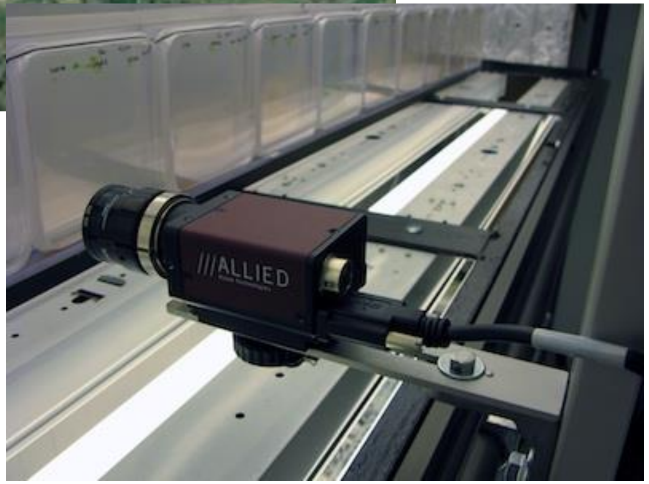
# COMP2005

Introduction to Image Compression

# Image Compression

- Individual images can be large
- Images are easy to acquire, collections increase rapidly
- In some applications, images are gathered automatically
- Luckily, image data is redundant in several ways
  - Coding redundancy
  - Spatial redundancy
  - Psychovisual redundancy



# Coding Redundancy

- The grey level histogram of an image gives the probability (frequency) of occurrence of grey level $r_k$

$$p(r_k) = \frac{n_k}{n} \qquad k = 0,2,\ldots,L-1$$

- If the number of bits used to represent each value of $r_k$ is $l(r_k)$, the **average number of bits required to represent a pixel is**

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k)p(r_k)$$

- To code an MxN image requires $MNL_{avg}$ bits

# Coding Redundancy

- If an m-bit natural binary code is used to represent grey level then
  - all pixels take the same amount of space,
  - $P(r_k)$ values sum to 1, so

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k)p(r_k) = \sum_{k=0}^{L-1} mp(r_k) = m$$

  - and an image occupies MNm bits
- But some pixel values are more common than others…..

# Variable Length Encoding

- Assigning fewer bits to the more probable grey levels than to less probable ones can achieve data compression, e.g:

| $r_k$ | $p_r(r_k)$ | **Code 1** | $l_1(r_k)$ | **Code 2** | $l_2(r_k)$ |
|---|---|---|---|---|---|
| $r_{87} = 87$ | 0.25 | 01010111 | 8 | 01 | 2 |
| $r_{128} = 128$ | 0.47 | 10000000 | 8 | 1 | 1 |
| $r_{186} = 186$ | 0.25 | 11000100 | 8 | 000 | 3 |
| $r_{255} = 255$ | 0.03 | 11111111 | 8 | 001 | 3 |
| $r_k$ for $k \neq 87, 128, 186, 255$ | 0 | — | 8 | — | 0 |

- Build a codebook, replace ‘true’ pixel values with code
- Lossless: the process can be reversed by inverting the codebook

# Spatial Redundancy

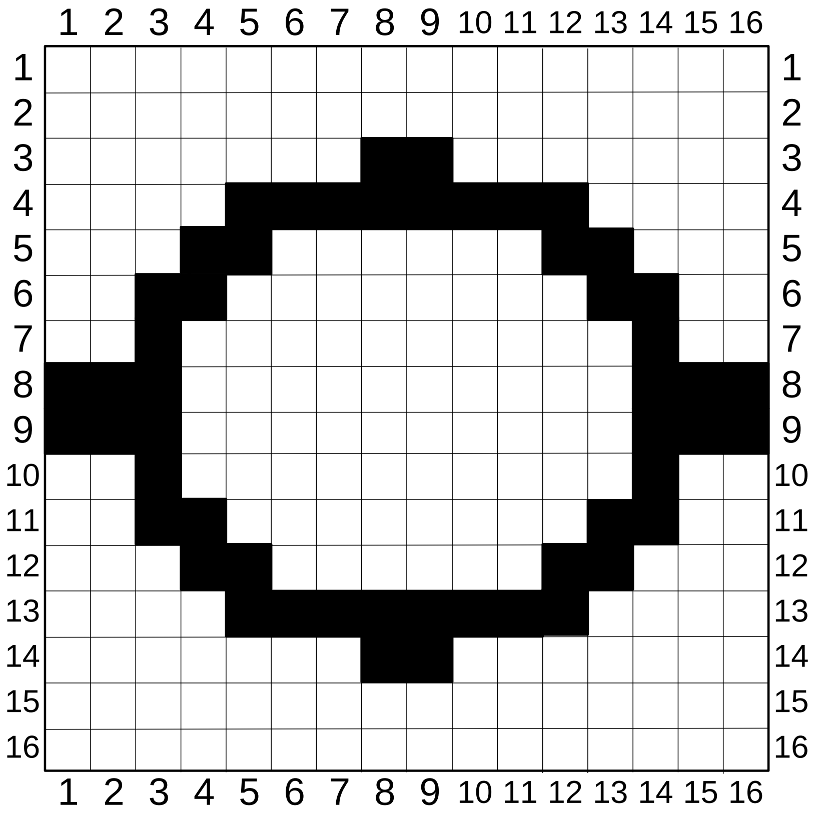
- Sometimes called *Interpixel Redundancy*
- Neighbouring pixels often have similar values
- Compression based on spatial redundancy involves some element of pixel grouping, or transformation
- Simplest is Run-Length Encoding
  - maps the pixels along each scan line into a sequence of pairs $(g_1, r_1)$, $(g_2, r_2)$, …,
  - where $g_i$ is the ith grey level, $r_i$ is the run length of ith run

# A Binary Example

Row 1: (0, 16)
Row 2: (0, 16)
Row 3: (0, 7) (1, 2) (0, 7)
Row 4: (0, 4), (1, 8) (0, 4)
Row 5: (0, 3) (1, 2) (0, 6) (1, 3) (0, 2)
Row 6: (0,2) (1, 2) (0,8) (1, 2) (0, 2)
Row 7: (0, 2) (1,1) (0, 10) (1,1) (0, 2)
Row 8: (1, 3) (0, 10) (1,3)
Row 9: (1, 3) (0, 10) (1, 3)
Row 10: (0,2) (1, 1) (0,10) (1, 1) (0, 2)
Row 11: (0, 2) (1, 2) (0, 8) (1, 2) (0, 2)
Row 12: (0, 3) (1, 2) (0, 6) (1, 3) (0, 2)
Row 13: (0, 4) (1,8) (0, 4)
Row 14: (0, 7) (1, 2) (0, 7)
Row 15: (0, 16)
Row 16: (0, 16)

encode ←

decode →



# Psychovisual Redundancy

- Some grey level and colour differences are imperceptible; goal is to compress without noticeable change to the image

256 gray levels

16 gray levels

16 gray levels

A simple method: add a small random number to each pixel before quantization

# Evaluating Compression

- Fidelity Criteria: success is judged by comparing original and compressed versions
- Some measures are objective, e.g. root mean square error ($e_{rms}$) and signal to noise ratio (SNR)
- Let f(x,y) be the input image, f'(x,y) be reconstructed input image from compressed bit stream, then

$$e_{rms} = \left( \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left( f'(x,y) - f(x,y) \right)^2 \right)^{1/2} \qquad SNR = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left( f'(x,y) \right)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left( f'(x,y) - f(x,y) \right)^2}$$

# Fidelity Criteria


$e_{rms} = 6.93$ $e_{rms} = 6.78$

$SNR_{rm} = 10.25$ $SNR_{rm} = 10.39$

# Fidelity Criteria

- $e_{rms}$ and SNR are convenient objective measures
- Most decompressed images are viewed by human beings
- Subjective evaluation of compressed image quality by human observers are often more appropriate

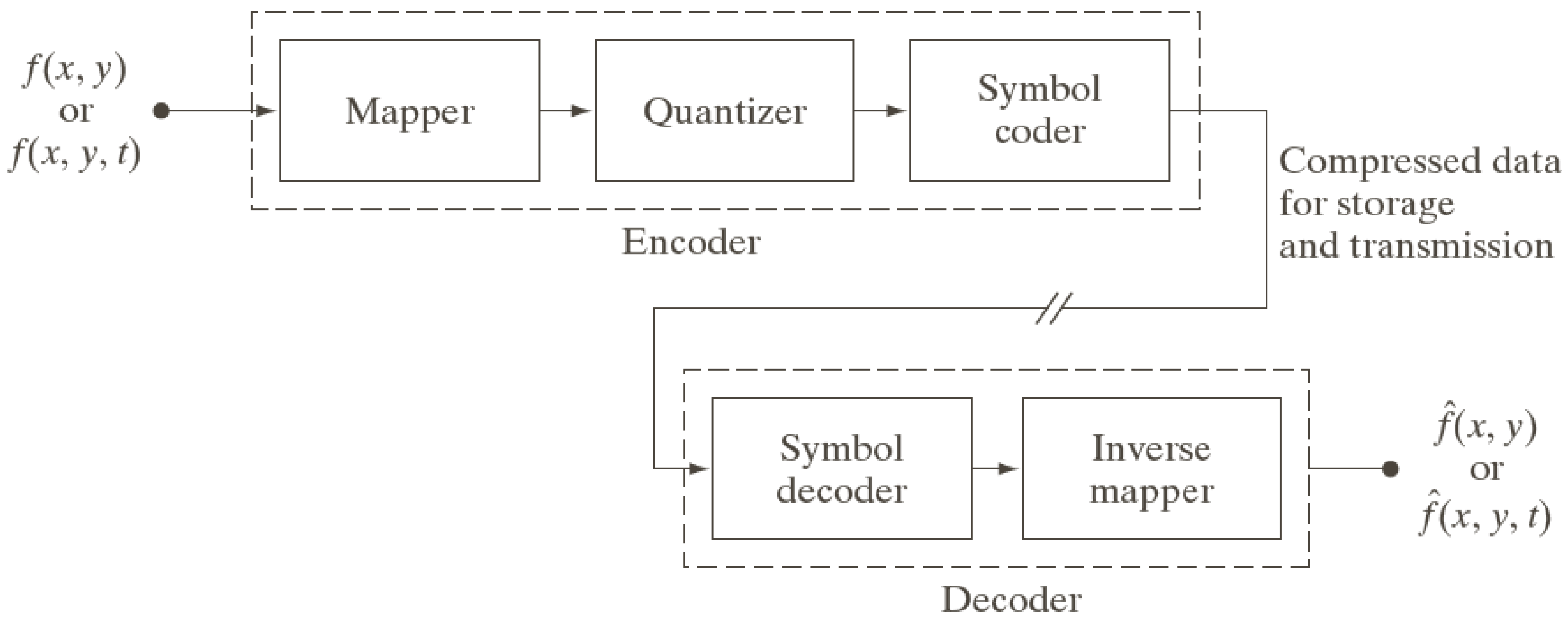
| Value | Rating | Description |
|---|---|---|
| 1 | Excellent | An image of extremely high quality, as good as you could desire. |
| 2 | Fine | An image of high quality, providing enjoyable viewing. Interference is not objectionable. |
| 3 | Passable | An image of acceptable quality. Interference is not objectionable. |
| 4 | Marginal | An image of poor quality; you wish you could improve it. Interference is somewhat objectionable. |
| 5 | Inferior | A very poor image, but you could watch it. Objectionable interference is definitely present. |
| 6 | Unusable | An image so bad that you could not watch it. |

Rating scale of the Television Allocations Study Organization. (Frendendall and Behrend.)

# Image Compression Systems

- Mapper (spatial redundancy)
  - transforms input data in a way that facilitates reduction of interpixel redundancies
  - reversible
- Quantiser (psychvisual redundancy)
  - transforms input data in a way that facilitates reduction of psychovisual redundancies
  - not reversible
- Symbol coder (coding redundancy)
  - assigns the shortest code to the most frequently occurring output values
  - reversible

# Image Compression Systems




Functional block diagram of a general image compression system

# Image Compression: Huffman Coding

# Exploiting Coding Redundancy

- These methods are derived from information theory: **try to maintain a high level of information in compressed images**
- Not limited to images, are applicable to any digital information.
  - speak of ***symbols* instead of *pixel values*** and ***sources* instead of *images***
  - exploit nonuniform probabilities of symbols (nonuniform histograms) and use a variable-length code.
- Evaluation requires a measure of the information content of a source: *Entropy*

# Entropy

- The idea: associate information with probability
- A random event *E* with probability *P*(*E*) contains:

$$I(E) = log(\frac{1}{P(E)}) = -log(P(E))$$

units of information

- Suppose that grey level values are generated by a random variable, then $r_k$ contains:

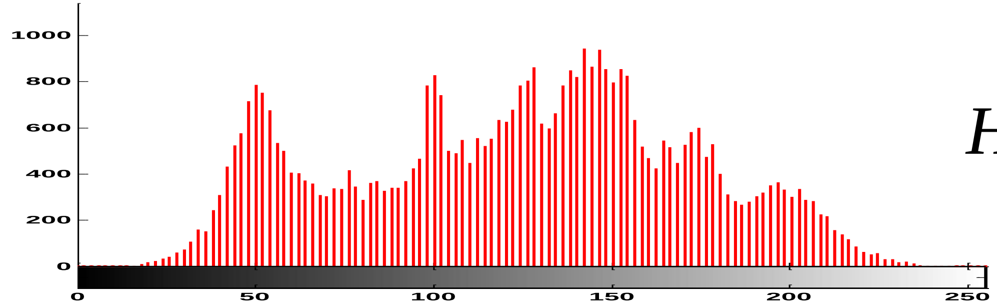
$$I(r_k) = -log(P(r_k))$$

Note: I(E)=0 when P(E)=1

units of information

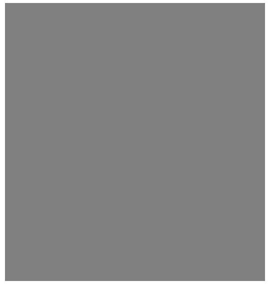
# Entropy

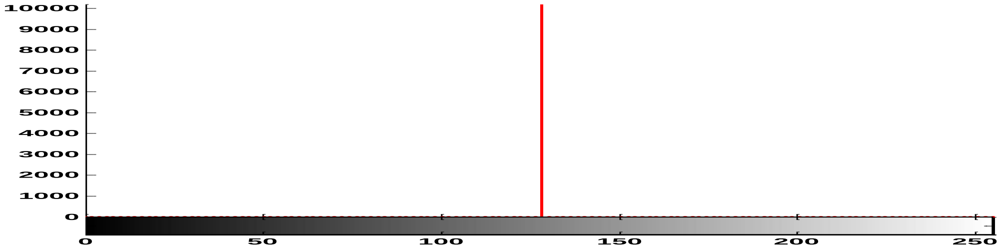
- Entropy is the average information content of an image, a measure of histogram dispersion




$$H = -\sum_{k=0}^{L-1} p(r_k) \log_2 p(r_k)$$

entropy=7.4635






entropy=0

- Can’t compress to less than H bits/pixel without losing information

# Huffman Coding

- Compute probabilities of each symbol by histogramming the source
- Process probabilities to pre-compute codebook: code(i)
  - codebook is static (fixed)
- Encode source symbol-by-symbol:
  symbol(i) -> code(i)
- Transmit coded signal and codebook
- The need to pre-process (histogram) the source before encoding begins is a disadvantage

# Huffman Coding

- Builds a binary tree in which symbols to be coded are nodes
- The algorithm:
  - Create a list of nodes, one per for symbol, sorted in order of symbol frequency (or probability)
  - REPEAT (until only one node left)
    - Pick the two nodes with the lowest frequencies/probabilities and create a parent of them
    - **Randomly** assign the codes 0,1 to the two new branches of the tree and delete the children from the list
    - Assign the sum of the children’s probabilities to their parent and insert it in the list
- Path from root to node gives code for corresponding symbol

# Huffman Coding

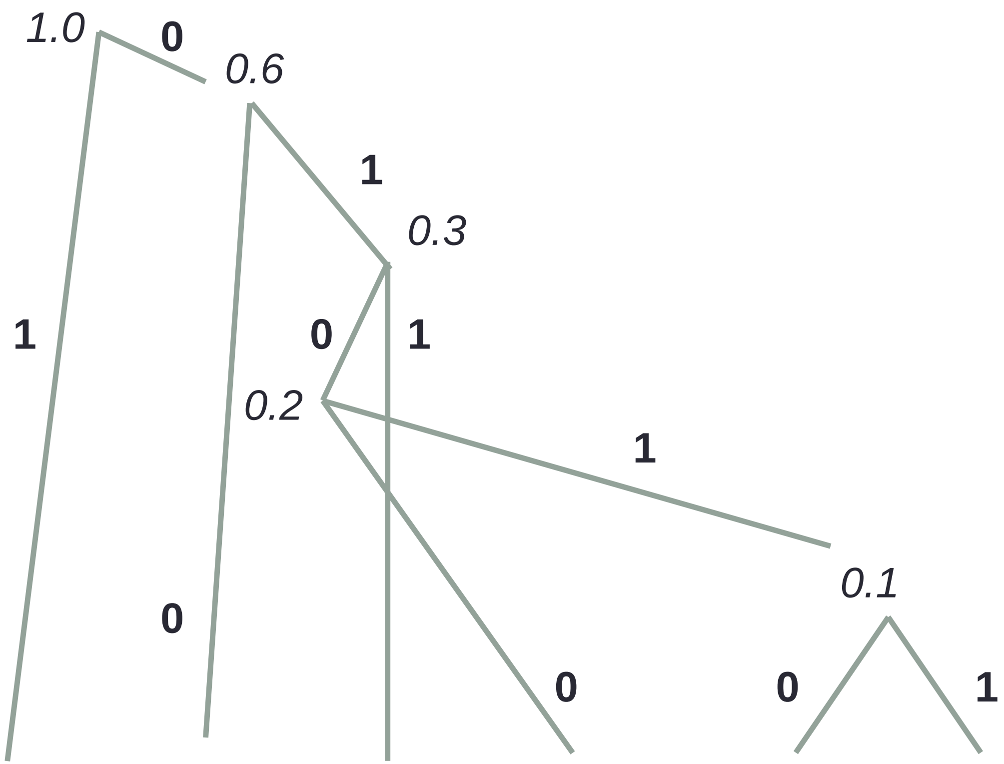
| Original source | | Source reduction | | | |
|---|---|---|---|---|---|
| Symbol | Probability | 1 | 2 | 3 | 4 |
| $a_2$ | 0.4 | 0.4 | 0.4 | 0.4 | 0.6 |
| $a_6$ | 0.3 | 0.3 | 0.3 | 0.3 | 0.4 |
| $a_1$ | 0.1 | 0.1 | 0.2 | 0.3 | |
| $a_4$ | 0.1 | 0.1 | 0.1 | | |
| $a_3$ | 0.06 | 0.1 | | | |
| $a_5$ | 0.04 | | | | |

- Create parent node of $a_3$ and $a_5$

- Create parent of new node and $a_4$

- New parent is higher up list (0.2) than $a_1$ (0.1)

# Huffman Coding




| | | | | | | |
|---|---|---|---|---|---|---|
| $Prob_i$ | 0.4 | 0.3 | 0.1 | 0.1 | 0.06 | 0.04 |
| $Symbol_i$ | $a_2$ | $a_6$ | $a_1$ | $a_4$ | $a_3$ | $a_5$ |
| $Code_i$ | **1** | **00** | **011** | **0100** | **01010** | **01011** |

| Original source | | | Source reduction | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Symbol | Probability | Code | 1 | | 2 | | 3 | | 4 | |
| $a_2$ | 0.4 | 1 | 0.4 | 1 | 0.4 | 1 | 0.4 | 1 | 0.6 | 0 |
| $a_6$ | 0.3 | 00 | 0.3 | 00 | 0.3 | 00 | 0.3 | 00 | 0.4 | 1 |
| $a_1$ | 0.1 | 011 | 0.1 | 011 | 0.2 | 010 | 0.3 | 01 | | |
| $a_4$ | 0.1 | 0100 | 0.1 | 0100 | 0.1 | 011 | | | | |
| $a_3$ | 0.06 | 01010 | 0.1 | 0101 | | | | | | |
| $a_5$ | 0.04 | 01011 | | | | | | | | |

# Huffman Coding

- The algorithm systematically places nodes representing high probability symbols further up the tree: their paths (and so codes) are shorter
- No code is a prefix to any other – don’t need to mark boundaries between codes
  - e.g. 01101010 must be $a_1a_3$
- In this example
  - Average length of the code is 2.2.bits/symbol
  - The entropy of the source is 2.14 bits/symbol
- In general
  - Break image into small (e.g. 8 x 8) blocks
  - Each block is a symbol to be encoded

# A Huffman Code Example

From a past exam paper:

*An image has the following normalized histogram. Derive a Huffman code for each pixel value, <u>showing how you obtained your code</u>*

10 mins

| Pixel Value | Normalised Frequency |
|---|---|
| 0 | 0.1 |
| 1 | 0.1 |
| 2 | 0.15 |
| 3 | 0.35 |
| 4 | 0.2 |
| 5 | 0 |
| 6 | 0.05 |
| 7 | 0.05 |

# A Huffman Code Example

| Pixel Value | Normalised Frequency |
|---|---|
| 0 | 0.1 |
| 1 | 0.1 |
| 2 | 0.15 |
| 3 | 0.35 |
| 4 | 0.2 |
| 5 | 0 |
| 6 | 0.05 |
| 7 | 0.05 |

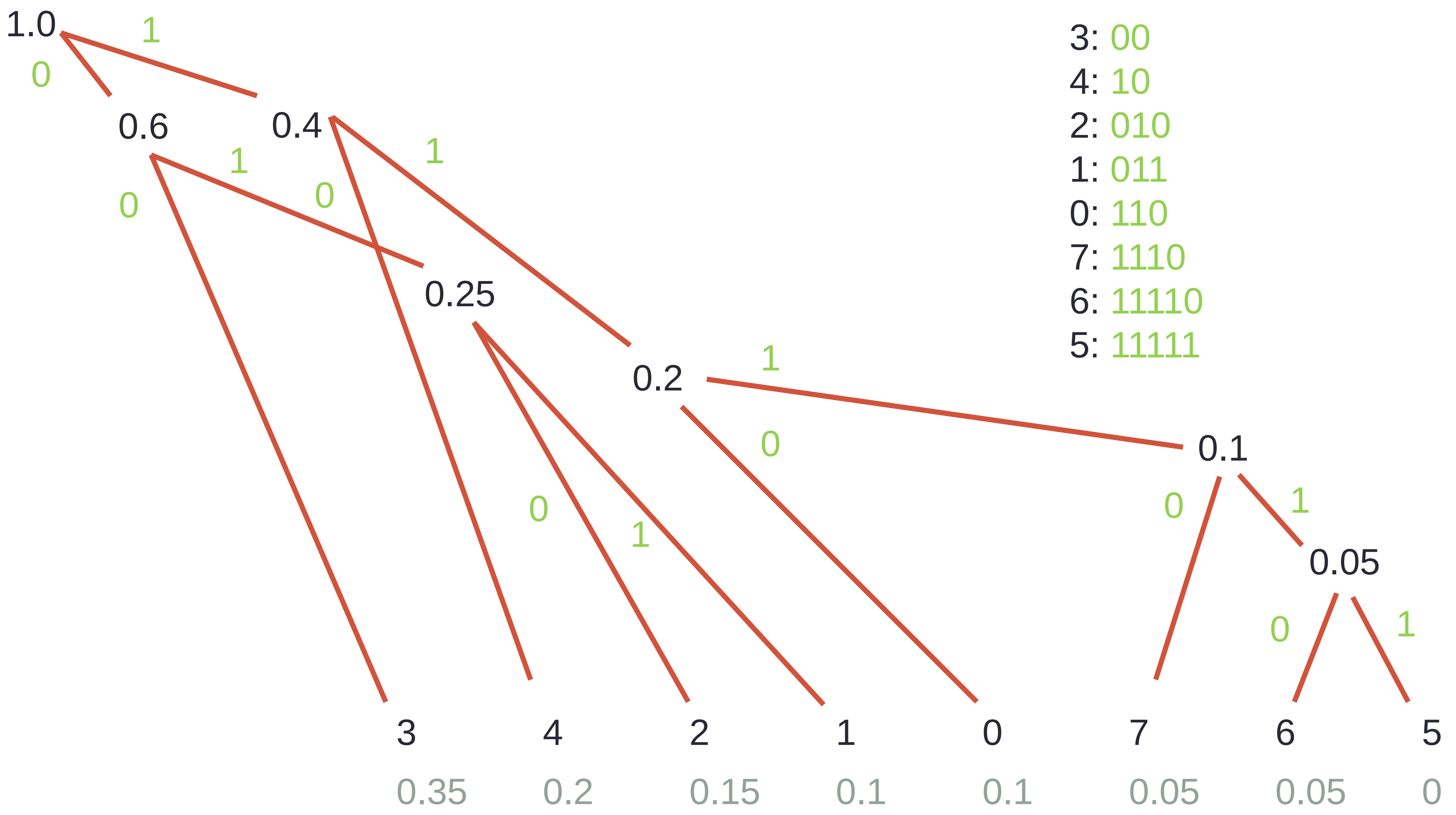
Sort →

| Pixel Value | Normalised Frequency |
|---|---|
| 3 | 0.35 |
| 4 | 0.2 |
| 2 | 0.15 |
| 1 | 0.1 |
| 0 | 0.1 |
| 7 | 0.05 |
| 6 | 0.05 |
| 5 | 0 |

# A Huffman Code Example: Table View

| Pixel Value | Normalised Frequency | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 3 | 0.35 | 0.35 | 0.35 | 0.35 | 0.35 | 0.4 | 0.6 | 1.0 |
| 4 | 0.2 | 0.2 | 0.2 | 0.2 | 0.25 | 0.35 | 0.4 | |
| 2 | 0.15 | 0.15 | 0.15 | 0.2 | 0.2 | 0.25 | | |
| 1 | 0.1 | 0.1 | 0.1 | 0.15 | 0.2 | | | |
| 0 | 0.1 | 0.1 | 0.1 | 0.1 | | | | |
| 7 | 0.05 | 0.05 | 0.1 | | | | | |
| 6 | 0.05 | 0.05 | | | | | | |
| 5 | 0 | | | | | | | |

# A Huffman Code Example: Tree View




3: 00
4: 10
2: 010
1: 011
0: 110
7: 1110
6: 11110
5: 11111

# A Huffman Code Example: The Code

| | | Code length | Normalised Freq, |
|---|---|---|---|
| 3: 00 | 0: 110 | 3 | 0.1 |
| 4: 10 | 1: 011 | 3 | 0.1 |
| 2: 010 | 2: 010 | 3 | 0.15 |
| 1: 011 → | 3: 00 | 2 | 0.35 |
| 0: 110 | 4: 10 | 2 | 0.2 |
| 7: 1110 | 5: 11111 | 5 | 0 |
| 6: 11110 | 6: 11110 | 5 | 0.05 |
| 5: 11111 | 7: 1110 | 4 | 0.05 |

Mean bits/pixel $L_{avg}$ = 0.3 +0.3 +0.45 +0.70 + 0.4 + 0 + 0.25 + 0.2 = 2.6

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k)p(r_k)$$

Without compression $L_{avg}$ = 3

Compression ratio = 2.6/3 = 0.86

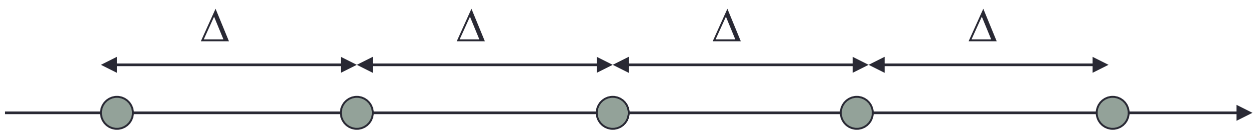
# Image Compression: GIF

# Using Psychovisual Redundancy

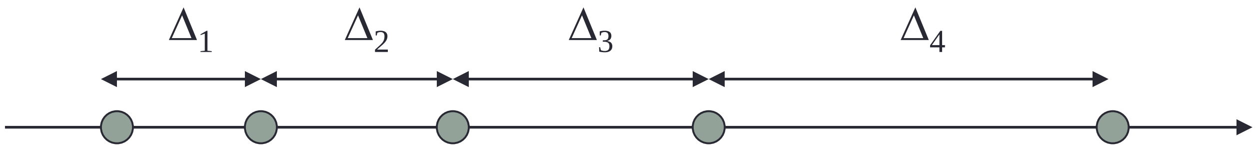
- Quantisation is widely used
  - Represent areas of grey level/colour space with fewer bits
  - Lossy: cannot be inverted
  - Find the best tradeoff between

    maximal compression ←→ minimal distortion
- Scalar Quantisation (i.e. quantising scalar values)

Uniform scalar quantization:




Non-uniform scalar quantization:



# Vector Quantisation

- Palettised images (gif)
  - Map vector values (R,G,B) onto scalar values
  - Multiple vectors map to each scalar

Vector quantisation

True colour R,G,B
8 bits each
1677216 possible colours

gif
8 bits per pixel
256 possible colours

# Paletised Images

(r, g, b)

For each pixel in the original image

Find the closest colour in the Colour Table

| | | |
|---|---|---|
| $r_0$ | $g_0$ | $b_0$ |
| $r_1$ | $g_1$ | $b_1$ |
| $\vdots$ | | |
| $r_{255}$ | $g_{255}$ | $b_{255}$ |

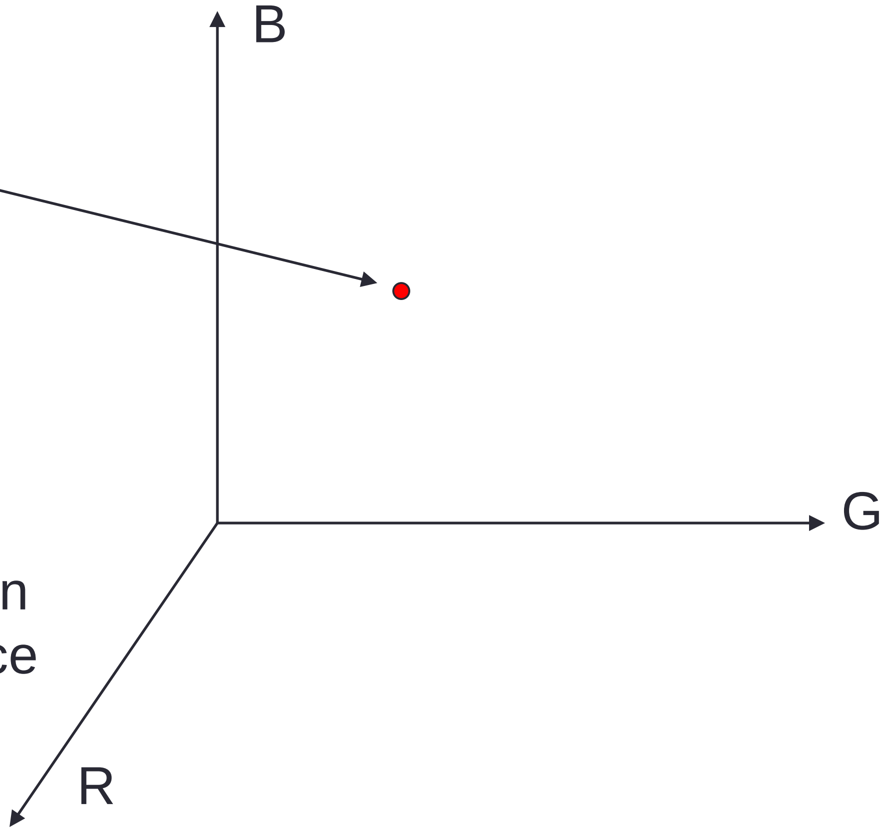
**Colour Table**

Record the index of that colour (for storage or transmission)

To reconstruct the image, place the indexed colour from the Colour Table at the corresponding spatial location

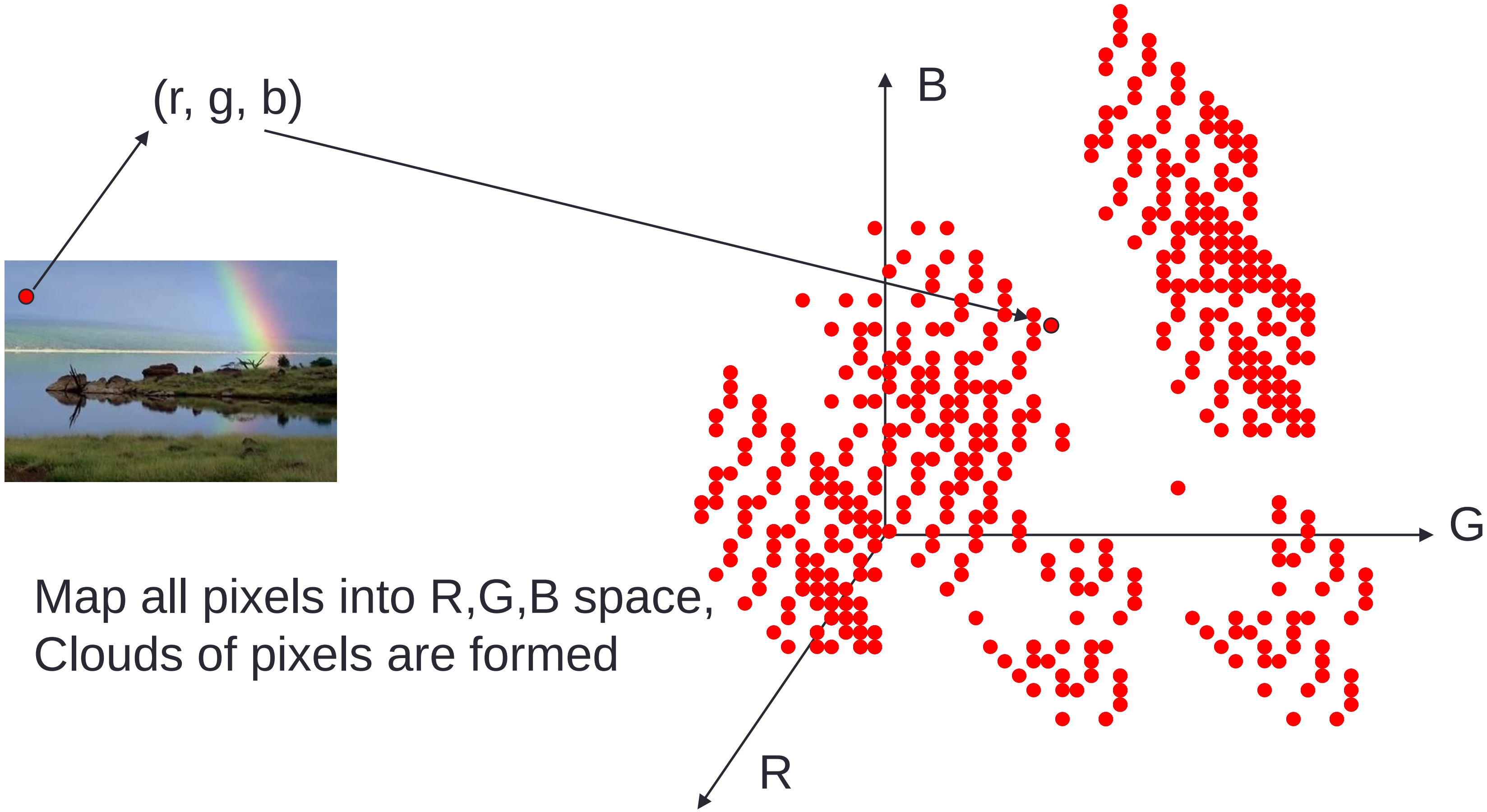
# Building a Palette

(r, g, b)



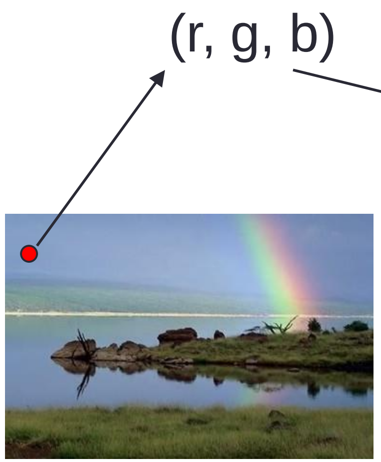

A pixel corresponds to a point in the 3 dimensional R, G, B space

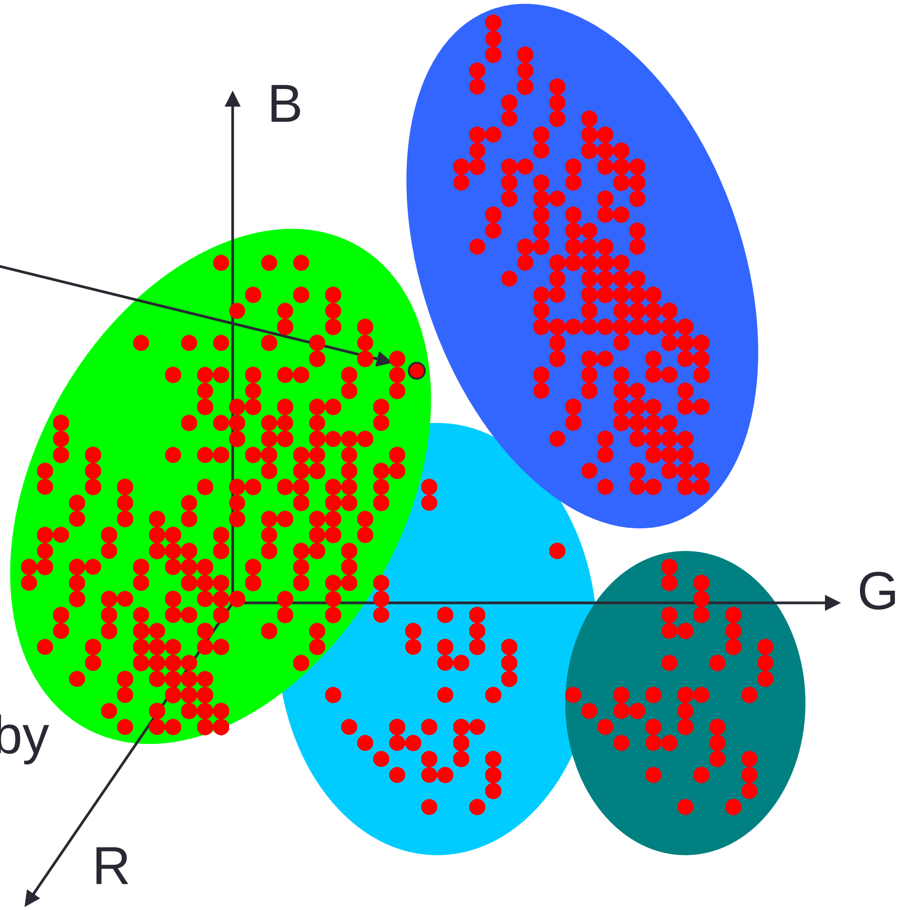
# Building a Palette




Map all pixels into R,G,B space,
Clouds of pixels are formed

# Building a Palette

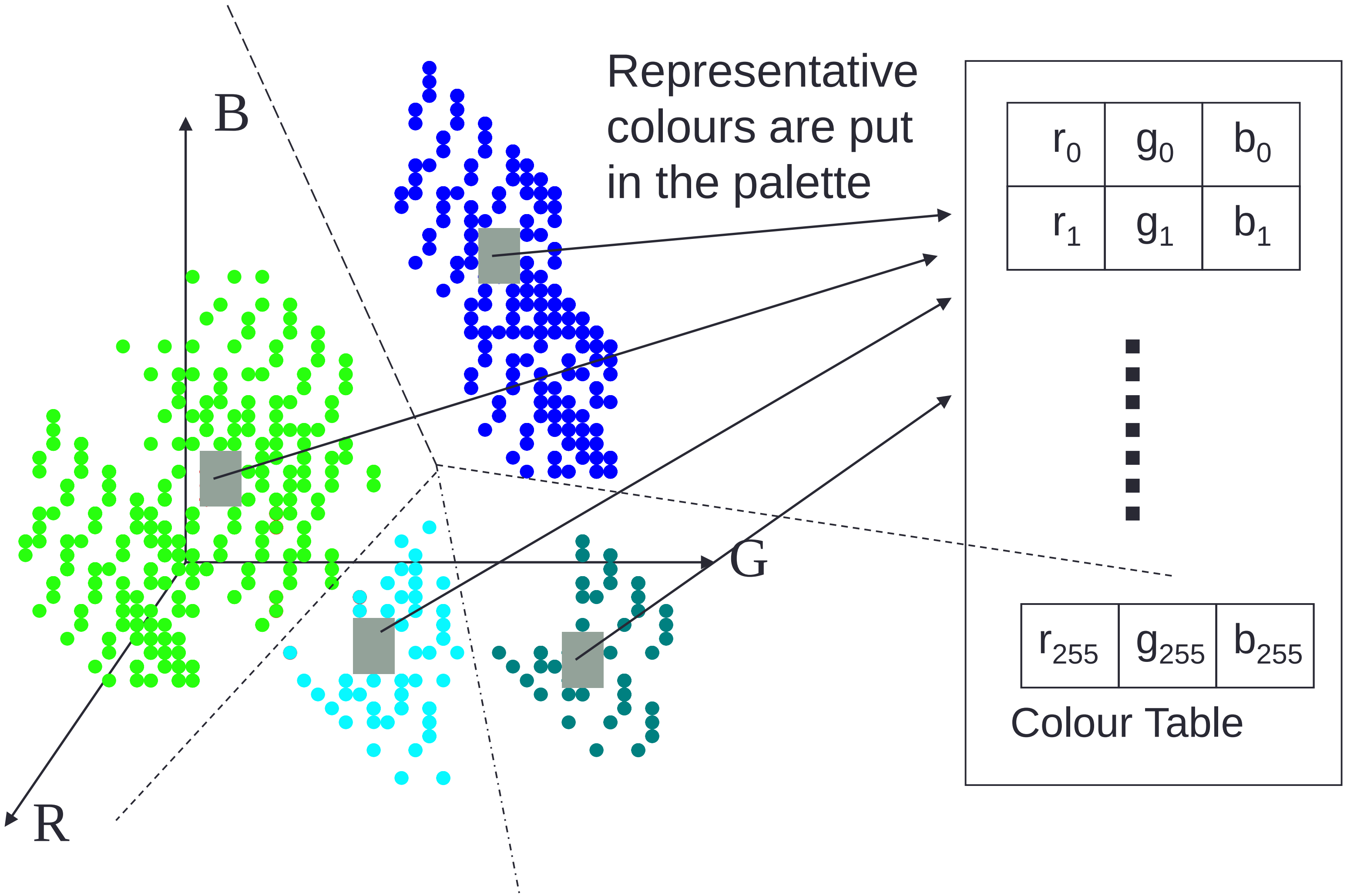



Group pixels that are close to each other, and replace them by one single colour

Close to each other means of a similar colour

# Building a Palette



# Building a Palette

- Many clustering algorithms exist
  - supervised
  - unsupervised
- We know how many clusters we need: one per palette entry
- We need clusters that are spread across the colour space
- A supervised method….

- K-Means Clustering
  - Start with estimates of the mean of each cluster

    $$\mu_1, \mu_2, \ldots, \mu_k$$

  - Assign each point, *p*, to the cluster where

    $$|p - \mu_i|$$

    is smallest
  - Recompute the means
  - Repeat until no changes are made to the clusters

# COMP2005

Image Compression: JPEG

# Exploiting Spatial Redundancy

- Run-length encoding needs adjacent pixels to be equal
- Pixels are more often highly correlated (dependent)
  - Not equal, but can predict the image pixels to be coded from those already coded
- Each pixel value (except at the boundaries) is predicted based on its neighbors (e.g., linear combination) to get a predicted image.
- The difference between the original and predicted images yields a differential or residual image <u>with a reduced set of values</u>.
- The differential image is encoded using Huffman coding, or similar.

# Differential Pulse-code Modulation

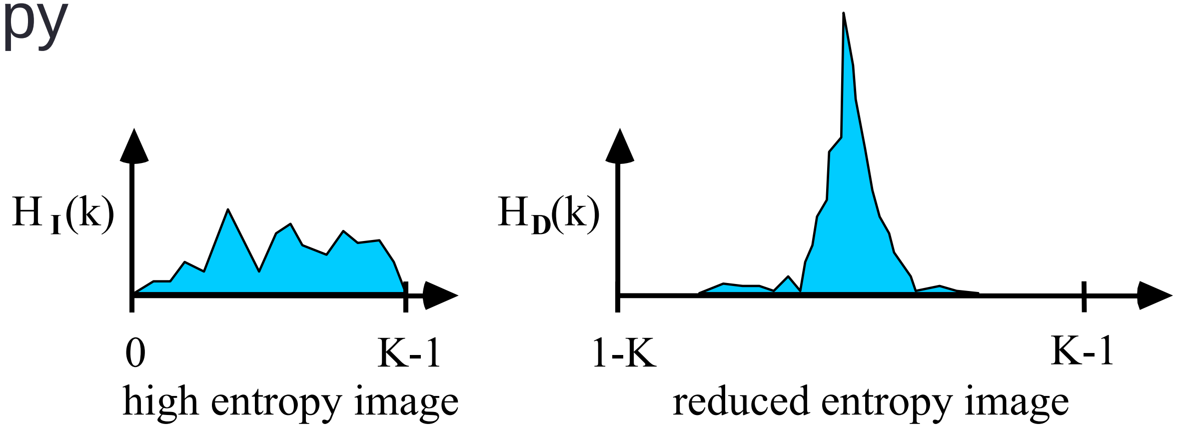
- Code the difference between adjacent pixels

Original pixels:
82, 83, 86, 88, 56, 55, 56, 60, 58, 55, 50, ……

→

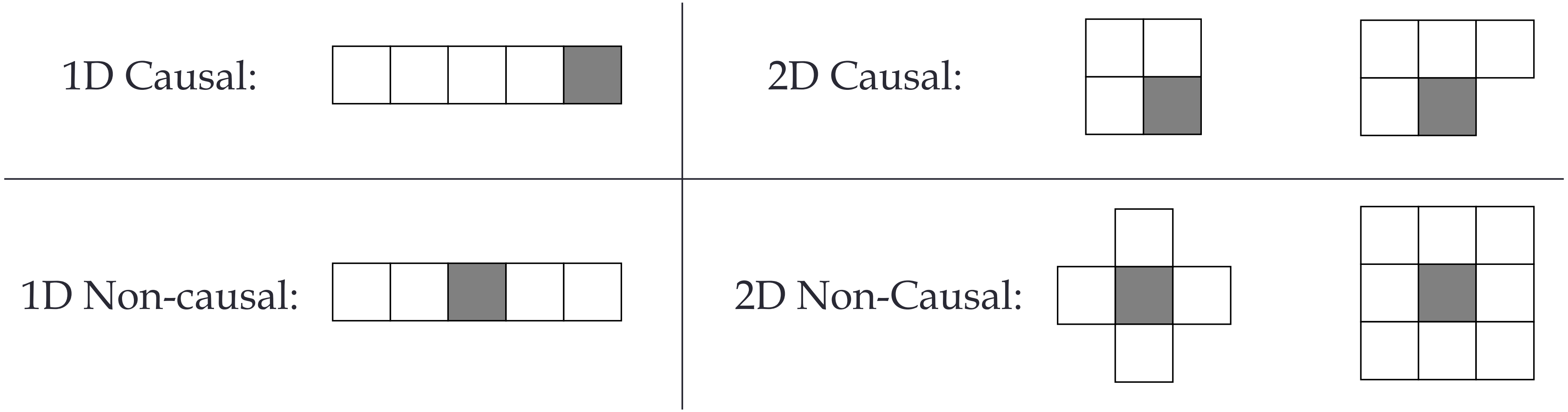
DPCM:
82, 1, 3, 2, -32, -1, 1, 4, -2, -3, -5, ……

- Prediction is that the next pixel value = current one
- Need the first value to provide a point of reference
- Invertible (lossless) and lower entropy



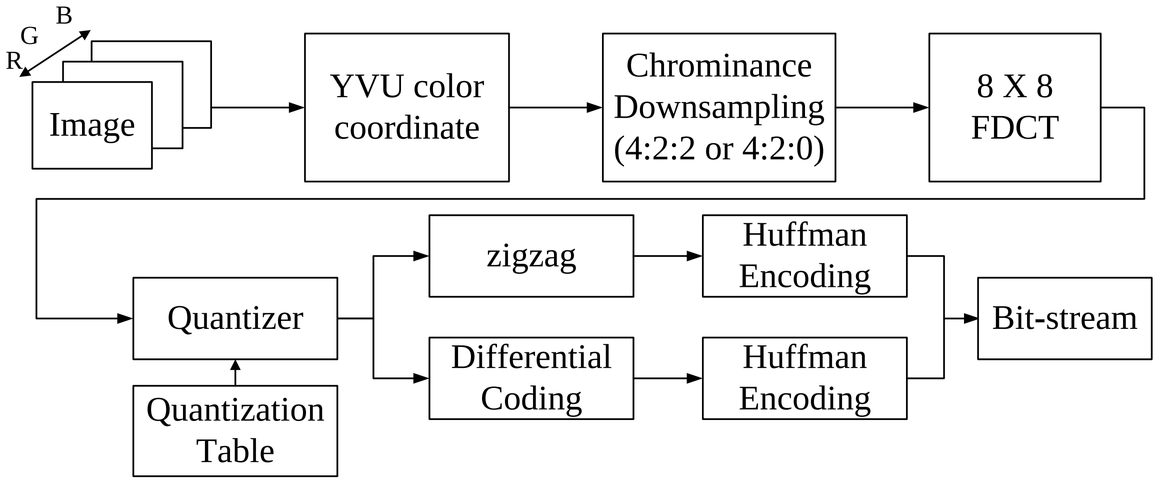
# Predictive Coding

- Higher order pattern prediction
- Use both 1D and 2D patterns (to predict shaded pixel)



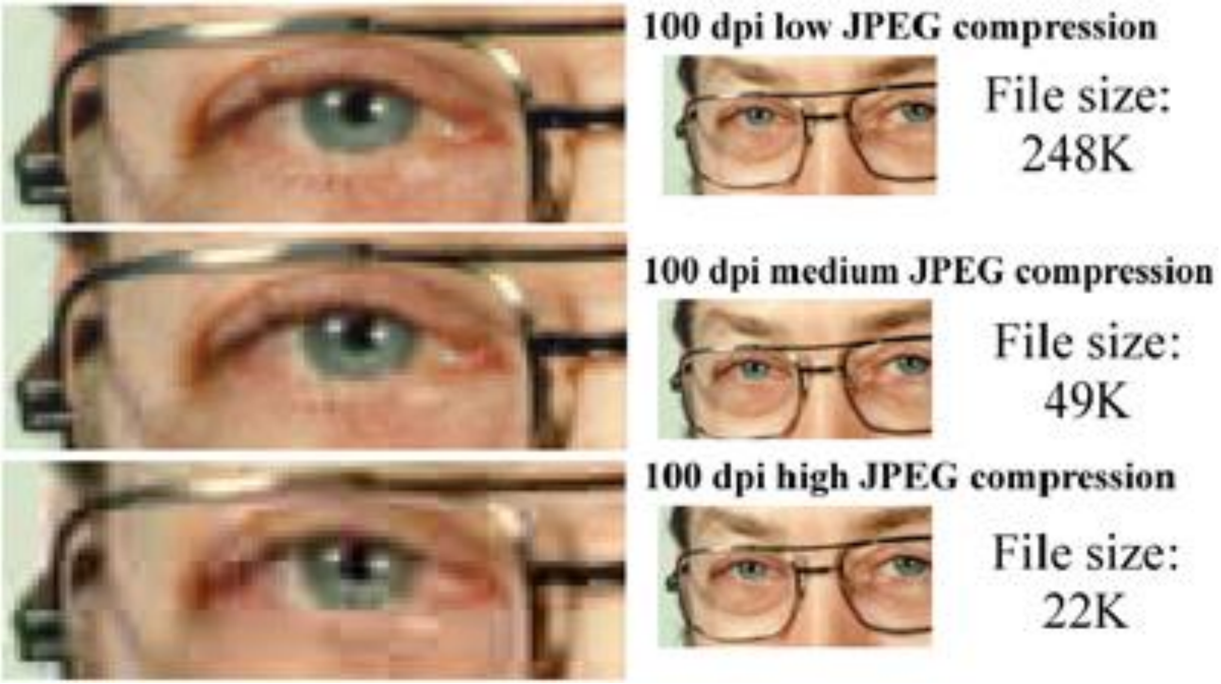
# A Complete System: JPEG

- A set of methods with a common *baseline* system
  - Discrete Cosine Transform
  - Quantisation
  - Variable length encoding
- A JPEG-compatible product must only include support for the baseline



# JPEG Compression

- Increasing the amount of quantisation reduces file size but introduces artefacts: blocks become visible



# NEXT WEEK

CNN

Conclusion and Exam Revision