LIFTPACK: A Software Package for Wavelet Transforms using Lifting

Gabriel Fernández Senthil Periaswamy

Department of Computer Science
University of South Carolina, Columbia, SC 29208
e-mail: fernande@cs.sc.edu periaswa@cs.sc.edu

Wim Sweldens

Lucent Technologies, Bell Laboratories, Rm. 2C-175
700 Mountain Avenue, Murray Hill, NJ 07974
e-mail: wim@bell-labs.com

ABSTRACT

We present LIFTPACK: A software package written in C for fast calculation of 2D biorthogonal wavelet transforms using the lifting scheme. The lifting scheme is a new approach for the construction of *biorthogonal wavelets* entirely in the spatial domain, i.e., independent of the Fourier Transform. Constructing wavelets using lifting consists of three simple phases: the first step or Lazy wavelet *splits* the data into two subsets, even and odd, the second step calculates the wavelet coefficients (high pass) as the failure to *predict* the odd set based on the even, and finally the third step *updates* the even set using the wavelet coefficients to compute the scaling function coefficients (low pass). The predict phase ensures polynomial cancelation in the high pass (vanishing moments of the dual wavelet) and the update phase ensures preservation of moments in the low pass (vanishing moments of the primal wavelet). By varying the order, an entire family of transforms can be built. The lifting scheme ensures fast calculation of the forward and inverse wavelet transforms that only involve FIR filters. The transform works for images of arbitrary size with correct treatment of the boundaries. Also, all computations can be done in-place.

Keywords: liftpack, wavelets, interpolating subdivision, lifting scheme, separable transform

1 INTRODUCTION

The purpose of this paper is to give an introduction to the *lifting scheme*, provide references for further reading, and, most importantly, give an explanation of LIFTPACK, a C library that implements biorthogonal wavelets using lifting.

A fast and efficient algorithm of constructions of biorthogonal wavelets on the infinite real line is established in Section 2. Basic polynomial interpolation is used to find the high frequency values (wavelet or γ coefficients), later used to construct the *scaling functions* in order to find the low frequency values (lambda or λ coefficients). Interpolating subdivision[12] is used as an efficient way to find wavelet coefficients through a refinable method that gives better results if a smoother function is used and treats values near the boundaries correctly. Later, it is shown how the *Lifting Scheme*[11,10,8,12] is used to construct "second generation wavelets" for more general frameworks.

The Lifting Scheme consists of three main steps: SPLIT, which subsamples the original data into odd and even sets; PREDICT, which finds the wavelet coefficients as the failure to predict the odd set based upon the even; and UPDATE, which updates the even set using the wavelet coefficients to compute the scaling function coefficients.

The implementation in 1-D consists of two stages. First, the calculation of filter and lifting coefficients. Second, the 1-D fast lifted wavelet transform. The extension to higher dimensions is obtained with the application of the 1-D transform as a separable transform.

2 LIFTING SCHEME ALGORITHM

The lifting scheme is a new method for constructing wavelets. The main difference with classical constructions[1–4] is that it is does not rely on the Fourier transform. This way, lifting can be used to construct *second generation wavelets*, i. e., wavelets which are not necessarily translations and dilations of one function. The latter we refer to as *first generation wavelets* or *classical wavelets*.

Since the lifting scheme does not depend on the Fourier transform, it has applications in the following examples:

Wavelets on bounded domains: The construction of wavelets on an interval is needed to transform finite length signals without introducing artifacts in the boundaries. The remainder of this paper gives more details.

Wavelets on curves and surfaces: This case is related to solving equations on curves or surfaces or analysis of data that live on curves or surfaces. See the paper on spherical wavelets[8] for more details.

Weighted wavelets: Wavelets biorthogonal with respect to a weighted inner product are needed for diagonalization of differential operators and weighted approximation.

Wavelets and irregular sampling: Many real life problems require basis functions and transforms adapted to irregular sampled data. See the paper on building your own wavelets at home[12] for a further explanation of implementation of wavelets using irregular sampled data.

It can be seen that wavelets adapted to these settings cannot be formed by translation and dilation. The Fourier transform can thus no longer be used as a construction tool. The lifting scheme provides an alternative.

The basic idea behind lifting scheme is very simple. We begin with a trivial wavelet, the "Lazy wavelet"; a function which essentially does not calculate anything, but which has the formal properties of a wavelet. The lifting scheme then gradually builds a new wavelet, with improved properties. This is the inspiration behind the name "lifting scheme". To explain the way the lifting scheme works, we refer to the block diagram in Figure 1 that shows the three stages of lifting: split, predict, and update. We start with a simple case. Then we develop a general framework to create any type of wavelet.

Suppose we sample a signal $f(t)$ with sampling distance 1. We denote the original samples as $\lambda_{0,k} = f(k)$ for $k \in \mathbf{Z}$. We would like to "decorrelate" this signal. In other words, we would like to see if it is possible to capture the information contained in this signal with fewer coefficients, i. e., coefficients with a larger sampling distance. A more compact representation is needed in applications such as data compression. Maybe it will not be possible to *exactly* represent the signal with fewer coefficients but instead find an *approximation* within an acceptable error bound. We thus want to have precise control over the information which is lost by using fewer coefficients. Obviously, we would like the difference between the original and approximated signals to be small.

2.1 Split phase

We can reduce the number of coefficients by simply subsampling the even samples and obtaining a new sequence given by

$$\lambda_{-1,k} := \lambda_{0,2k} \quad \text{for} \quad k \in \mathbf{Z}, \tag{1}$$

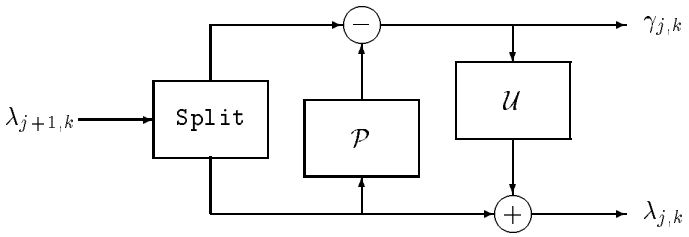

Figure 1: *The Lifting Scheme.* Split, predict, and update.

where we use negative indices because the convention is that the smaller the data set, the smaller the index.

We also would like to have an idea on how much information was lost. In other words, which extra information (if any) do we need to recover the original set $\{\lambda_{0,k}\}$ from the set $\{\lambda_{-1,k}\}$. We use coefficients $\{\gamma_{-1,k}\}$ to encode this difference and refer to them as *wavelet coefficients*. Many different choices are possible and, depending on the statistical behavior of the signal, one will be better than the other. Better means smaller wavelet coefficients.

The most naive trivial choice would be to say that the lost information is simply contained in the odd coefficients, $\gamma_{-1,k} := \lambda_{0,2k+1} \quad \text{for} \quad k \in \mathbf{Z}$. This choice corresponds to the *Lazy wavelet*. Indeed, we have not done very much except for subsampling the signal in even and odd samples. Obviously this will not decorrelate the signal. The wavelet coefficients are only small when the odd samples are small and there is no reason whatsoever why this should be the case.

It is important to point out that we do not impose any restriction on how the data set should be split, nor on the relative size of each of the subsets. We simply need a procedure to join $\{\lambda_{-1,k}\}$ and $\{\gamma_{-1,k}\}$ again into the original data set $\{\lambda_{0,k}\}$. The easiest possibility for the split is a simply brutal cut of the data set into two disjoint parts, but a split between even and odd samples is a better choice. We refer to this choice as the *Lazy wavelet*.

The next step, the predict, will help us find a more elaborate scheme to recover the original samples $\{\lambda_{0,k}\}$ from the subsampled coefficients $\{\lambda_{-1,k}\}$.

2.2 Predict phase

As mentioned before, we would like to obtain a more compact representation of $\{\lambda_{0,k}\}$. Consider the case where $\{\gamma_{-1,k}\}$ does not contain any information (e. g. that part of the signal is equal to zero). Then we would immediately have a more compact representation since we can simply replace $\{\lambda_{0,k}\}$ with the smaller set $\{\lambda_{-1,k}\}$. Indeed, the extra part needed to reassemble $\{\lambda_{0,k}\}$ does not contain any information.

The previous situation hardly ever occurs in practice. Therefore, we need a way to find the odd samples $\{\gamma_{-1,k}\}$. The even samples $\{\lambda_{0,2k}\}$ can immediately be found as $\lambda_{0,2k} := \lambda_{-1,k}$. On the other hand, we try to find or, at least, predict the odd samples $\{\lambda_{0,2k+1}\}$ based on the correlation present in the original data. If we can find a prediction operator $\mathcal{P}$, independent of the data, so that

$$\gamma_{-1,k} := \mathcal{P}(\lambda_{-1,k}),$$

then again we can replace the original data set with $\{\lambda_{-1,k}\}$, since now we can predict the part missing to reassemble $\{\lambda_{0,k}\}$. The construction of the prediction operator is typically based on some model of the data which reflects its correlation structure. Obviously, the prediction operator $\mathcal{P}$ cannot be dependent on the data, otherwise we would hide information in $\mathcal{P}$.

Again, in practice, it might not be possible to exactly predict $\{\gamma_{-1,k}\}$ based on $\{\lambda_{-1,k}\}$. However, $\mathcal{P}(\lambda_{-1,k})$ is likely to be close to $\{\gamma_{-1,k}\}$. Thus, we might want to replace $\{\gamma_{-1,k}\}$ with the difference between itself and its predicted value $\mathcal{P}(\lambda_{-1,k})$. If the prediction is reasonable, this difference will contain much less information than the original $\{\gamma_{-1,k}\}$ set. We denote this abstract difference operator with a $-$ sign and thus get

$$\gamma_{-1,k} := \lambda_{0,2k+1} - \mathcal{P}(\lambda_{-1,k}) \tag{2}$$

The wavelet subset now encodes how much data deviates from the model on which $\mathcal{P}$ was built. If the signal is somehow correlated, the majority of the wavelet coefficients is small.

We now have more insight on how to split the original data set. Indeed, in order to get maximal data reduction from prediction, we need the subsets $\{\lambda_{-1,k}\}$ and $\{\gamma_{-1,k}\}$ to be maximally correlated. Cutting the signal into left and right parts might not be the best idea since values on the far left and the far right are hardly correlated. Predicting the right half of the signal based on the left is thus a tough job. A better method is to interlace the two sets, as we did in our previous step.

Now, we can replace the original data set with the smaller set $\{\lambda_{-1,k}\}$ and the wavelet set $\{\gamma_{-1,k}\}$. With a good prediction, the two subsets $\{\lambda_{-1,k}, \gamma_{-1,k}\}$ yield a more compact representation than the original set $\{\lambda_{0,k}\}$.

To find a good prediction operator, we again assume maximal correlation among neighboring samples. Hence, we simply suggest to predict an odd sample $\lambda_{0,2k+1}$ as the average of its two (even) neighbors: $\lambda_{-1,k}$ and $\lambda_{-1,k+1}$. The difference with this prediction scheme then becomes

$$\gamma_{-1,k} := \lambda_{0,2k+1} - 1/2(\lambda_{-1,k} + \lambda_{-1,k+1}).$$

The model used to build $\mathcal{P}$ is a function piecewise linear over intervals of length 2. If the original signal complies with the model, all wavelet coefficients in $\{\gamma_{-1,k}\}$ are zero. In other words, the wavelet coefficients measure to which extent the original signal *fails to be linear*. Their expected value is small. In terms of frequency content, the wavelet coefficients capture high frequencies present in the original signal.

We are quite happy with these wavelet coefficients. They encode the detail needed to go from the $\{\lambda_{-1,k}\}$ coefficients to the $\{\lambda_{0,k}\}$. They capture the high frequencies present in the original signal while the $\{\lambda_{-1,k}\}$ somehow capture the low frequencies. We can now iterate this scheme. We split $\{\lambda_{-1,k}\}$ into two subsets $\{\lambda_{-2,k}\}$ and $\{\gamma_{-2,k}\}$ (by subsampling) and then replace $\{\gamma_{-2,k}\}$ with the difference between $\{\gamma_{-2,k}\}$ and $\mathcal{P}(\lambda_{-2,k})$ (with our example, this would be as the failure to be linear). After n steps, we have replaced the original data with the wavelet representation $\{\lambda_{-n,k}, \gamma_{-n,k}, \cdots, \gamma_{-1,k}\}$. Given that the wavelet sets encode the difference with some predicted value based on a correlation model, this is likely to give a more compact representation.

2.3 Different prediction functions

The prediction does not necessarily have to be linear. We could try to find the *failure to be cubic* and any other higher order. This introduces us to the concept of *interpolating subdivision*.[12] We want to define an extension of our original sampled function to a function defined on the whole real line. We saw in the previous paragraphs how we used a recursive procedure for finding the value of a prediction (interpolating) function at every level.

We use some value N to denote the *order* of the subdivision (interpolation) scheme. For instance, to find a piecewise linear approximation, we use N equal to 2. To find a *cubic* approximation N should be equal to 4. It can be seen that N is important because it sets the smoothness of the interpolating function used to find the wavelet coefficients (high frequencies). We will refer to this function as the *dual wavelet* and to N as the number of *dual vanishing moments*.

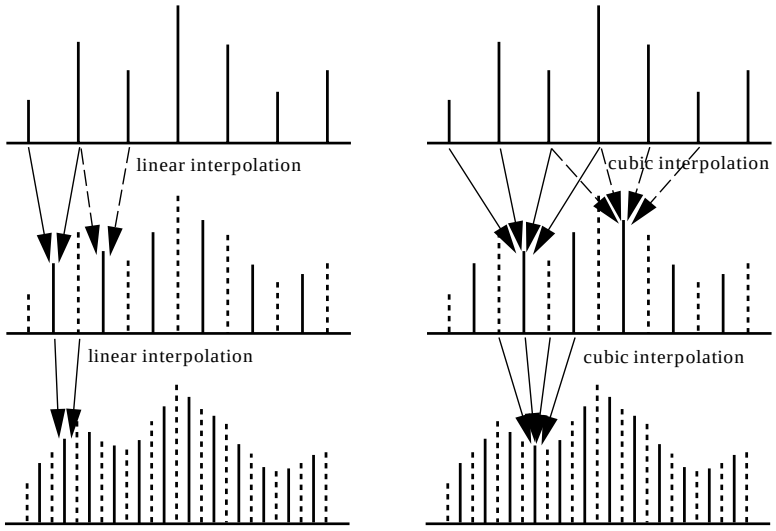

Figure 2: *Examples of Interpolating Subdivision.* On the left, a diagram showing the filling in of "in between" samples by linear interpolation between neighboring samples. On the right, the same idea is applied to higher order interpolation using two neighbors to either side and the unique cubic polynomial which interpolates these. This process is repeated ad infinitum to define the limit function.

Let us consider fancier interpolation schemes than the piecewise linear. Say, instead of defining the new value at the midpoint between two old values as a linear interpolation of the neighboring values, we can use two neighboring values on either side and define the (unique) cubic polynomial $p(x)$ which interpolates those four values

$$\lambda_{j,k-1} = p(x_{j,k-1}),\ \lambda_{j,k} = p(x_{j,k}),\ \lambda_{j,k+1} = p(x_{j,k+1}),\ \lambda_{j,k+2} = p(x_{j,k+2}).$$

The new sample value (odd index) will then be the value that this cubic polynomial takes on at the midpoint, while all odd samples (even index) are preserved

$$\lambda_{j+1,2k} = \lambda_{j,k}\ \lambda_{j+1,2k+1} = p(x_{j+1,2k+1}).$$

Figure 2 shows this process for linear and cubic interpolations in a diagram.

Even though each step in the subdivision involves cubic polynomials, the limit function will not be a polynomial anymore. While we do not have a sense yet as to what the limit function looks like, it is easy to see that it can reproduce cubic polynomials. Assume that the original sequence of samples came from some given cubic polynomial. In that case the interpolating polynomial over each set of 4 neighboring sample values will be the same polynomial and all newly generated samples will be on the original cubic polynomial, in the limit reproducing it. In general, we use N (N even) samples and build polynomials of degree N-1. We then say that the *order* of the subdivision scheme is N.

The prediction function $\mathcal{P}$, thus, uses polynomial interpolation of order $N-1$ to find the predicted values. The higher the order of this function, the better approximation of the γ coefficients based on the λ coefficients.[12] This is good if we know that the original data set resembles some polynomial of order $N-1$, as we said before. Then, the $\{\gamma_{j,k}\}$ set is going to be zero or very small, for there is almost no difference between the original data and the predicted values.

What makes interpolating subdivision so attractive from an implementation point of view is that we only need a routine which can construct an interpolating polynomial given some numbers and locations. The new sample

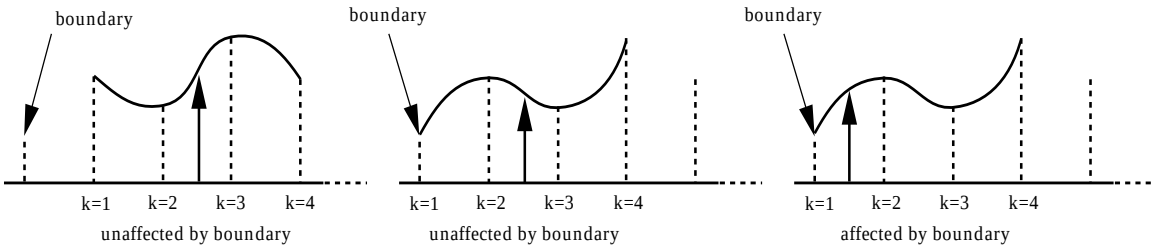

Figure 3: *Behavior of the Cubic Interpolating Subdivision near the boundary.* The midpoint samples between $k = 2, 3$ and $k = 1, 2$ are unaffected by the boundary. When attempting to compute the midpoint sample for the interval $k = 0, 1$, we must modify the procedure since there is no neighbor to the left for the cubic interpolation. Instead, we choose three neighbors to the right. Note how this results in the same cubic polynomial as used in the definition of the midpoint value $k = 1, 2$. The procedure clearly preserves the cubic reconstruction property even at the interval boundary and is thus the natural choice for the boundary modification.

value is then simply given by the evaluation of this polynomial at the new, refined location. The algorithm that best adapts to our interpolating subdivision scheme is Neville's algorithm.[7,9] Notice also that nothing in the definition of this procedure requires the original samples to be located at integers. We can use this feature to define scaling functions over irregular subdivisions,[12] which is not part of the scope of this paper.

This interpolation scheme allows us to easily accommodate interval boundaries for finite sequences, such as our case. For the cubic construction described above, we can take 1 sample on the left and 3 on the right at the left boundary of an interval. The cases are similar if we are at the right boundary. As soon as we start calculating new γ values near the right boundary, we begin having less λ coefficients on the right side and more on the left. If the γ coefficient is at the right boundary, we don't have any λ coefficients on the right. All of them will be on the left. A list of all the possible cases when $N = 4$ is the following:

Case 1: Near Left Boundary: More λ coefficients on the right side of the γ coefficient than on the left side.

- 1 λ on the left and 3 λ's on the right (remember that due to the splitting, we always have a λ in the first position)

Case 2: Middle: Enough λ coefficients on either side of the γ coefficient.

- 2 λ's on the left and 2 λ's on the right

Case 3: Near Right Boundary: More λ coefficients on the left side of the γ coefficient than on the right side.

- 3 λ's on the left and 1 λ on the right
- 4 λ's on the left and 0 λ's on the right

Figure 3 shows the cases where the γ coefficient is near the left boundary for a cubic interpolating subdivision ($N = 4$).

Using our interpolation scheme and Neville's algorithm, we generate a set of coefficients that will help us find the correct approximation using a function of order $N - 1$. For example, if $N = 2$, then we need two coefficients for the two possible cases (one λ on the left and one on the right, and 2 λ's on the left and none on the right). If $N = 4$, we need four coefficients for each one of the four cases as mentioned. These coefficients will be called *filter coefficients*.

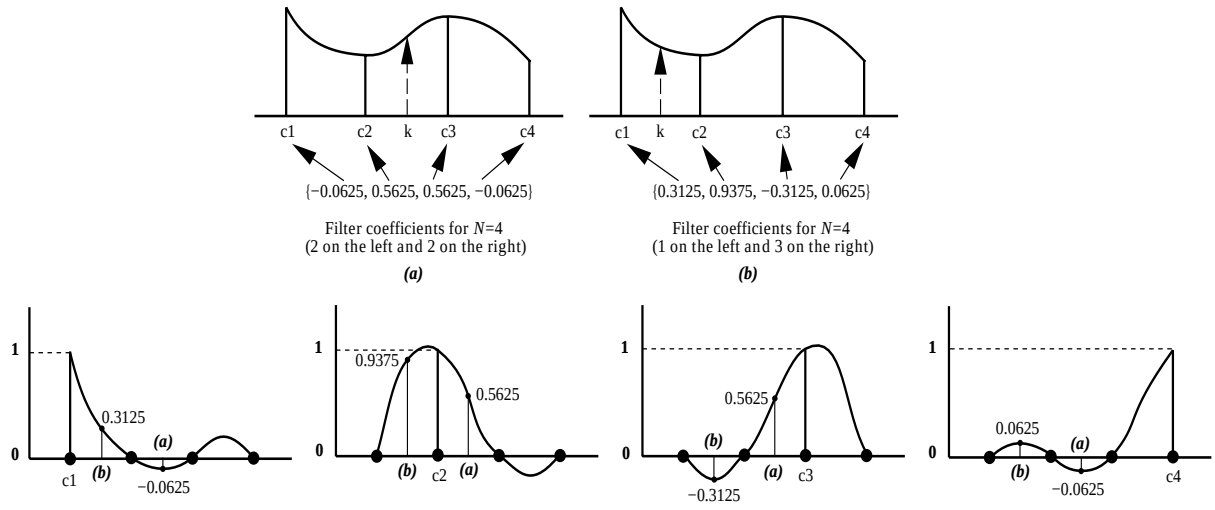

Figure 4: *How to find the Filter Coefficients ($N = 4$).* To find the value of a filter coefficient, we put that value to one and all the others to zero. We construct the polynomial corresponding to that configuration and evaluate the function at the positions we are interested in.

Due to symmetry,[12] we know that all the cases on the right side are the opposite to the cases on the left side. For example, the coefficients used for the case "3 λ's on the left and 1 λ on the right" are the same as the ones used in the case "1 λ on the left and 3 λ's on the right", but in opposite order. Thus, we have a total of $N/2 + 1$ different cases (one for the middle case and $N/2$ for the symmetric boundary cases).

An example of how to calculate the coefficients for a cubic interpolation is shown in Figure 4. We want to find a set of coefficients for the cases shown in Figure 3. That is, when there are two λ's on either side and when there are one λ on the left and three λ's on the right. They are referred as cases *(a)* and *(b)*.

Since there is a *unique* cubic interpolation (it does not matter how separated the samples are, we always have the same interpolating cubic function), we want to know the set of coefficients that will help us to predict any γ every time we have a case such as *(a)* or *(b)*.

The basic idea is the following: N is equal to 4; therefore, we will have 4 coefficients for every case. If we want to calculate, say, *c1*, we would put its value to 1 and all the other three, *c2, c3* and *c4*, to zero. We construct the polynomial that best fits what we have (in this case, a cubic polynomial) and start evaluating the function at the points we are interested in. For case *(a)*, we want to evaluate the function where we have two coefficients to the left and two to the right. For case *(b)*, we evaluate the function where there is one coefficient on the left and three on the right. The procedure is the same with the other coefficients. Tables 1 and 2 list the filter coefficients needed for the interpolation with $N = 2$ and 4. One property of these filter coefficients is that *every set of N coefficients for every case adds up to 1.*

The prediction phase thus gets reduced to a lookup in the previous tables in order to be able to calculate the wavelet coefficients. For example, if we want to predict a γ value using $N = 4$ and three λ coefficients on the left and one λ on the right, we would perform the following operation

$$\gamma_{-j,k} := \lambda_{-j+1,k} - (0.0625 * \lambda_{-j,k-3} - 0.3125 * \lambda_{-j,k-2} + 0.9375 * \lambda_{-j,k-1} + 0.3125 * \lambda_{-j,k+1}).$$

The prediction of other γ coefficients would be a similar process except that we would use the neighboring λ coefficients and the filter coefficients corresponding to the case under study.

Cases		*Coefficients*			
# λ's on left	# λ's on right	$k-3$	$k-1$	$k+1$	$k+3$
0	2			-0.5	1.5
1	1		0.5	0.5	
2	0	1.5	-0.5		

Table 1: *Filter Coefficients for* $N = 2$. Note that the first case would not happen with our splitting method: the first coefficient is always a λ.

Cases		*Coefficients*							
# λ's on left	# λ's on right	$k-7$	$k-5$	$k-3$	$k-1$	$k+1$	$k+3$	$k+5$	$k+7$
0	4					2.1875	-2.1875	1.3125	-0.3125
1	3				0.3125	0.9375	-0.3125	0.0625	
2	2			-0.0625	0.5625	0.5625	-0.0625		
3	1		0.0625	-0.3125	0.9375	0.3125			
4	0	-0.3125	1.3125	-2.1875	2.1875				

Table 2: *Filter Coefficients for* $N = 4$. Note the symmetry of the coefficients at the boundaries. Remember that the first case does not occur with the splitting method we are using.

We mentioned before that we were quite happy with the wavelet coefficients we are able to calculate. However, we are not very pleased with the choice of the $\{\lambda_{-1,k}\}$. The reason is the following. Suppose we are given $2^n + 1$ original samples $\{\lambda_{0,k} \mid 0 \leq k \leq 2^n\}$. We could apply our scheme (split and predict) n times thus obtaining $\{\gamma_{j,k} \mid -n \leq j \leq -1, 0 \leq k < 2^{n+j}\}$ and two (coarsest level) coefficients $\lambda_{-n,0}$ and $\lambda_{-n,1}$. These are the first ($\lambda_{-n,0} = \lambda_{0,0}$) and the last ($\lambda_{-n,1} = \lambda_{0,2^n}$) original samples. This introduces considerable aliasing. We would like some global properties of the original data set to be maintained in the smaller version $\{\lambda_{-j,k}\}$. For example, in the case of an image, we would like the smaller images $\{\lambda_{-j,k}\}$ to have the same overall brightness, i. e., the same average pixel value. Therefore, we would like to have the last values be the average of all the pixel values in the original image. We can solve part of this problem by introducing a third stage: the update.

2.4 Update phase

We want to *lift* the $\lambda_{-1,k}$ with the help of the wavelet coefficients $\gamma_{-1,k}$. Again we use the neighboring wavelet coefficients. The idea is to find a better $\lambda_{-1,k}$ so that a certain scalar quantity $Q()$, e. g., the mean, is preserved, or

$$Q(\lambda_{-1,k}) = Q(\lambda_{0,k}).$$

We could do so by finding a new operator to extract $\lambda_{-1,k}$ directly from $\lambda_{0,k}$, but we decide not to for two reasons. First, this would create a scheme which is very hard to invert. Secondly, we would like to reuse the work already done maximally. Therefore, we propose to use the already computed wavelet set $\{\gamma_{-1,k}\}$ to update $\{\lambda_{-1,k}\}$ so that the latter preserves $Q()$. In other words, we construct an operator $\mathcal{U}$ and update $\{\lambda_{-1,k}\}$ as

$$\lambda_{-1,k} := \lambda_{-1,k} + \mathcal{U}(\gamma_{-1,k}). \tag{3}$$

We are interested in finding a *scaling function*[12] using the previously calculated wavelet coefficients in order to maintain some properties among all the λ coefficients throughout all levels. One way is to set all $\{\lambda_{0,k}\}$ to zero except for $\lambda_{0,0}$ which is set to one. Then, we run the interpolating subdivision ad infinitum. The resulting function is $\varphi(x)$, the scaling function, which will help us create a *real* wavelet that will maintain some desired properties from the original signal. This function will have an order depending of some (even) value $\tilde{N}$, which is not necessarily equal to N. We will call $\tilde{N}$ the number of *real vanishing moments*. The higher the order of this function, the less aliasing effect we see in the resulting transform.

The basic properties we want to preserve are the moments of the wavelet function, ψ, at every level. One of the things we know from the properties of wavelets, is that the integral of ψ along the real line from $-\infty$ to ∞ must be equal to zero. This is also true for higher moments. Thus,

$$\int \psi(x)dx = 0, \quad \int x\psi(x) = 0, \quad \int x^2\psi(x) = 0, \quad \cdots, \quad \int x^{\tilde{N}-1}\psi(x) = 0.$$

We basically want to preserve up to $\tilde{N}-1$ moments of the λ's at every level and use this information to see how much of every γ coefficient is needed to update every λ. These update values are named *lifting coefficients*. Before starting the algorithm to calculate the lifting coefficients, we first need to initialize the moments information for every coefficient at the first level. The integral is set to one for all the coefficients because all the filter coefficients for each λ add to one (see section 2.3). The initial values for the higher order moments are calculated using the coefficient indices as shown in the table below.

Initial Moments using index k							
$Mom_1 = k$	0	1	2	3	4	5	$\cdots$
$Mom_2 = k^2$	0	1	4	9	16	25	
$Mom_3 = k^3$	0	1	8	27	64	125	
$\vdots$							

Once the moments are initialized, we can apply the following steps.

1. Check for the λ's that contributed to predicting every γ and see how much this contribution was. (These values are given by the filter coefficients found in the prediction stage.)

2. Update the moments for every λ at the current level with the following equation,

$$m_{j,k} := m_{j,k} + f_j * m_{l,k}, \tag{4}$$

   where j is the index relative to a λ coefficient, $f(j)$ is its corresponding filter coefficient $(0 < j \leq N)$, k is the moment being updated $(0 < k \leq \tilde{N})$, and l is the index relative to a γ coefficient.

3. Knowing that all the moments must be zero at every level, we can construct a linear system to find the lifting coefficients for every γ. The steps are:

   (a) Put a one in a γ coefficient and zero in all the remaining γ's.

   (b) Apply an inverse transform one step up to see how much this γ is contributing to the λ's that update it and create a linear system of $\tilde{N} \times \tilde{N}$ variables as follows,

$$\begin{aligned}
&c_1 * m_{l_0,0} + c_2 * m_{l_1,0} + c_3 * m_{l_2,0} + \cdots + c_{\tilde{N}} * m_{l_N,0} = m_{g_j,0} \\
&c_1 * m_{l_0,1} + c_2 * m_{l_1,1} + c_3 * m_{l_2,1} + \cdots + c_{\tilde{N}} * m_{l_N,1} = m_{g_j,1} \\
&c_1 * m_{l_0,2} + c_2 * m_{l_1,2} + c_3 * m_{l_2,2} + \cdots + c_{\tilde{N}} * m_{l_N,2} = m_{g_j,2} \\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
&c_1 * m_{l_0,\tilde{N}} + c_2 * m_{l_1,\tilde{N}} + c_3 * m_{l_2,\tilde{N}} + \cdots + c_{\tilde{N}} * m_{l_L,\tilde{N}} = m_{g_j,\tilde{N}},
\end{aligned} \tag{5}$$

   where c_i $(0 < i \leq \tilde{N})$ are the lifting coefficients to be found, l_i $(0 < i \leq N)$ is the index relative to the contributing λ coefficients at the current level, g_j $(0 < j \leq signal\_length)$ is the index relative to the γ with value set to one.

   (c) Solve the system and find the set of lifting coefficients for the γ coefficient with value set to one.

   This linear system is built and solved for every γ coefficient to find its corresponding set of lifting coefficients.

After applying the previous steps, we have a set of $\tilde{N}$ lifting coefficients for every γ at every level. These values are used to apply the update operator, $\mathcal{U}$, to the λ coefficients before iterating to the next level. To update the λ's, position yourself at a $\gamma_{-j,k}$ coefficient and take its corresponding lifting coefficients, e. g. (a,b) for $\tilde{N} = 2$. Identify the λ's which were affected by this γ, e. g. $\lambda_{-j,k-1}$ and $\lambda_{-j,k+1}$. Now,

$$\lambda_{-j,k-1} := \lambda_{-j,k-1} + a * \gamma_{-j,k}, \quad \text{and} \quad \lambda_{-j,k+1} := \lambda_{-j,k+1} + b * \gamma_{-j,k}.$$

Then, move to the next γ and do the same.

An example of the split, predict and update phases for a 1-D signal with length $L = 8$, $N = 2$ and $\tilde{N} = 2$ follows. First, consider the split and predict:

$$\begin{array}{cccccccc} \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 & \lambda_6 & \lambda_7 & \lambda_8 \\ & \gamma_1 & & \gamma_2 & & \gamma_3 & & \gamma_4 . \end{array}$$

γ_1 uses λ_1 and λ_3 for prediction. Similarly, γ_2 uses λ_3 and λ_5, γ_3 uses λ_5 and λ_7, and γ_4 uses λ_5 and λ_7. The second stage, the update, is performed using equation 4 and 5. In this example, the following lifting coefficients ((a,b) pairs) are obtained,

$$\begin{array}{ccccccc} & \gamma_1 & & \gamma_2 & & \gamma_3 & & \gamma_4 \\ & (2/5,1/5) & & (0,2/3) & & (4/15,1/5) & & (-2/15,2/5) \\ \lambda_1 & & \lambda_3 & & \lambda_5 & & \lambda_7 . & \end{array}$$

λ_1 uses a from γ_1 for updating. Similarly, λ_3 uses b from γ_1 and a from γ_2; λ_5 uses b from γ_2 a from γ_3, and a from γ_4; and λ_7 uses b from γ_3 and b from $\gamma 4$.

At the next level, the coefficients get organized as follows after the split and predict stages.

$$\begin{array}{cccc} \lambda_1 & \lambda_3 & \lambda_5 & \lambda_7 \\ & \gamma_1 & & \gamma_2 \end{array}$$

γ_1 uses λ_1 and λ_5, and γ_2 uses λ_1 and λ_5 for prediction. In the update stage,

$$\begin{array}{ccccc} & \gamma_1 & & \gamma_2 \\ & (1/2,0.214286) & & (-1/3,0.476190) \\ \lambda_1 & & \lambda_5 , & \end{array}$$

λ_1 uses a from γ_1 and a from γ_2, and λ_5 uses b from γ_1 and b from γ_2 for updating.

It is important to note that for a longer signal, the lifting coefficients are going to be (1/4,1/4) for all the λ's unaffected by the boundaries. Using these values, we can update the λ coefficients with the following equation,

$$\lambda_{-1,k} := \lambda_{-1,k} + 1/4 * \gamma_{-1,k-1} + 1/4 * \gamma_{-1,k} .$$

The three stages of lifting described by Equations 1, 2, and 3 and depicted in the block diagram in figure 1 are combined and iterated to generate the 1-D fast lifted forward wavelet transform algorithm:

$$\texttt{For j = -1 downto -n:} \begin{cases} \{\lambda_{j,k}, \gamma_{j,k}\} & := & \texttt{Split}(\lambda_{j+1,k}) \\ \gamma_{j,k} & -= & \mathcal{P}(\lambda_{j,k}) \\ \lambda_{j,k} & += & \mathcal{U}(\gamma_{j,k}). \end{cases}$$

We can now illustrate one of the nice properties of lifting: once we have the forward transform, we can immediately derive the inverse. We just have to reverse the operations and toggle + and −. This leads to the following algorithm for the inverse transform:

$$\texttt{For j = -n to -1:} \begin{cases} \lambda_{j,k} & -= & \mathcal{U}(\gamma_{j,k}) \\ \gamma_{j,k} & += & \mathcal{P}(\lambda_{j,k}) \\ \{\lambda_{j+1,k}\} & := & \texttt{Join}(\lambda_{j,k}, \gamma_{j,k}). \end{cases}$$

To calculate the total number of iterations of this transform, three factors have to be considered: the signal length (L), the number of dual vanishing moments (N), and the number of real vanishing moments ($\tilde{N}$). It can be proven that the total number of iterations is given by,

$$n = \left\lfloor \log_2 \left(\frac{L-1}{N_{max}-1} \right) \right\rfloor , \tag{6}$$

where $N_{max} = \max(N, \tilde{N})$. It can be seen from Equation 6 that the size of the signal does not matter, i. e., signals do not necessarily have to have dyadic dimensions. The interpolating subdivision guarantees correct treatment of the boundaries for every case.

An extension of the 1-D algorithm for 2-D signals is a simple repetitive scheme of the 1-D transform through rows and columns, as the transform is separable. For better support of frequencies, we propose the application of the "square 2-D" method. The basic idea is to apply the 1-D transform to all the rows first and, afterwards, to all the columns. This is done at every level in order to create a square window transform that gives better frequency support than a rectangular window transform. We would also have different filter and lifting coefficients for each dimension (X,Y) if they are different. Equation 6 is updated simply by setting $L = \max(L_x, L_y)$, i. e., the maximum between the two dimensions.

Using the filter coefficients (1/2,1/2) and lifting coefficients (1/4,1/4), the wavelet transform presented here is the $(N = 2, \tilde{N} = 2)$ biorthogonal wavelet transform of Cohen-Daubechies-Feauveau.[3] This simple example already shows how the lifting scheme can speed up the implementation of the wavelet transform. Classically, the $\{\lambda_{-1,k}\}$ coefficients are found as the convolution of the $\{\lambda_{0,k}\}$ coefficients with the filter $\tilde{h} = \{-1/8, 1/4, 3/4, 1/4, -1/8\}$. This step would take 6 operations per coefficient while lifting only needs 3.

A whole family of biorthogonal wavelets can be constructed by varying the three stages of lifting:

1. *Split*: Choices other than the Lazy wavelet are possible as the initial split. A typical alternative is the Haar wavelet transform.[12]
2. *Predict*: In wavelet terminology, the prediction step establishes the number of vanishing moments (N) of the dual wavelet. In other words, if the original signal is a polynomial of degree less than N, all wavelet coefficients will be zero. It is shown that schemes with order higher than $N = 2$ are easily obtained by involving more neighbors.
3. *Update*: Again in wavelet terminology, we said that the update step establishes the number of vanishing moments ($\tilde{N}$) of the primal or real wavelet. In other words, the transform preserves the first $\tilde{N}$ moments of the $\lambda_{j,k}$ sequences. It is shown that schemes with order higher than $\tilde{N} = 2$ can be constructed by involving more neighbors. In some cases, namely when the split stage already creates a wavelet with a vanishing moment (such as the Haar), the update stage can be omitted.[12] In other cases, with more than one update step applied, another family of wavelets, rather than the biorthogonal ones, can be created.

The in-place implementation can be easily derived from the diagrams and equations. Let's assume the original samples are stored in a vector $v[k]$. Each coefficient $\lambda_{j,k}$ or $\gamma_{j,k}$ is stored in location $v[2^{-j}\ k]$. The Lazy wavelet transform is then immediate. All other operations can be done with $+=$ or $-=$ operations. In other words, when predicting the γ coefficients, the λ coefficients can be substituted by them in the same position. When updating the λ coefficients, they can be saved in the same position. See Figure 5.

3 RESULTS

The versatility of the lifting scheme for wavelet-related implementations is that we can adapt the split, prediction, and update stages to the type of dataset we are working with. This feature has proven to be effective for edge detection,[5] pattern recognition,[6] and texture mapping,[8] among other examples.

Blur and enhance operations are common tasks when manipulating images. Many algorithms use a mask to implement the filter. This approach only modifies the raw data and, therefore, does not offer flexibility with the amount of information we want to attenuate or enhance. A wavelet approach makes more sense, because we have more control of the frequency and time values along different resolutions.

A direct and easy way to implement enhance and blur operators using lifting is by modifying the wavelet coefficients (γ) after the in-place transform. This modification consists of a multiplication by a factor β^j, where j is the current level in the wavelet transform. The initial value of β sets the type of operation done to the original image. If $\beta < 1$, then the result is blurred. If $\beta > 1$, then the result is enhanced. Figure 6*(a)* shows an example

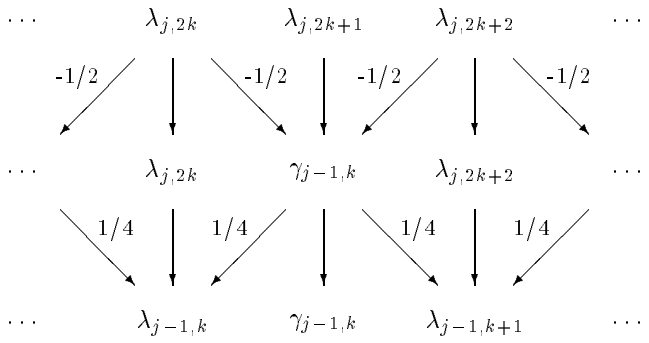

Figure 5: *The Lifting Scheme (in-place calculation $N = 2, \tilde{N} = 2$).* Split, calculate the wavelet coefficients $\gamma_{-1,m}$ as the failure to be linear, and use them to update the $\lambda_{-j-1,k}$.

image. The blurred version of it with $\beta = 0.8$ is shown in figure 6*(b)*. The enhanced version with $\beta = 1.2$ is shown in Figure 6*(c)*.

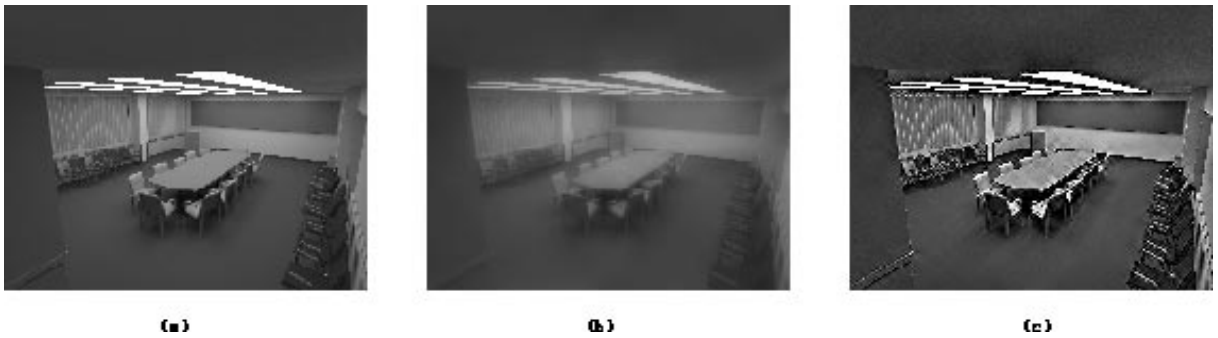

Figure 6: *Sample Image (720×576) for Processing.* Figure *(a)* shows the original image. Figure *(b)* shows the result of a blur operation with $\beta = 0.8$. Figure *(c)* shows the result of an enhance operation with $\beta = 1.2$.

3.1 LIFTPACK software

LIFTPACK is in Beta testing stage and has been proven to run under several platforms. Timings and other specific issues about the implementation can be found in the LIFTPACK Home Page at

`http://www.cs.sc.edu/~fernande/liftpack/`.

4 CONCLUSIONS

An efficient and fast method to generate wavelets on the interval was implemented. Since the checkpoint of the algorithm is the number of λ coefficients available at the moment and not the actual size of the step with respect to the size of the direction (either X or Y), issues like dataset size and shape of region of work are treated with efficiency. Therefore, the dataset size does not have to be necessarily a power of two (dyadic) and the region does not have to be square. In fact, by defining a set of boundary cases, the lifting scheme can be adapted to work under regions of any shape.

Acknowledgment

We would like to thank Dr. Peter Schröder and all the beta testers for their suggestions to create a quality and optimal code. We also would like to thank Alexander Keller for donating the sample image for testing purposes.

5 REFERENCES

[1] A. Aldroubi and M. Unser. Families of multiresolution and wavelet spaces with optimal properties. *Numer. Funct. Anal. Optim.*, 14:417–446, 1993.

[2] C. K. Chui, *An Introduction to Wavelets*, Academic Press, San Diego, CA, 1992.

[3] A. Cohen, I. Daubechies, and J. Feauveau, "Bi-orthogonal bases of compactly supported wavelets," *Comm. Pure Appl. Math.*, vol. 45, pp. 485–560, 1992.

[4] I. Daubechies, *Ten Lectures on Wavelets*, CBMS-NSF Regional Conf. Series in Appl. Math., Vol. 61, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[5] G. Fernández, *Multiresolution Stereo Analysis using the Lifting Scheme and Moments*, Master's thesis, University of South Carolina, Columbia, SC, 1996.

[6] S. Periaswamy, *Detection of Microcalcifications in Mammograms using Hexagonal Wavelets*, Master's thesis, University of South Carolina, Columbia, SC, 1996.

[7] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes*, Cambridge University Press, 2nd edn., 1993.

[8] P. Schröder and W. Sweldens, "Spherical wavelets: Efficiently representing functions on the sphere," *Computer Graphics, (SIGGRAPH '95 Proceedings)*, 1995.

[9] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer Verlag, New York, 1980.

[10] W. Sweldens, "The lifting scheme: A construction of second generation wavelets," Tech. Rep. 1995:6, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina, 1995, (`ftp://ftp.math.sc.edu/pub/imi_95/imi95_6.ps`).

[11] W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets," *Appl. Comput. Harmon. Anal.*, vol. 3(2), pp. 186–200, 1996.

[12] W. Sweldens and P. Schröder, "Building your own wavelets at home," Tech. Rep. 1995:5, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina, 1995, (`ftp://ftp.math.sc.edu/pub/imi_95/imi95_5.ps`).