transform Coding




## JPEG Lossy Image Compression Using DCT




Block Transform:
Input: nxn block
Output: nxn block

$$\begin{cases} T(u,v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} g(x,y)\, r(x,y,u,v) \\ g(x,y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u,v)\, s(x,y,u,v) \end{cases}$$

```
x =
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
```

```
>> y = dct2(x)
y =
   34.0000         0         0         0
         0         0   13.5140         0
         0    3.3785         0    2.9302
         0         0   11.7206         0
```

## Frobenius Norm

The Frobenius norm of an m-by-n matrix X (with m,n >= 2) is defined by

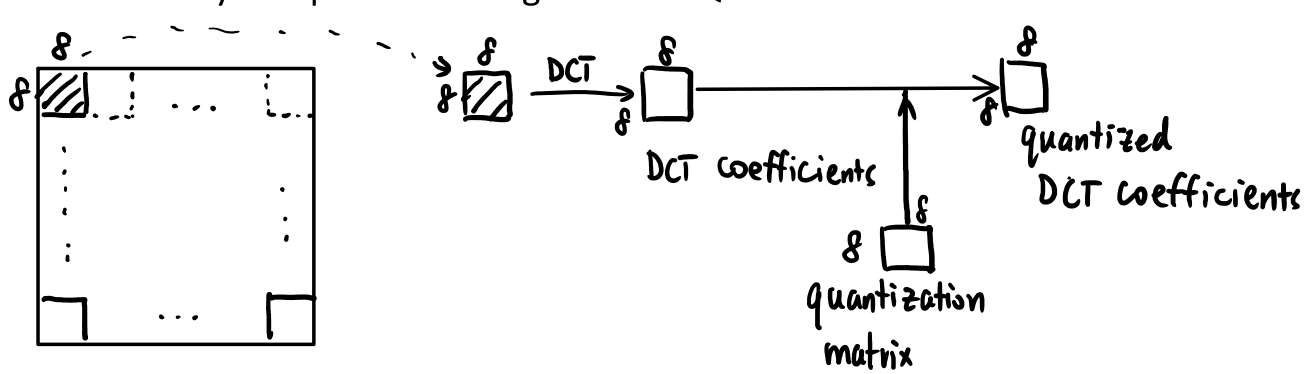
$$\|X\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{\mathrm{trace}(X^\dagger X)}\ .$$

```
>> (norm(x,'fro'))^2
ans =
      1496
```

```
>> (norm(y,'fro'))^2
ans =
   1.4960e+03
```
(energy is preserved)

```
>> x_bar = idct2(y)
x_bar =
   16.0000    2.0000    3.0000   13.0000
    5.0000   11.0000   10.0000    8.0000
    9.0000    7.0000    6.0000   12.0000
    4.0000   14.0000   15.0000    1.0000
```

Block-based lossy compression using DCT and Quantization



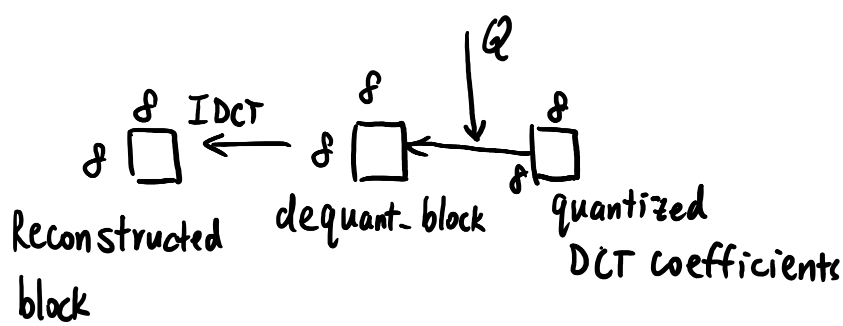
$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120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

https://en.wikipedia.org/wiki/Quantization_(image_processing)

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

quant_coeff (i, j) = round (dct_coeff (i, j) / Quant_Matrix (i,j));

dequant_block(i, j) = quant_coeff (i, j) x Quant_Matrix (i, j);




```
>> I = imread('coins.png');
>> imshow(I)
>> J = dct2(I);
>> whos J
  Name        Size            Bytes  Class     Attributes
  J         246x300          590400  double

>> whos I
```

```
  Name        Size           Bytes  Class    Attributes
  I        246x300           73800  uint8
```

```
>> max(J(:))
ans =
2.7975e+04
```

```
>> min(J(:))
ans =
  -4.2375e+03
```

```
>> sum(I(:))/(sqrt(246*300))
ans =
2.7975e+04
```

Using DCT2 definition

```
>> J(1:1)
ans =
2.7975e+04
```

'DC' component

```
>> (max(J(:)))^2/(norm(double(I),'fro')^2)
ans =
   0.7725
```

77.25% of total energy contained by the 'DC' DCT coefficient.

## Discrete Cosine Transform

The discrete cosine transform (DCT) is closely related to the discrete Fourier transform. It is a separable linear transformation; that is, the two-dimensional transform is equivalent to a one-dimensional DCT performed along a single dimension followed by a one-dimensional DCT in the other dimension. The definition of the two-dimensional DCT for an input image A and output image B is

$$B_{pq} = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos\frac{\pi(2m+1)p}{2M} \cos\frac{\pi(2n+1)q}{2N}, \quad \begin{array}{l} 0 \le p \le M-1 \\ 0 \le q \le N-1 \end{array}$$

If $p=0$, $q=0$, then

$$B_{00} = \alpha_0 \alpha_0 \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn}$$

where

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}}, & p = 0 \\ \sqrt{\frac{2}{M}}, & 1 \le p \le M-1 \end{cases}$$

Where

$$a_0 = \frac{1}{\sqrt{M}}$$

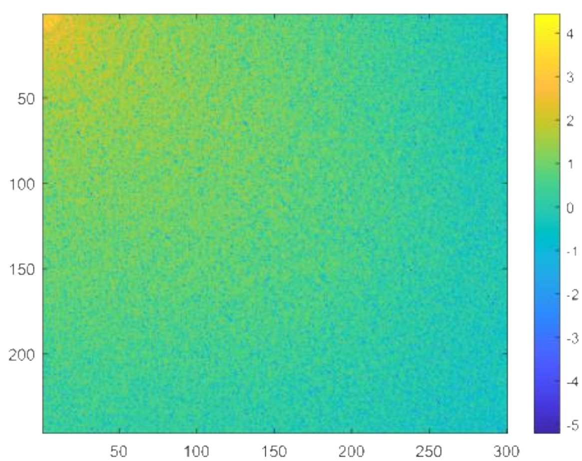
$$a_0 = \frac{1}{\sqrt{N}}$$

and

$$\alpha_q = \begin{cases} \frac{1}{\sqrt{N}}, & q = 0 \\ \sqrt{\frac{2}{N}}, & 1 \le q \le \text{N-1} \end{cases}$$

*M* and *N* are the row and column size of A, respectively.

```
>> figure; imagesc(log10(abs(J)));
>> colorbar;
```




```
>> K = idct2(J);
>> whos K
  Name       Size           Bytes  Class
Attributes
  K        246x300          590400  double

>> isequal(uint8(K), I)
ans =
  logical
   1
```