**THESE NOTES ARE NOT COMPLETE/CHECKED AND COULD DO WITH UPDATING. I’LL GET ROUND TO IT OVER THIS WEEKEND BUT ANYONE FEEL FREE TO AMEND/ADD TO THEM**

Colour can be represented by 3 colour components RGB,

or a luminance (monochrome) component and 2 chrominance (colour) components

**Frequency Domain Processing**

Allows us to take advantage of the physiological features of the way we see images. It avoids encoding what the eye will not see.

We use the DCT to transform into the FD, although this time we need a *2 dimensional DCT.*

Most of the energy in a picture is in the lower frequencies. Take a threshold, and set all low frequencies to zero. Slightly blurred picture, but recognizable.

**JPEG Compression**

**Step 1**

Divide image into 8x8 tiles

For each tile, convert each RGB pixel to a measure of luminance, plus two chrominance measurements

Reduce 8x8 chrominance to 4x4 by averaging 2x2 blocks.

Convert back at the receiver, by copying each tile 4 times.

**Step 2** Apply 2D Discrete Cosine Transform to each 8x8 tile of luminance, and chrominance.

**Step 3** Quantize DCT coefficients by dividing by correlating entries in a “quantization table”, and then rounding to an integer.

The division occurs to get rid of a certain number of bits.

Can be inversed by multiplying out by the quantization coefficients, although quantization error will have occurred

**Step 4** Encode differences in DC-DC components between tiles.

**Step 5** reads coefficients in a zigzag fashion, and counts number of successive zeros. record this number as “run length code”.

Can then be encoded using “run length encoding”. Record in tuples, the number of consecutive zeros, paired with the next non-zero number.

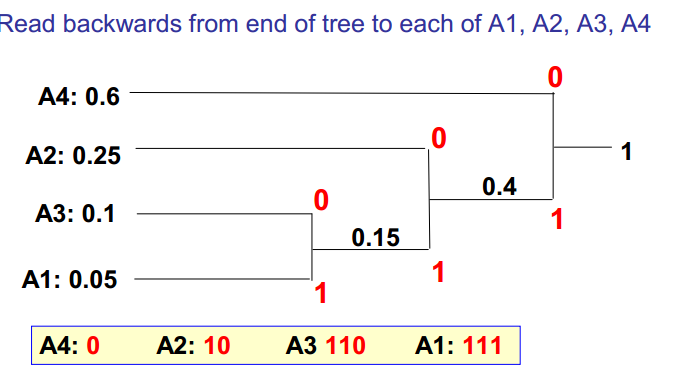
**Example** 0 0 0 0 0 2 3 0 0 0 0 0 0 0 0 1

(5,2), (0,3), (8,1)

**Step 6** Use Huffman coding, rarer numbers get longer codes.

**Step 7** 

**Huffman coding** - arrange in order of probability. Link two lowest and add probabilities, forming a sort of tree. Add this to the next most probable etc etc. Put 0 on top vertices, 1 on bottom vertices. Read tree from right to left and note each bit. The resulting string is the huffman code for that number.



Huffman coding is very sensitive to bit-errors. An error in one code could be misinterpreted as part of another adjacent code.

**MPEG Encoding**

Encode the difference between frames, to exploit temporal redundancy. Encode frames as I-Frame, P-Frame, B-Frame

* I-Frame - Image encoded as JPEG
* P-Frame - encodes positions of moving blocks predicted from previous I&P frames
* B-Frame - encodes positions of moving blocks estimated from both previous and next I&P frames.