**Wavelet transforms for 1D signals:** A wavelet transform combines both lowpass and highpass filtering in a spectral decomposition of signals along with an extremely fast implementation. Given a 1D signal s0[n], its 1-level wavelet transform is the mapping s0[n] (s1[2n] | d1[2n]) defined by



The signals s1[2n] and d1[2n] are respectively lowpass and highpass filterings of s0[n]. These filterings have also been *down sampled:* they are defined over the indices {2n} rather than {n}. Viewed as sampled signals, they are sampled at half the rate as s0. The coefficients {αk} are the *lowpass coefficients* and the coefficients {βk} are the *highpass coefficients.* A simple example of these ideas is the Daub 5/3 wavelet system [3]. For this system, the lowpass and highpass coefficients are defined by



These coefficients have some basic properties which are shared by other wavelet systems. One important property is that they define an invertible transform; we shall discuss this further later. Perhaps just as importantly, the highpass coefficients satisfy  Consequently, if s0 is linear (or approximately linear) over the indices 2n, 2n + 1, 2n + 2, then d1[2n] = 0 (or d1[2n] 0). When s0 is obtained from samples of a piecewise smooth function, the highpass filtering d1 will be essentially zero-valued (except near transitions between pieces of the piecewise smooth function). This provides the foundation for compression. When the transform is iterated on the lowpass outputs s1,s2... , then several levels of transform will produce large numbers of zero values (or nearly zero values) in the highpass outputs d2, d3, ... .Such high redundancy of zero values, in d1, d2, d3, …. , allows for significant compression. It is also worth noting that the Daub 5/3 lowpass coefficients satisfy These equations imply the approximation s1[2n]s0[2n], at most indices n, when s0 is sampled from a piecewise smooth signal. This approximation is useful in detection and denoising algorithms. The equation  also tells us that each lowpass output is an average: s1[2n] is an average over the signal values , while s2[4n] is an average over the lowpass values  and hence also over the signal values , and so on. Notice that s2[4n] is an average over a wider range of original signal values than s1[2n] (more than twice as many values, a doubling in scale). Similarly, the highpass output d2[4n] is a response to more than twice the range (twice the scale) of signal values as d1[2n]. This doubling of scale occurs in passing to each new level of the transform. Another important wavelet system for image compression is the Daub 9/7 system [3]. This system involves more complicated highpass and lowpass coefficients. In particular, these coefficients are not simple rational numbers—as in the Daub 5/3 system. The additional requirement of using 32-bit floating point numbers for implementing the Daub 9/7 system is one reason that our image compression algorithm employs the Daub 5/3 system (other advantages of the Daub 5/3 system will be described below). The Daub 9/7 highpass coefficients satisfy  and . The extra zero sum implies even greater compression. If a signal is well-approximated by piecewise quadratics, then a huge number of (approximately) zero-values will appear in its wavelet transform. This is especially important in image compression, where approximation of an image by piecewise quadratic surfaces is justified based on standard illumination models [6].

**Tree type representation in the discrete case:**

**Generalization for images:** Some of the generalizations are:

* Biorthogonal wavelet bases

As we mentioned in Section 12.2, the wavelet used for reconstruction in the continuous wavelet transform need not be the same as that used for decomposition, the two have only to satisfy a cross-compatibility condition. The same idea in the discrete case leads to biorthogonal bases, i.e. one has two hierarchies of approximation spaces, Vj and , with cross-orthogonality relations. This gives a better control, for instance, on the regularity or decrease properties of the wavelets [91].

* Wavelet packets and best basis algorithm

The construction of orthonormal wavelet bases leads to a special sub-band coding scheme, rather asymmetrical: each sequence cj gets further decomposed into cj+1 and dj+1, whereas the detail sequence dj is left unmodified. Thus more flexible sub-band schemes have been considered, called wavelet packets where both subspaces Vj-1 and Wj-1 are decomposed at each step. They provide rich libraries of orthonormal bases, and also strategies for determining the optimal basis in a given situation.

The best basis algorithm finds a set of wavelet bases that provide the most desirable representation of the data relative to a particular cost function. A cost function may be chosen to fit a particular application. For example, in a compression algorithm the cost function might be the number of bits needed to represent the result. *K* is given as

The value of the cost function is a real number. Given two vectors of finite length, **a** and **b**, we denote their concatenation by [**a b**]. This vector simply consists of the elements in **a** followed by the elements in **b**. We require the following two properties:

1. The cost function is additive in the sense that *K*([**a b**]) = *K*(**a**) + *K*(**b**) for all finite length vectors **a** and **b**.
2. *K*(**0**) = 0, where **0** denotes the zero vector

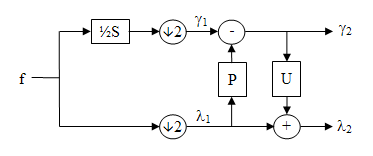
* The lifting scheme: Second generation wavelets

One can go further and abandon the regular dyadic scheme and the Fourier transforms altogether. Using the `lifting scheme' of Sweldens [276], one obtains the so-called second-generation wavelets, which are essentially custom-designed for any given problem. This approach uses the fact that, in the biorthogonal scheme, a given wavelet fixes its biorthogonal partner only up to a rather arbitrary function. Then one starts from a very simple wavelet and gradually obtains the needed one by choosing successive appropriate biorthogonal partners.

The input signal *f* is split into odd γ1 and even λ1 samples using shifting and [downsampling](http://en.wikipedia.org/wiki/Downsampling). The detail coefficients γ2 are then interpolated using the values of γ1 and the *prediction operator* on the even values:

γ2 = γ1 − *P*(λ1)

The next stage (known as the *updating operator*) alters the approximation coefficients using the detailed ones:



* Integer wavelet transforms

In their standard numerical implementation, the classical (discrete) WT converts floating point numbers into floating point numbers. However, in many applications (data transmission from satellites, multimedia), the input data consists of integer values only and one cannot afford to lose information: only lossless compression schemes are allowed. Recent developments have produced new methods that allow one to perform all calculations in integer arithmetic.

**Frequency interpretation of wavelet transforms:**

**An idea to compression:** The simplest version of a forward wavelet transform expressed in the Lifting Scheme is shown below in Figure 1. The predict step is the subject of this web page, which will considered in isolation. The predict step calculates the wavelet function in the wavelet transform. This is a high pass filter. The update step calculates the scaling function, which results in a smoother version of the data. Figure 1

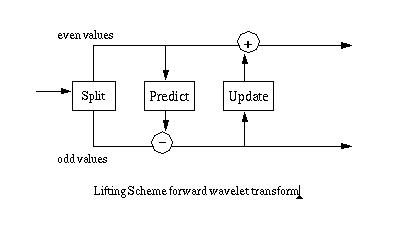
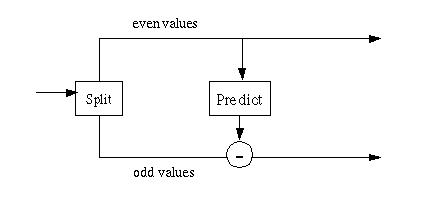
****

Figure 2 shows the *ur*-Lifting Scheme transform, consisting of two steps:

Split step: divide the input data into odd and even elements. In a finite data set the odd elements are moved to the second half of the array, leaving the even elements in the first half. Predict step: predict the odd elements from the even elements. Figure 2

****

One way to view the predict step is through the lens of data compression. If our objective is to compress a set of data and the odd elements can be absolutely predicted from the even elements using the equation

*odd* = *even* \* 2;

the odd elements can be replaced by zero. If we apply a compression algorithm like run length encoding the odd elements will be reduced to a count and zero, compressing the original data set by almost 50%. If the data set consists of points on a line, then it can be reduced to something close to a single element and the length of the data set. In most cases the data set is more complex and it cannot be entirely represented by a starting condition, a length and an equation. However, a more compact representation might be arrived at by approximating the data in a local region using a function. The predict stage replaces an odd element with the difference between the odd element a function calculated from the even elements. The simplest example of such a predict stage takes a single even element as its argument to calculate the predicted value of the odd element:

*odd*j+1,i = *odd*j,i - **P**(*even*j,k)

Here the function **P**() is the predict function. Wavelet algorithms are recursive, so the recursive step *j* generates data for the next recursive step *j+1*. The subscript *i* indexes the odd part of the array. The subscript *k* indexes the even part of the array.

One of the simplest predict functions is simply

*odd*j+1,i = *even*j,k

If the split step had not divided the odd and even elements, the predict step predicts that the odd value is equal to its even predecessor.

ai+1 = ai;

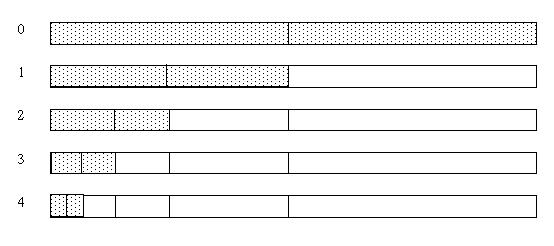
The predict step replaces the odd elements with the difference between the actual odd value and the predicted value:

*odd*j+1,i = *odd*j,i - *even*j,k

If the data shows a trend (in the language of statistics the data shows autocorrelation), then the odd element can be predicted from the even element, to some degree. As a result, the difference between the odd element and its predictor (the even element) will be smaller than the odd element itself. Smaller values can be represented in fewer bits, so some level of compression can be achieved.

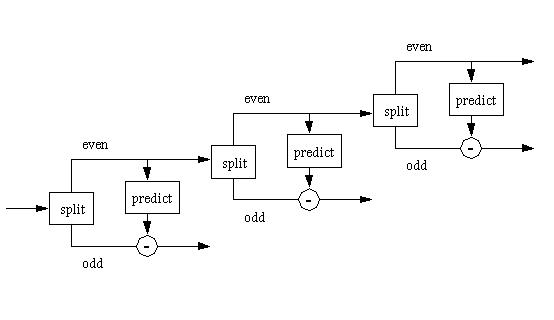
The process of "predicting" the odd elements from the even elements is recursive, as long as the number of data elements is a power of two. After the first pass, the odd (upper) half of the array will contain the differences between the prediction and the original odd element values. The next recursive pass divides the lower half of the array into odd and even halves. The difference between the prediction and the odd element value is stored in the new odd half. The recursive passes continue until the last step where a single odd element is predicted from a single even element. This is shown in Figure 3 below.

Figure 3

****

Another way to view this is as a chain of split/predict steps, where the even elements from one step become the input for the next step.

Figure 4

****