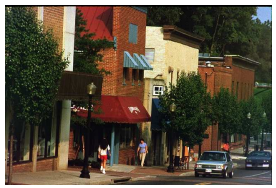
Lecture 23:

JPEG Lossy Image Compression

December 1, 2006

Lossy Compression 1

- Many kinds of data — such as images and audio signals — contain "noise" and other information that is not really of interest. Preserving such useless information seems wasteful.
- **A common approach:** *Lossy compression*, for which decompressing a compressed file gives you something *close* to the original, but not necessarily exactly the original.
- We should be able to compress to a smaller file size if we don't have to reproduce the original exactly.


100% size


5% size

What do We Mean by "Close"? 2

- Any lossy compression scheme is based (at least implicitly) on some idea of what counts as "close to the original".
- This is a question that can only be answered by considering the *users* of the compression program, and what they want.
- For images and audio signals, two fundamental issues are:
  - What differences can humans perceive? For example, to a first approximation, humans perceive only *relative energies* of different frequency bands in audio but not the associated *phases* of sine waves.

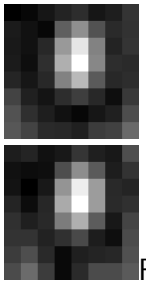
  - What differences do humans find annoying or distracting? Slight changes in colour might be regarded as less important than making a straight line be jagged.

JPEG Lossy Image Compression 3

- JPEG is a lossy image compression standard for full colour or grayscale pictures, named after the *Joint Photographic Experts Group* which defined it.
- The basic method involves breaking the image into small blocks, transforming each block into a Fourier basis, quantizing the coefficients and encoding the quantized coefficients using either Huffman codes or arithmetic coding.
- The algorithm achieves much of its compression by exploiting known limitations of the human eye, notably the fact that small color details aren't perceived as well as small details of light-and-dark. Thus, JPEG is intended for compressing images that will be looked at by humans.
- Unlike the GIF standard, colour quantization is *not* part of JPEG, but many JPEG viewers try to do quantization which can severely affect the quality if done poorly.

JPEG Compression Algorithm 4

- The baseline compression algorithm involves several steps:
  1. Transform into a luma/chroma colour space.
  2. Chroma subsampling.
  3. DCT on image blocks.
  4. Quantize DCT coefficients.
  5. Encode quantized coefficients.



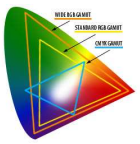
Original Block

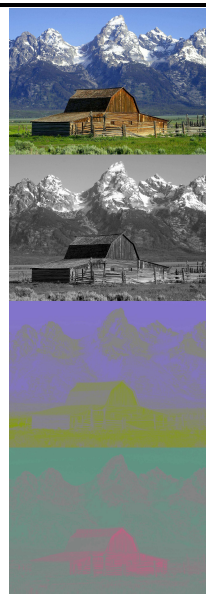
Reconstructed

- Usually the threshold of visible difference from the source image is somewhere around 10x to 20x smaller than the original, ie, 1 to 2 bits per pixel for color images.
- At 3x smaller it is usually impossible to tell any difference at all.

Transforming the Colour Space 5

- Since the human eye is more sensitive to high-frequency details of brightness than to colour, we first transform the image representation into a new colour space called "YCbCr".
  (For grayscale images, we do nothing.)
- This new space represents the luminosity (Y) and chromacity (CbCr) separately, allowing us to compress the latter more than the former. Broadcast television does exactly this also.
- Colourspace transformation is slightly lossy due to roundoff error, but the amount of error here is much smaller than what we typically introduce later on.





Spatially Downsample Chroma Signal 6

- The next step is "chroma subsampling": we spatially downsample each of the chroma channels either in the horizontal or both the horizontal and vertical directions.
- Typically we leave the luminance channel at full resolution, and downsample the chroma channels by a factor of 2 horizontally. (This is called "2h1v" or "422" sampling.)
  We may additionally downsample the chroma channels by a factor of 2 vertically (which is called "2h2v" or "411" sampling).
- This step immediately reduces the data volume by one-half or one-third. In numerical terms it is highly lossy, but for most images it has almost no impact on perceived quality, because of the eye's poorer resolution for chroma info.
- Note that downsampling is not applicable to grayscale data; this is one reason color images are more compressible than grayscale.

Discrete Cosine Transform on Blocks 7

- The main step of JPEG is to group the pixel values into 8x8 blocks. We transform each channel of each 8x8 block by a *discrete cosine transform* (DCT), which maps it to frequency components.
- By quantizing coefficients corresponding to high-frequency information more and those corresponding to low-frequencies less, we can efficiently preserve the main spatial structure of the block.

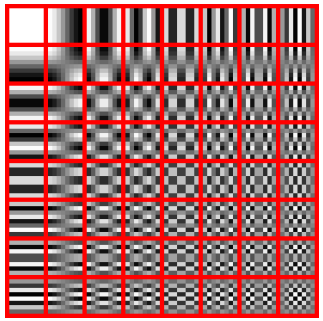
Example

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

⇐ Original
⇐ Block

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

⇐ After
⇐ DCT



Quantize DCT Coefficients 8

- Next, we divide each of the 64 DCT coefficients by a "quantization factor" and round down the result. This is the fundamental compression step.
- The larger the quantization factor, the more data we discard, so we use larger factors for the high-frequency coefficients.
- There are separate quantization factors for the luma and chroma channels.
- Most coders use a quantization factors which are a simple linear scaling of the standard factors given in the JPEG standard; this scaling is the user specified "quality level".

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

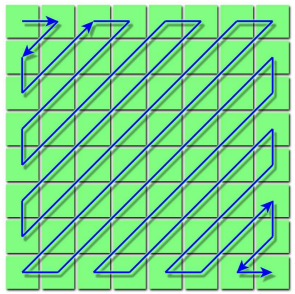
$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120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

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

Encode Quantized Coefficients 9

- Finally, we (losslessly) encode the reduced coefficients using either Huffman or arithmetic coding based on a particular zig-zag "scan order".
- The input alphabet uses run-length encoding which captures the sign and magnitude of the coefficients as well as a special "end-of-block" symbol (EOB) to indicate that all the rest of the coefficients are zero.
- To decode, we reverse the steps above, namely we multiply by the quantization coefficients, take the inverse DCT, and place the block into the appropriate location and channel of the YCbCr image.

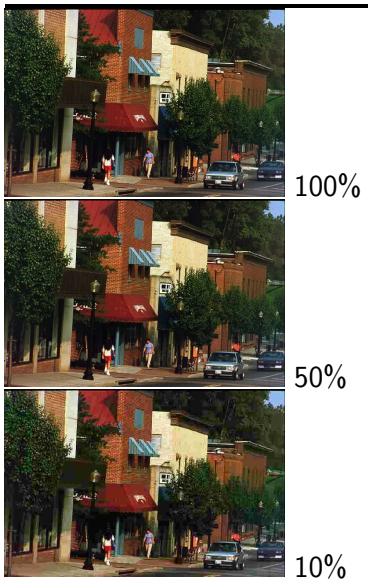
$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



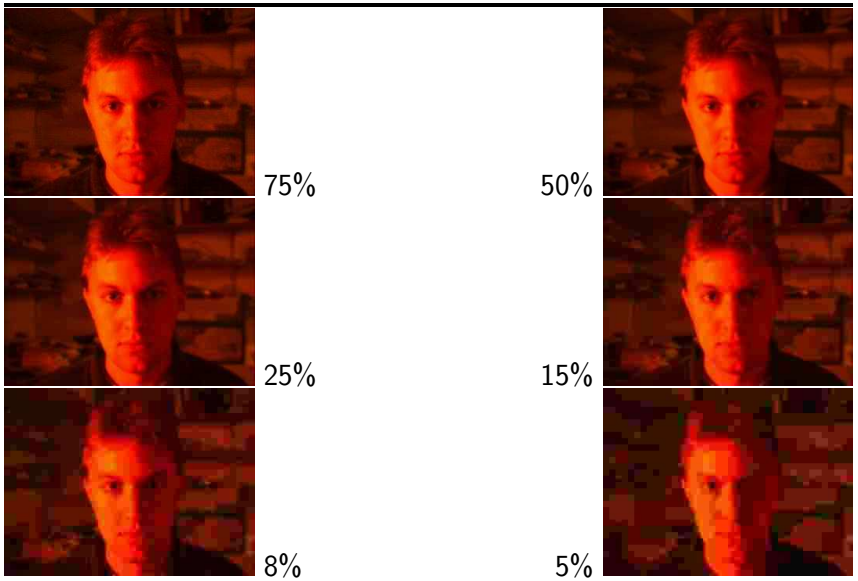
Examples – JPEG Compression 10

Sizes: 100%:83K 50%:15K
25%9K 10%5K 1%:1.5K

Examples – JPEG Compression 11



Examples – JPEG Compression 12



Examples – JPEG Compression 13