

Data Compression

(For Images)

Why Use Data Compression?

Images in bitmap format take a lot of memory to store

- e.g. 1024×768 pixels \times 24 bits-per-pixel = 2.4Mbytes
- 16 Mega pixel RAW format from a camera = 20 Mbytes

Terabyte hard drives are now readily available so why do we need to compress?

- How long does a graphics-intensive web page take to download?
- Even over a broadband connection it could be slow
- How many images can your digital camera hold?
- Picture messaging over mobile phones?

CD (650Mb) can only hold less than 10 seconds of uncompressed video (and DVD only a few minutes). We need to make graphical image data as small as possible for many applications

Types of Compression

Pixel packing

RLE (run-length encoding)

Dictionary-based methods

JPEG compression

Factors to look out for:

- *Lossy* or *lossless* compression?
- What sort of data is a method good at compressing?
- What is its compression ratio?

Lossy vs Lossless Compression

Lossless compression implies that one can re-create the original data precisely.

- Usable anywhere,
- Critical for e.g. documents, binary programs, etc.
- Usually requires significant processing to re-create original

Lossy compression implies that one cannot recreate the precise original

- But one can re-create something almost as good
- Useful for images, sounds, where what matters is the percept, not precise bit-for-bit re-creation.
- Some processing required but less than lossless approach

Pixel Packing

Not a standard “data compression technique” but nevertheless a way of not wasting space in pixel data, e.g.

- suppose pixels can take grey values from 0-15
- each pixel requires half a byte
- but computers prefer to deal with bytes
- two pixels per byte doesn't waste space

Pixel packing is simply ensuring no bits are wasted in the pixel data (lossless if assumption true)

Run Length Encoding (RLE)

Basic idea is:

- AAAAAAAAAAAAAAAAAA would encode as 15A
- AAAAAAbbbXXXXXt would encode as 6A3b5X1t

So this compression method is good for compressing large expanses of the same colour - or is it?



RLE Compression Ratio

Of the 110 pixels in the 10×11 pixels sample taken from the image, there are 59 different colours altogether. RLE compression ratios not good in general, because there are rarely repeat runs of pixels:

Full image: 371×247 bitmap

- 275Kb raw data
 - $(274911 = 371 \times 247 \times 3)$ bytes
- 91K RLE encoded



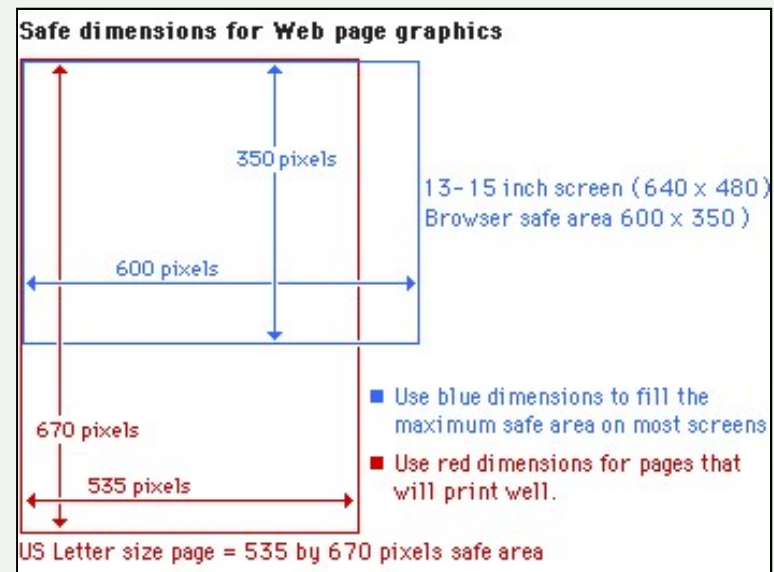
A compression ratio approx 3:1 in this case

RLE Compression Ratio

Another example, with a diagram this time:

Full image: 350×264 bitmap

- 277Kb raw data
- $350 \times 264 \times 3$ bytes
- 46.5K RLE encoded



A compression ratio of approx 6:1 in this case

Dictionary Methods

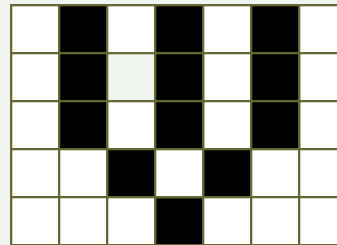
A common way to compress data (pixels or characters) is to use a *dictionary*

- The dictionary contains strings of bytes
 - particular pixel patterns
 - not limited to patterns of one colour, as with RLE

Data is encoded by replacing each data string that has an entry in the dictionary with its index number in the dictionary

- Shorter to write an index number than a whole string
- Dictionary may be particular to the image, or may be “standard” for particular image types

Patterns of Pixels



RLE will produce poor results as runs of pixels with same colour are very short. However, there are repeating patterns with two colours that could be included in a dictionary.

- Trade off between pattern size and likelihood of being in dictionary.



Huffman & CCITT Compression

Developed for fax machines and document scanners

- Uses a predefined dictionary of commonly occurring byte patterns from B&W documents containing large amounts of text in a variety of languages and typical examples of line art
- Commonly occurring patterns are given low indices (coded using short codes) in the dictionary
- Data is encoded by replacing each image string that occurs in the dictionary with its index number
- Dictionary is not part of the compressed file.

The Lempel-Ziv-Welch Algorithm

The **Lempel-Ziv-Welch method** is another such dictionary algorithm, in which the dictionary is constructed as the encoding (compression) progresses. **LZW** starts with a **dictionary** of byte strings:

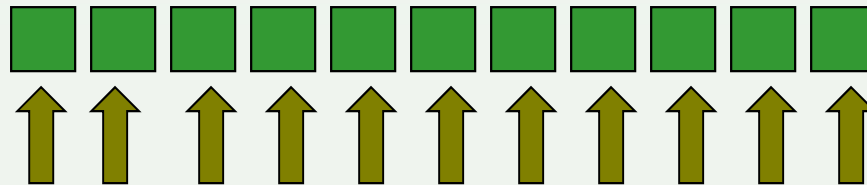
- Entries 0-255 refer to those individual bytes
- Entries 256 onwards will be defined as the algorithm progresses
- Each time a new code is generated it means a new string of bytes has been found.
- New strings are generated by appending a byte **c** to the end of an existing string **w**
- Single pass: see algorithm on next slide

The LZW Algorithm (2)

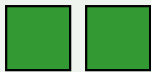


```
set w = "";  
read a character c;  
while (not EOF)  
    if w+c exists in the dictionary  
    {  
        w = w+c;  
    }  
    else  
    {  
        output the code for w;  
        add w+c to the dictionary;  
        w = c;  
    }  
    read a character c;  
endwhile  
output code for w;
```



A Pixel Example

Pixel data to encode:



Dictionary:

#256 
#257 
#258 

Output:  #256 #257 #258 

When Is LZW Useful?

Good for encoding pixel data with a limited palette, and/or repetitive data

- line drawings
- diagrams
- plain text on a plain background

Not good for photographic images

- large colour range and complex features results in few repeating patterns to include in a dictionary

Lossless and fast

JPEG

Joint Photographic Experts Group

Designed to compress photographs

- colour or greyscale
- good at compressing “real” scenes
- not good for line drawings, diagrams, lettering, cartoons

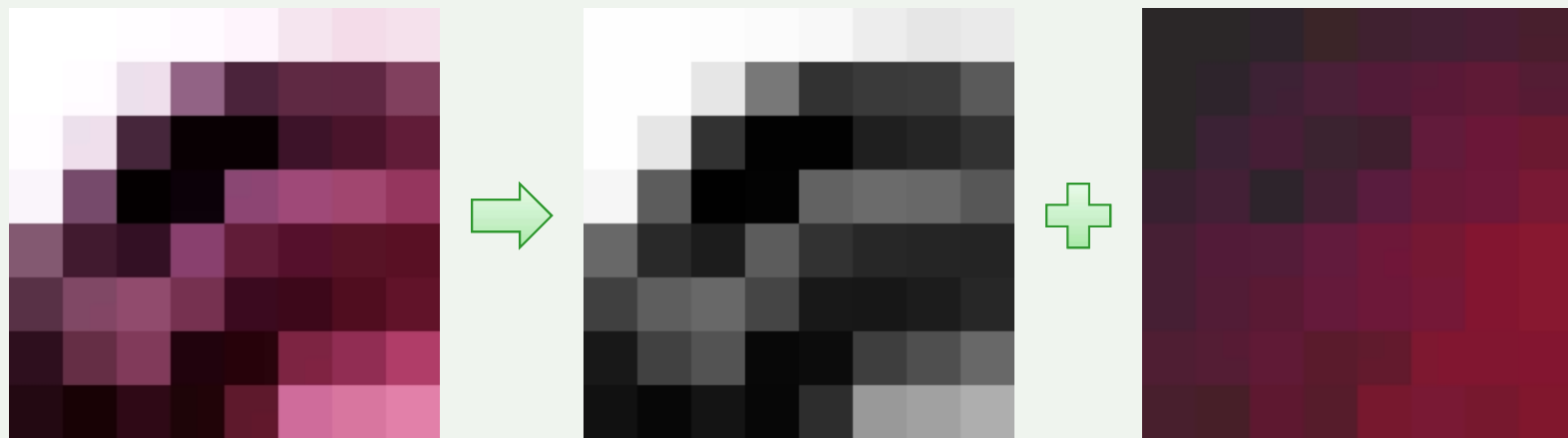
Designed for human viewing, exploits our inability to see a full range of colours

- **Lossy algorithm**
- Not good for computer analysis of data
 - e.g. medical imaging
 - may throw away vital data

JPEG: How it works

Step 1: If a colour image, transform the image into a suitable colour space, with a γ component

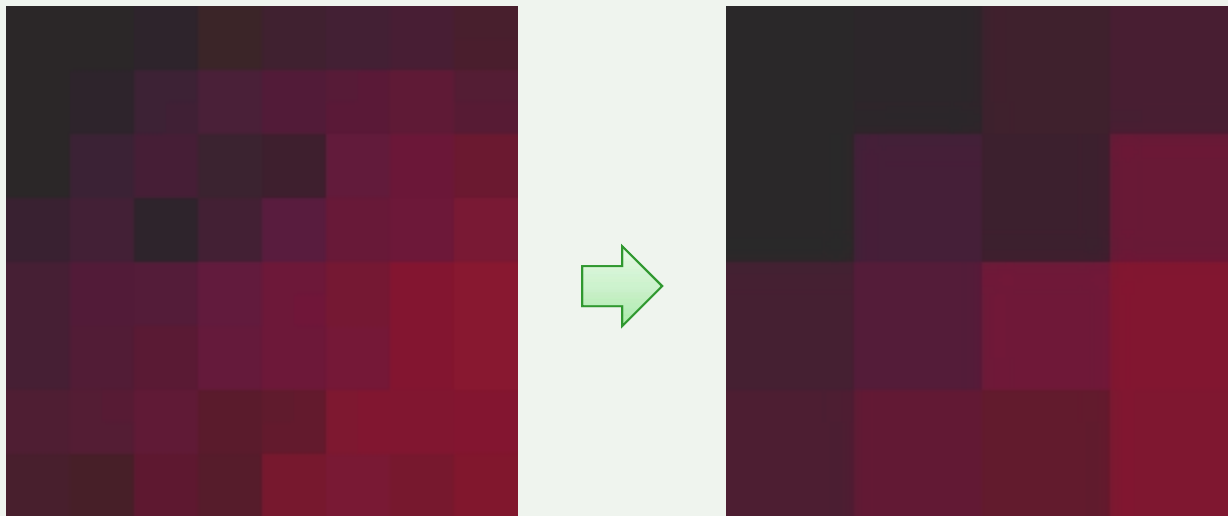
- e.g. from RGB to HSV / XYZ / Lab / YCbCr
- not necessary for greyscale images



JPEG: How it works (2)

Step 2 (optional):

Leave the γ data alone, but “downsample” both lots of colour data. Reduce resolution by 2, in the y and x directions



JPEG: How it works (3)

Step 3:

- Divide the image (both γ and colour data) into 8×8 pixel blocks



Step 4:

- For each block, perform a DCT (Discrete Cosine Transform) on the data
- This transforms the 64 (integer) values to a different set of 64 (non-integer) values
 - amplitudes of spatial frequencies

JPEG: How it works (4)

Step 5:

- Use quantization on these 64 values
 - divide by a specially chosen number, and round to the nearest integer
 - e.g. amplitudes of lowest frequencies may range from 0-255
 - slightly higher frequencies have amplitudes divisible by 4
 - highest frequencies may only have amplitudes of 0 or 128

Step 6:

- Store these numbers in a space-efficient way
 - RLE and Huffman coding
 - long strings of 0 coefficients

JPEG Compression (contd)

By choosing a different number in step 5 (the *quantization coefficient*), we get different amounts of compression

- Trade-off of quality versus size of compressed data

Decoding a JPEG is the reverse process:

- unpack the efficiently-stored data
- do a reverse DCT on both the colour data and the γ to get the 8×8 pixel blocks
- combine the colour data with the γ and display the result
- BUT no recovery from the *quantization* processes

JPEG Compression - Original



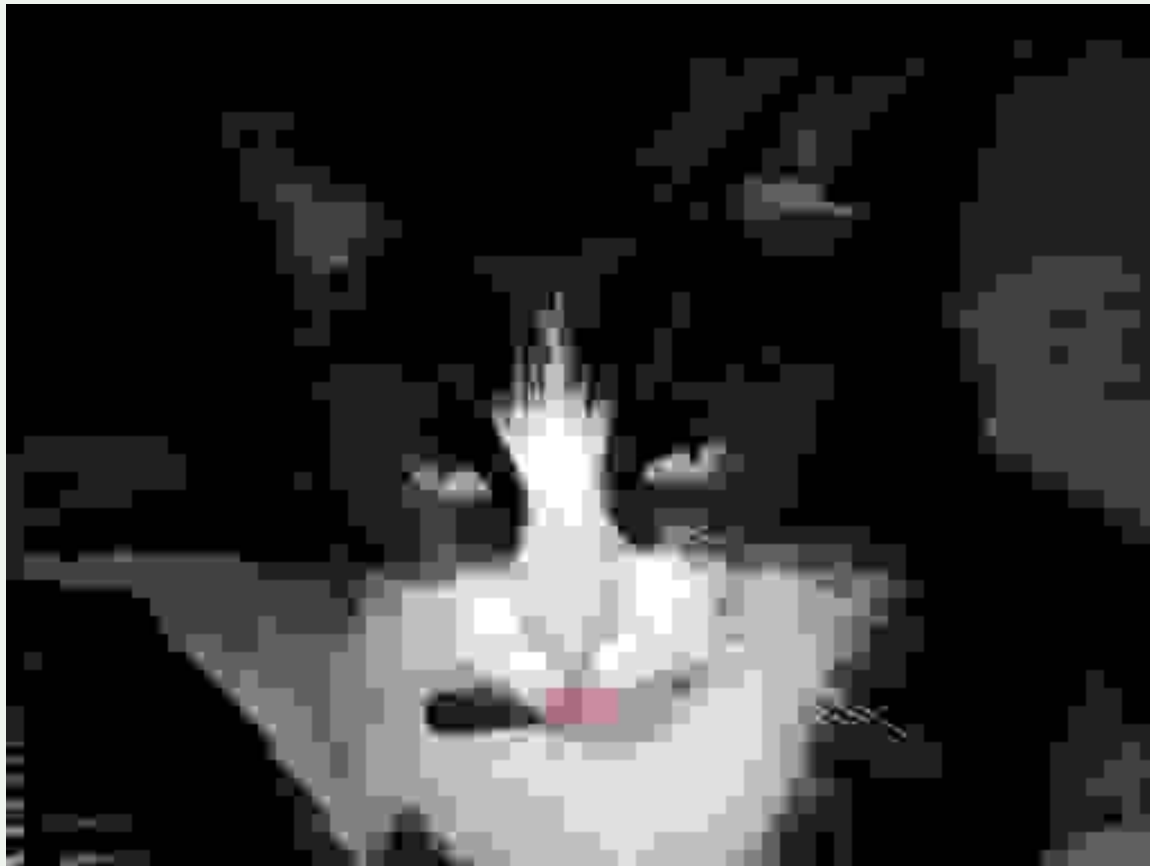
JPEG Compression

Medium Compression



JPEG Compression

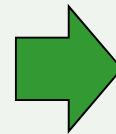
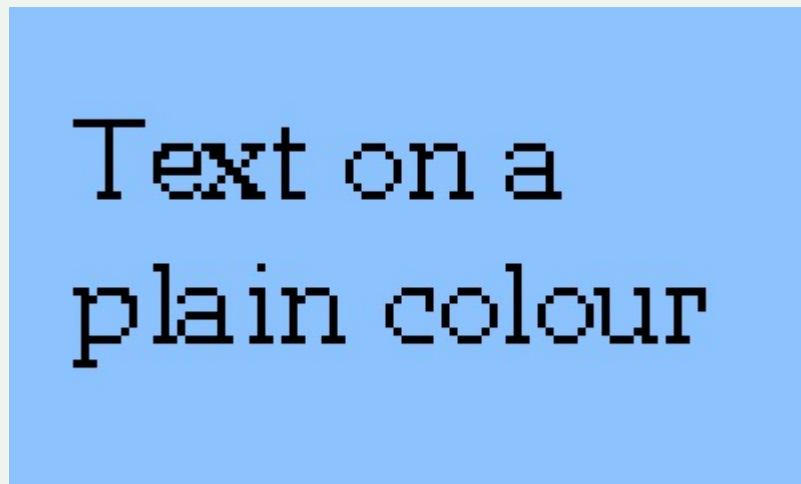
High Compression



JPEG & Text

Text to background transition is a high spatial frequency

- JPEG attempts to smooth this with poor results...



JPEG Compression

How good is it?

- For full-colour images, the best-known lossless compression gives about 2:1 compression
- For reasonable quality, compression ratios of 10:1 or 20:1 are quite feasible for JPEGs
- For poor quality images (thumbnails?), 100:1 possible
- Repeating the process with subsequent re-encoding loses more information

How Do We Compress Movies?

Compress individual frames using any of the techniques mentioned already. High, *lossy* compression is acceptable as the quality of individual frames can be lower than for still images as our perception is dominated by motion.

Make use of limited changes between frames

- key frames
- difference frames
- *temporal compression*

More on this in a later lecture

End of Lecture

The next lecture will look at the multitude of different graphic file formats.