Images on the Web

- Use the `<img>` tag to add an image to a Web page
  - `<img src="picture.gif"/>`
- But must trade-off
  - image quality
  - download time

Formats

- Two major image formats
  - GIF (Graphics Interchange Format)
    - GIF87a, GIF89a
    - lossless coding using run-length encoding
    - can only handle 256 different colours
    - can do animation / transparency
    - better suited for line art and smaller images
  - JPEG (Joint Photographic Experts Group)
    - Uses lossy compression
    - can tradeoff quality and size
    - Better suited for photographs
GIF Compression

- Uses Lempel-Ziv-Welch compression
  - owned and licensed by Unisys Corporation
  - A variant proposed by Welch of a compression scheme described in a paper:
    - *A universal algorithm for sequential data compression*
    - *Compression of individual sequences via variable rate coding*
        » IEEE Transactions on Information Theory, IT-24(1978) 530-536
    - *A Technique for High Performance Data Compression*
      - Terry A. Welch
        » IEEE Computer, 17(6), June 1984, pp.8-19

LZW

- Many files have certain strings that repeat very often,
  - an example in text files is the string " the ".
  - With the spaces, the string takes 5 bytes, or 40 bits to encode.
- But what if we were to add the whole string to the list of characters after the last one, at 256.
  - Then every time we came across " the ", we could send the code 256 instead of 32,116,104,101,32.
  - This would take 9 bits instead of 40 (since 256 does not fit into 8 bits).
LZW Compression Algorithm

String w = "";
loop
    read a character k
    if w+k exists in the dictionary
        w = w+k
    else
        output the code for w
        add w+k to the dictionary
        w = k
end loop

LZW Compression

- Need never transmit the dictionary. It is implicit in the input stream ([lzw/lzw.html](lzw/lzw.html))

<table>
<thead>
<tr>
<th>iteration</th>
<th>input</th>
<th>w (at end)</th>
<th>added to dict.</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>abab</td>
<td>&quot;&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>b</td>
<td>ab</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>a</td>
<td>ba</td>
<td>b</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>ab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>eof</td>
<td></td>
<td>ab</td>
<td></td>
</tr>
</tbody>
</table>
LZW legal issues

- Patent on LZW owned by Unisys
  - GIF and UNIX compress utility use it
  - Unisys charges a license fee for its use
- Alternatives
  - PNG and pkzip uses compression based on an unencumbered LZ77

JPEG

- A joint ISO/CCITT committee established a standard for
  - the compression of
  - continuous-tone still images,
  - colour and greyscale
JPEG Compression Algorithm

- Transform into a colour space that separates luminance and chrominance
  - can lose a lot more info in chrominance than in luminance and still have the image look ok
  - e.g. YCbCr
- Downsample chrominance by averaging together 2x2 square groups of pixels
  - leave luminance alone
- Group pixels into 8x8 blocks and apply a Discrete Cosine Transform
  - generates a pixel average + change info
  - no savings here, not lossy

JPEG compression

Input 8x8 (x8 bit) Image Matrix (luminance or reduced chrominance components)

```
117 118 120 121 121 121 120 119
118 119 121 122 122 121 120 120
120 120 121 122 121 121 120 120
119 120 120 119 119 119 119 119
119 118 118 117 117 117 117 117
118 118 117 116 115 115 115 116
120 119 117 116 114 115 116 116
121 120 118 117 115 115 116 117
```
JPEG DCT cont'd

Discrete Cosine Transformed Matrix
(input matrix level shifted -128)

DC component (usually encoded as the difference from the previous block)

-75 3 0 -1 0 0 0 0
11 -6 -0 0 0 0 0 0
0 0 0 0 0 0 0 0
-6 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
0 0 0 0 0 0 0 0

JPEG DCT cont'd

Sample quantization table

8, 6, 5, 8, 12, 20, 26, 31,
6 6, 7, 10, 13, 29, 30, 28,
7, 7, 8, 12, 20, 29, 35, 28,
7, 9, 11, 15, 26, 44, 40, 31,
9, 11, 19, 28, 34, 55, 52, 39,
12, 18, 28, 32, 41, 52, 57, 46,
25, 32, 39, 44, 52, 61, 60, 51,
36, 46, 48, 49, 56, 50, 52, 50
JPEG compression cont'd

Quantized rounded-off matrix

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Quantize in the frequency domain
- In each block, divide each of the 64 frequency components by a distinct quantization coefficient and round the result to the nearest integer
- use higher coefficient for higher frequencies
- this is the main lossy step.
- amount of quantization is user controlled

Encode the quantized coefficients
- using lossless Huffman encoding
- zig-zag through frequency space

In "progressive mode", send the lower frequency components first, and follow up with progressively higher frequency components.
JPEG Overhead

- Unlike GIFs, which have an implicit dictionary in their byte stream, JPEGs carry the overhead of transmitting the tables of quantization coefficients (and possibly the Huffman keys as well).
  - a 1x1 pixel white image is 2,128 bytes as a JPEG versus 35 bytes as a GIF
- Therefore JPEG is not suitable for small images, where the overhead would swamp the image size.

The Colour Cube

- Because browsers first emerged when most computers could only handle 256 unique colours, Netscape defined a 216-colour "cube" that it used to display images.
  - (leaves 20 for OS use, and 20 for other apps)
- Whenever a computer is set to 256 colours, browsers will use this set of colours.
  - If an image needs to be displayed that uses different colours, dithering is used
Dithering

Dithering Example
Options

- If the end-user can't figure out how to up the colours past 256, give them crummy images.
- Create line art that uses only the 216 colours defined in the Netscape colour cube
- Reduce the number of colours in a photograph using adaptive palette and histogram-based techniques

Image Size

- If no size is given in the `<img>` tag, image will be displayed at full resolution
  - (e.g., a typical digital camera will give 1024 x 768, filling up the entire screen or more)
- Can specify a size in the image tag
  `<img src="image.gif" height="100" width="100">`
  - Image will be shrunk down (or blown up) by the browser to occupy a 100x100-pixel square space.
- Should specify a size so that the browser can load the rest of the page while downloading the image in parallel.
Choosing an image size

- Should always shrink the image to the size it will be displayed
  - Can do the shrinking in, e.g., Photoshop and get a better quality result than allowing the browser to do it
  - Smaller file size to transmit

Optimizing GIFs

- Because of the use of LZW compression, it is best to use the fewest possible number of colours in a GIF
  - you get longer runs as a result
- Compromise:
  - jaggies versus download time
Anti-Aliasing

- A technique used to eliminate the "jaggies".
  - Will add many extra colours
    - (e.g., if you draw a red triangle against a white background in Photoshop, you get a 54-colour image)
  - Colours used depend upon the foreground and the background colours

The image shows a triangle, anti-aliased against a white background, moved onto a black background.
Optimizing GIFs

- Can usually get a good anti-aliasing result with far fewer colours than the optimal.
- Therefore,
  - Always anti-alias line art
  - but then reduce the colours used by mapping colours to nearest neighbours
  - preferably use the Web colour cube to avoid client-side dithering

Optimizing JPEGs

- Choose the largest quantization coefficients you can get away with and still have the image look good.
  - yes, it's subjective.
- Reducing the number of colours won't help compress JPEGs much
- Blurring (softening) the image reduces to closer to zero the high-frequency components of the DCT, and hence will allow them to compress better (long runs of zeros).
  - Line art does not compress well with JPEG.
Image Inflation

- No matter the compression technology, before being displayed, the browsers will un-compress the images
  - this can take up a lot of browser memory
- avoid very large background images (even if they compress well)
  - use tiling instead

JPEG - 8 - 44k
JPEG - 0 - 20k

GIF - 16 - 28k
GIF versus JPEG (both ~28k)