**Source Coding Techniques**

Source coding is based on the content of the original signal is also called ***semantic-based coding***

High compression rates may be high but a price of loss of information. Good compression rates make be achieved with source encoding with ***lossless*** or little loss of information.

There are three broad methods that exist:

1  **Transform Coding**

A Simple Transform Encoding procedure maybe described by the following steps for a 2x2 block of monochrome pixels:

**1.**

Take top left pixel as the base value for the block, pixel A.

**2.**

Calculate three other transformed values by taking the difference between these (respective) pixels and pixel A, i.e. B-A, C-A, D-A.

**3.**

Store the base pixel and the differences as the values of the transform.

Given the above we can easily for the forward transform:

and the inverse transform is:

The above transform scheme may be used to compress data by exploiting redundancy in the data:

Any Redundancy in the data has been transformed to values, *Xi*. So We can compress the data by using fewer bits to represent the differences. I.e if we use 8 bits per pixel then the 2x2 block uses 32 bits/ If we keep 8 bits for the base pixel, X0, and assign 4 bits for each difference then we only use 20 bits. Which is better than an average 5 bits/pixel

**Example**

Consider the following 4x4 image block:

|  |  |
| --- | --- |
| 120 | 130 |
| 125 | 120 |

then we get:

We can then compress these values by taking less bits to represent the data.

However for practical purposes such a simple scheme as outlined above is not sufficient for compression:

* It is **Too Simple**
* Needs to operate on larger blocks (typically 8x8 min)
* Calculation is also too simple and from above we see that simple encoding of differences for large values will result in loss of information -- v poor losses possible here 4 bits per pixel = values 0-15 unsigned, -7 - 7 signed so either quantise in multiples of 255/max value or massive overflow!!

## 2 Frequency Domain Methods

Frequency domains can be obtained through the transformation from one (Time or Spatial) domain to the other (Frequency) via

* Discrete Cosine Transform,
* Fourier Transform etc.

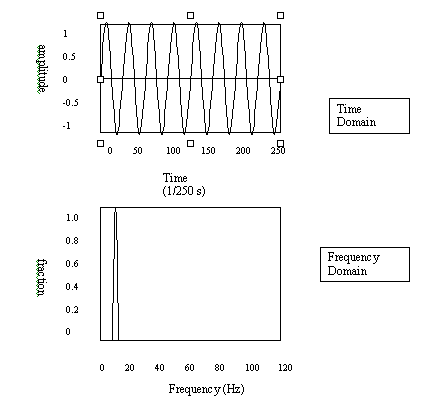
### 1D Example

Lets consider a 1D (e.g. Audio) example to see what the different domains mean:

Consider a complicated sound such as the noise of a car horn. We can describe this sound in two related ways:

* sample the amplitude of the sound many times a second, which gives an approximation to the sound as a function of time.
* analyse the sound in terms of the pitches of the notes, or frequencies, which make the sound up, recording the amplitude of each frequency.

In the example below (Fig [[*]](http://www.cs.cf.ac.uk/Dave/Multimedia/node221.html#fig:frq)) we have a signal that consists of a sinusoidal wave at 8 Hz. 8Hz means that wave is completing 8 cycles in 1 second and is the frequency of that wave. From the frequency domain we can see that the composition of our signal is one wave (one peak) occurring with a frequency of 8Hz with a magnitude/fraction of 1.0 i.e. it is the whole signal.

   
**Relationship between Time and Frequency Domain**

### 2D (Image) Example

Now images are no more complex really:

Similarly brightness along a line can be recorded as a set of values measured at equally spaced distances apart, or equivalently, at a set of spatial frequency values.

Each of these frequency values is referred to as a **frequency component**.

An image is a two-dimensional array of pixel measurements on a uniform grid.

This information be described in terms of a two-dimensional grid of spatial frequencies.

A given frequency component now specifies what contribution is made by data which is changing with specified *x* and *y* direction spatial frequencies.

### What do frequencies mean in an image?

If an image has large values at **high** frequency components then the data is changing rapidly on a short distance scale. **e.g.** a page of text

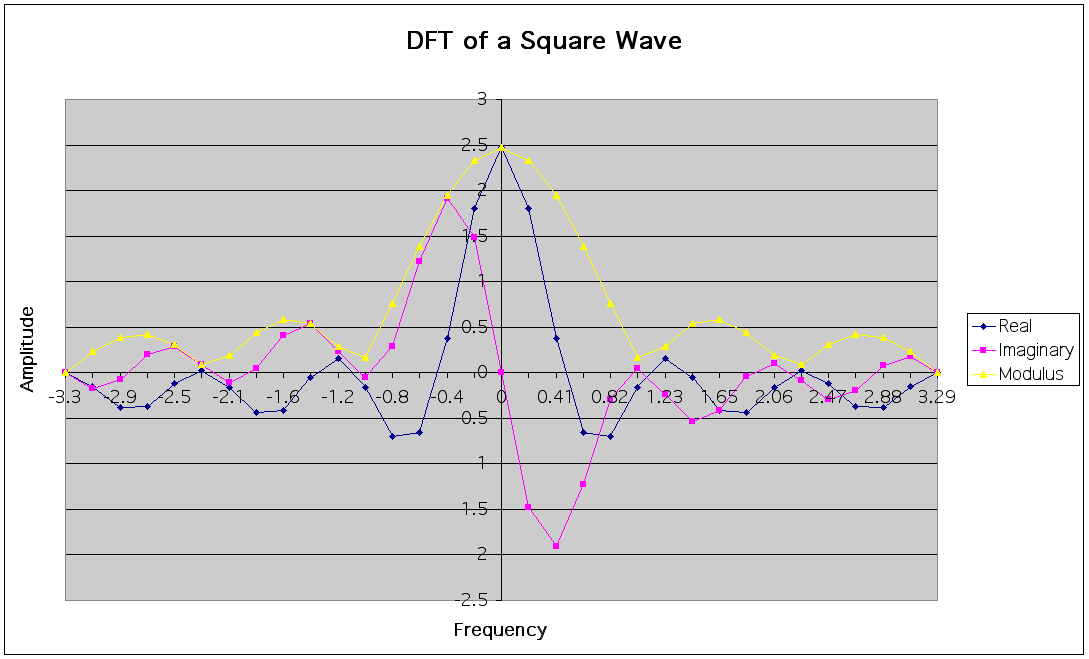
If the image has large **low** frequency components then the large scale features of the picture are more important. **e.g.** a single fairly simple object which occupies most of the image.

For colour images, The measure (now a 2D matrix) of the frequency content is with regard to colour/chrominance: this shows if values are changing rapidly or slowly. Where the fraction, or value in the frequency matrix is low, the colour is changing gradually. Now the human eye is insensitive to gradual changes in colour and sensitive to intensity. So we can ignore gradual changes in colour and throw away data without the human eye noticing, we hope.

### How can transforms into the Frequecny Domain Help?

Any function (signal) can be decomposed into purely sinusoidal components (sine waves of different size/shape) which when added together make up our original signal.

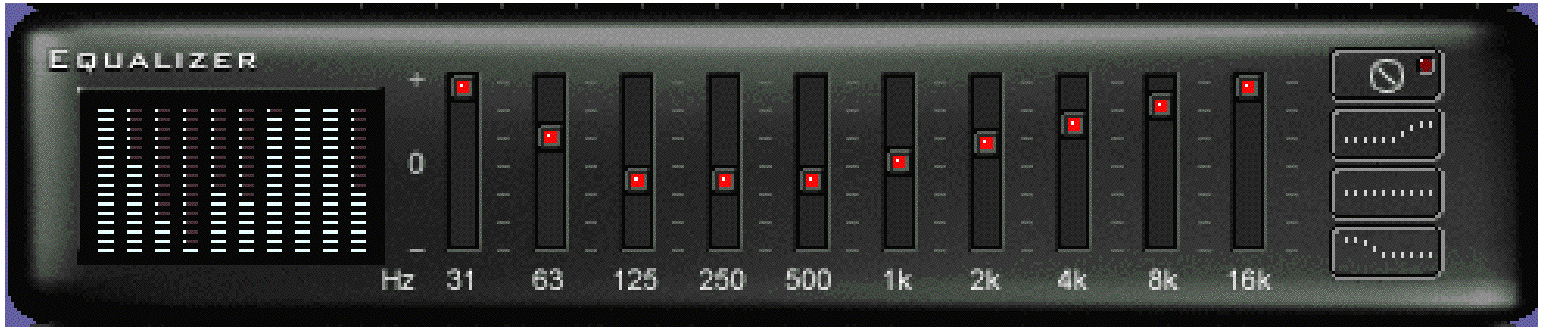
In the example below (Fig [7.4](http://www.cs.cf.ac.uk/Dave/Multimedia/node224.html#fig:DFT_sq)) we have a square wave signal that has been decomposed by the Fourier Transform to render its sinusoidal components. Only the first few sine wave components are shown here. You can see that a the Square wave form will be roughly approximated if you add up the sinusoidal components.

   
**DFT of a Square Wave**

Thus Transforming a signal into the frequency domain allows us to see what sine waves make up our signal e.g. One part sinusoidal wave at 50 Hz and two parts sinusoidal waves at 200 Hz.

More complex signals will give more complex graphs but the idea is exactly the same. The graph of the frequency domain is called the frequency spectrum.

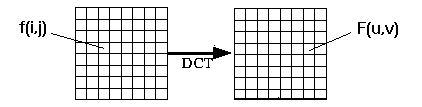
An easy way to visualise what is happening is to think of a graphic equaliser on a stereo (Fig [7.5](http://www.cs.cf.ac.uk/Dave/Multimedia/node224.html#fig:graphic_eq)).

  
**A Graphic Equaliser**

The bars on the left are the frequency spectrum of the sound that you are listening to. The bars go up and down depending on the type of sound that you are listening to. It is pretty obvious that the accumulation of these make up the whole. The bars on the right are used to increase and decrease the sound at particular frequencies, denoted by the numbers (Hz). The lower frequencies, on the left, are for bass and the higher frequencies on the right are treble.

## The Discrete Cosine Transform (DCT)

  The discrete cosine transform (DCT) helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality). The DCT is similar to the discrete Fourier transform: it transforms a signal or image from the spatial domain to the frequency domain (Fig [7.8](http://www.cs.cf.ac.uk/Dave/Multimedia/node231.html#DCTenc)).



**DCT Encoding**

The general equation for a 1D (*N* data items) DCT is defined by the following equation:

\begin{displaymath}
F(u) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1}
\Lambda(i).cos\left[
\frac{\pi.u}{2.N}(2i+1)
\right]f(i)\end{displaymath}

and the corresponding **inverse** 1D DCT transform is simple *F-1*(*u*), i.e.:

where

\begin{displaymath}
\Lambda(i) = \left\{ \begin{array}
{ll} \frac{1}{\sqrt{2}} & {\rm
for}
\xi = 0\ 1 & {\rm otherwise}\end{array} \right.\end{displaymath}

The general equation for a 2D (*N* by *M* image) DCT is defined by the following equation:

\begin{displaymath}
F(u,v) = \left(\frac{2}{N}\right)^{\frac{1}{2}}
\left(\frac{...
 ...}(2i+1)
\right]cos\left[ \frac{\pi.v}{2.M}(2j+1) \right].f(i,j)\end{displaymath}

and the corresponding **inverse** 2D DCT transform is simple *F-1*(*u*,*v*), i.e.:

where

\begin{displaymath}
\Lambda(\xi) = \left\{ \begin{array}
{ll} \frac{1}{\sqrt{2}} & {\rm
for}
\xi = 0 \ 1 & {\rm otherwise}\end{array} \right.\end{displaymath}

The basic operation of the DCT is as follows:

* The input image is N by M;
* f(i,j) is the intensity of the pixel in row i and column j;
* F(u,v) is the DCT coefficient in row k1 and column k2 of the DCT matrix.
* For most images, much of the signal energy lies at low frequencies; these appear in the upper left corner of the DCT.
* Compression is achieved since the lower right values represent higher frequencies, and are often small - small enough to be neglected with little visible distortion.
* The DCT input is an 8 by 8 array of integers. This array contains each pixel's gray scale level;
* 8 bit pixels have levels from 0 to 255.
* Therefore an 8 point DCT would be:

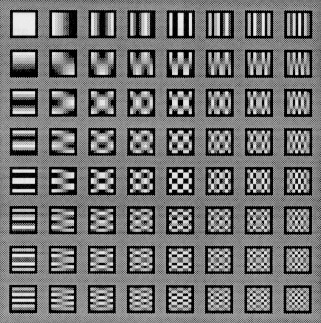
where

\begin{displaymath}
\Lambda(\xi) = \left\{ \begin{array}
{ll} \frac{1}{\sqrt{2}} & {\rm
for}
\xi = 0 \ 1 & {\rm otherwise}\end{array} \right.\end{displaymath}

**Question**: What is F[0,0]?

**answer:** They define DC and AC components.

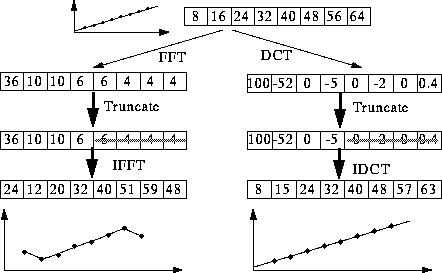
* The output array of DCT coefficients contains integers; these can range from -1024 to 1023.
* It is computationally easier to implement and more efficient to regard the DCT as a set of **basis functions** which given a known input array size (8 x 8) can be precomputed and stored. This involves simply computing values for a convolution mask (8 x8 window) that get applied (summ values x pixelthe window overlap with image apply window accros all rows/columns of image). The values as simply calculated from the DCT formula. The 64 (8 x 8) DCT basis functions are illustrated in Fig [7.9](http://www.cs.cf.ac.uk/Dave/Multimedia/node231.html#DCTbasis).



**DCT basis functions**

* Why DCT not FFT?

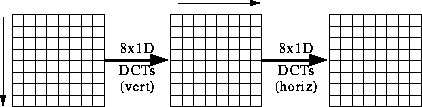
DCT is similar to the Fast Fourier Transform (FFT), but can approximate lines well with fewer coefficients (Fig [7.10](http://www.cs.cf.ac.uk/Dave/Multimedia/node231.html#DFTFFT))



**DCT/FFT Comparison**

* Computing the 2D DCT
  + Factoring reduces problem to a series of 1D DCTs (Fig [7.11](http://www.cs.cf.ac.uk/Dave/Multimedia/node231.html#DCTcomp)):
    - apply 1D DCT (Vertically) to Columns
    - apply 1D DCT (Horizontally) to resultant Vertical DCT above.
    - or alternatively Horizontal to Vertical.

The equations are given by:



* + Most software implementations use fixed point arithmetic. Some fast implementations approximate coefficients so all multiplies are shifts and adds.
  + World record is 11 multiplies and 29 adds. (C. Loeffler, A. Ligtenberg and G. Moschytz, "Practical Fast 1-D DCT Algorithms with 11 Multiplications", Proc. Int'l. Conf. on Acoustics, Speech, and Signal Processing 1989 (ICASSP `89), pp. 988-991)

## 3 Differential Encoding

Simple example of transform coding mentioned earlier and instance of this approach.

Here:

* The difference between the actual value of a sample and a prediction of that values is encoded.
* Also known as **predictive encoding**.
* Example of technique include: differential pulse code modulation, delta modulation and adaptive pulse code modulation -- differ in prediction part.
* Suitable where successive signal samples do not differ much, but are not zero. **E.g.** Video -- difference between frames, some audio signals.
* **Differential pulse code modulation** (DPCM) simple prediction:

*fpredict*(*ti*) = *factual*(*ti*-1)

**i.e.** a simple Markov model where current value is the predict next value.

So we simply need to encode:

\begin{displaymath}
\Delta f(t_i) = f_{actual}(t_{i}) - f_{actual}(t_{i-1})\end{displaymath}

If successive sample are close to each other we only need to encode first sample with a large number of bits:

Actual Data: 9 10 7 6

Predicted Data: 0 9 10 7

$\Delta f(t)$: +9, +1, -3, -1.

* **Delta modulation** is a special case of DPCM: Same predictor function, coding error is a single bit or digit that indicates the current sample should be increased or decreased by a step.

Not Suitable for rapidly changing signals.

* **Adaptive pulse code modulation** -- Fuller Markov model: data is extracted from a function of a series of previous values: **E.g.** Average of last *n* samples. Characteristics of sample better preserved.

## 4 Vector Quantization

The basic outline of this approach is:

* Data stream divided into (1D or 2D square) blocks -- **vectors**
* A table or **code book** is used to find a pattern for each block.
* Code book can be dynamically constructed or predefined.
* Each pattern for block encoded as a look value in table
* Compression achieved as data is effectively sub-sampled and coded at this level.