Morning Section: Introductory Material

Building Your Own Wavelets at Home

Wim Sweldens and Peter Schröder

Chapter 1

First Generation Wavelets

1.1 Introduction

Wavelets have been making an appearance in many pure and applied areas of science and engineering. Computer graphics with its many and varied computational problems has been no exception to this rule. In these notes we will attempt to motivate and explain the basic ideas behind wavelets and what makes them so successful in application areas.

The main motivation behind the development of wavelets and the many related ideas (see Figure 1.1) was the search for *fast* algorithms to compute *compact* representations of functions and data sets. How can such compact representations be achieved? There are many approaches, some more computationally intensive, others less, but they all amount to exploiting structure in the data or underlying functions. Depending on the application area this goes by different names such as the exploitation of "structure," "smoothness," "coherence," or "correlation." Of course, for purely random signals or data no compact representations can be found. But most of the time we are interested in realistic data and functions which do exhibit some smoothness or coherence. In these cases wavelets and the fast wavelet transform turn out to be very useful tools.

While the name "wavelets" is relatively young (early 80's,) the basic ideas have been around for a long time in many areas from abstract analysis to signal processing and theoretical physics. The main contribution of the wavelet field as such has been to bring together a number of similar ideas from different disciplines and create synergy between these techniques. The result is a flexible and powerful toolbox of algorithmic techniques combined with a solid underlying

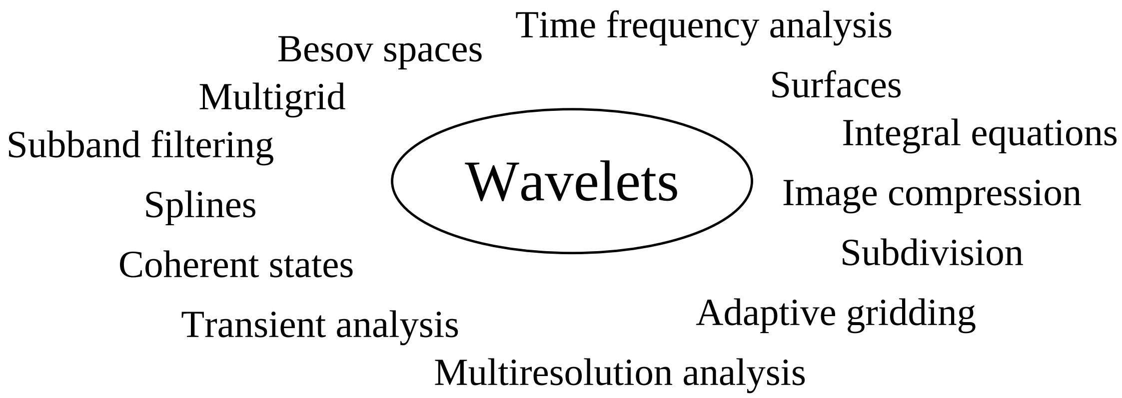

Figure 1.1: *Many areas of science, engineering, and mathematics have contributed to the development of wavelets. Some of these are indicated surrounding the center bubble.*

theory.

Because of the different "parents" of wavelets, there are many ways to motivate their construction and understand their properties. One example is subband filtering from the area of signal processing, where the aim is to decompose a given signal into frequency bands. In this case filter design and Fourier analysis are essential tools. Researchers in approximation theory and abstract analysis were interested in the characterization of function spaces defined through various notions of smoothness. Yet others were attempting to build approximate eigenfunctions for certain integral operators to enhance their understanding of the underlying structures.

Instead of retracing these developments we will focus primarily on the idea of coherence, or smoothness, and its exploitation to motivate and derive the wavelet transform together with a large class of different wavelets. The tool that we use to build wavelets transforms is called the *lifting scheme* [27, 26]. The main feature of the lifting scheme is that all constructions are derived in the *spatial domain*. This is in contrast to the traditional approach, which relies heavily on the frequency domain. Staying in the spatial domain leads to two major advantages. First, it does not require the machinery of Fourier analysis as a prerequisite. This leads to a more intuitively appealing treatment better suited to those interested in applications, rather than mathematical foundations. Secondly, lifting leads to algorithms that can easily be generalized to complex geometric situations which typically occur in computer graphics. This will lead to so-called "Second Generation Wavelets."

The lifting scheme was developed in 1994, but has numerous connections with earlier developments and can even be traced back all the way to the Euclidean algorithm! The development of lifting was inspired by earlier work of Lounsbery et al. concerning wavelet transforms of meshes [18] and work of Donoho concerning interpolating wavelet transforms [13]. Both these developments are special cases of lifting. Lifting is also closely related to filter bank constructions of Vetterli and Herley [28] and local decompositions of Carnicer, Dahmen and Peña [2].

To make the treatment as accessible as possible we will take a very "nuts and bolts," algorithmic approach. In particular we will initially ignore many of the mathematical details and introduce the basic techniques with a sequence of examples. Other sections will be devoted to more formal and rigorous mathematical descriptions of the underlying principles. These sections are marked with an asterisk and can be skipped on first reading.

In this first chapter, we treat the classical or "First Generation Wavelets." We begin with a simple example of a wavelet transform to introduce the basic ideas. Later we introduce lifting in general and move on to the mathematical background. The second chapter is concerned with generalizations to more complex geometries and "Second Generation Wavelets."

A word of caution is in order before we dive in the wavelet sea. Some readers might be familiar with other overview or tutorial material concerning wavelets. In most cases these expositions use the classical frequency domain framework. Since we are staying entirely in the spatial domain our exposition may initially look rather foreign. This is due to the fact that our approach relies entirely on the new lifting philosophy. However, we assure the reader that by the end of the first chapter the connections between lifting and the classical treatment will be apparent. We hope that as a result of this approach the reader will gain new insight into what makes wavelets "tick."

1.2 A Simple Example: The Haar Wavelet

Consider two numbers a and b and think of them as two neighboring samples of a sequence. So a and b have some correlation which we would like to take advantage of. We propose a well-known, simple linear transform which replaces a and b by their *average* s and *difference* d:

$$\begin{aligned} s &= \frac{a+b}{2} \\ d &= b-a. \end{aligned} \tag{1.1}$$

The idea is that if a and b are highly correlated, the expected absolute value of their difference d will be small and can be represented with fewer bits. In case that $a = b$ the difference is simply zero. We have not lost any information because given s and d we can always recover a and b as:

$$\begin{aligned} a &= s - d/2 \\ b &= s + d/2. \end{aligned}$$

These reconstruction formulas can be found by inverting a 2×2 matrix.

This simple observation is the key behind the so-called *Haar* wavelet transform. Consider a signal s_n of 2^n sample values $s_{n,l}$:

$$s_n = \{s_{n,l} \mid 0 \leq l < 2^n\}.$$

Apply the average and difference transform onto each pair $a = s_{2l}$ and $b = s_{2l+1}$. There are 2^{n-1} such pairs ($l = 0 \ldots 2^{n-1}$); denote the results by $s_{n-1,l}$, and $d_{n-1,l}$:

$$\begin{aligned} s_{n-1,l} &= \frac{s_{n,2l} + s_{n,2l+1}}{2} \\ d_{n-1,l} &= s_{n,2l+1} - s_{n,2l}. \end{aligned}$$

The input signal s_n, which has 2^n samples, is split into two signals: s_{n-1} with 2^{n-1} averages $s_{n-1,l}$ and d_{n-1} with 2^{n-1} differences $d_{n-1,l}$. Given the averages s_{n-1} and differences d_{n-1} one can recover the original signal s_n.

We can think of the averages s_{n-1} as a coarser resolution representation of the signal s_n and of the differences d_{n-1} as the information needed to go from the coarser representation back to the original signal. If the original signal has some local coherence, e.g., if the samples are values of a smoothly varying function, then the coarse representation closely resembles the original signal and the detail is very small and thus can be represented efficiently.

We can apply the same transform to the coarser signal s_{n-1} itself. By taking averages and differences, we can split it in a (yet) coarser signal s_{n-2} and another difference signal d_{n-2} where each of them contain 2^{n-2} samples. We can do this n times before we run out of samples, see Figure 1.2. This is the *Haar* transform. We end up with n detail signals d_j with $0 \leq j \leq n-1$, each with 2^j coefficients, and one signal s_0 on the very coarsest scale. The coarsest level signal s_0 contains only one sample $s_{0,0}$ which is the average of all the samples of the original signal, i.e., it is the DC component or zero frequency of the signal. By using the inverse transform we start

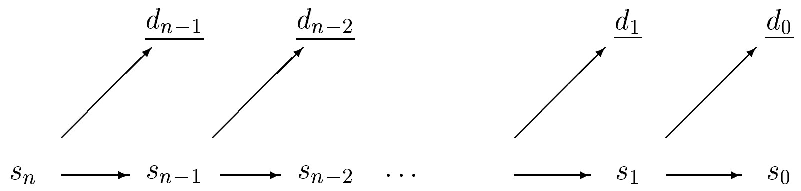

Figure 1.2: *Structure of the wavelet transform: recursively split into averages and differences.*

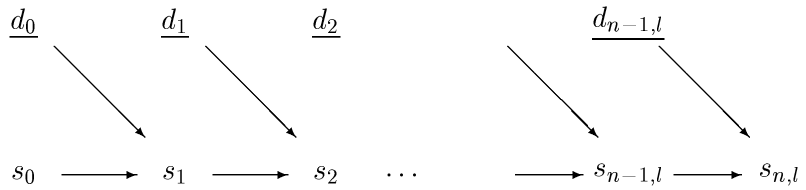

Figure 1.3: *Structure of the inverse wavelet transform: recursively merge averages and differences.*

from s_0 and d_j for $0 \leq j < n$ and obtain s_n again. Note that the total number of coefficients after transform is 1 for s_0 plus 2^j for each d_j. This adds up to

$$1 + \sum_{j=0}^{n-1} 2^j = 2^n,$$

which is exactly the number of samples of the original signal. The whole Haar transform can be thought of as applying a $N \times N$ matrix ($N = 2^n$) to the signal s_n. The cost of computing the transform is only proportional to N. This is remarkable as in general a linear transformation of an N vector requires $O(N^2)$ operations. Compare this to the Fast Fourier Transform, whose cost is $O(N \log N)$. It is the hierarchical structure of a wavelet transform which allows switching to and from the wavelet representation in $O(N)$ time.

1.3 Haar and Lifting

In this section we propose a new way of looking at the Haar transform. The novelty lies in the way we compute the difference and average of two numbers a and b. Assume we want to compute the whole transform in-place, i.e., without using auxiliary memory locations, by overwriting the locations that hold a and b with the values of respectively s and d. This can not immediately be done with the formulas of (1.1). Indeed, assume we want to store s in the same location as a and d in the same location as b. Then the formulas (1.1) would lead to the wrong result. Computing s and overwriting a leads to a wrong d (assuming we compute the average *after* the difference.) We therefore suggest an implementation in two steps. First we only compute the difference:

$$d = b - a,$$

and store it in the location for b. As we now lost the value of b we next use a and the newly computed difference d to find the average as:

$$s = a + d/2.$$

This gives the same result because $a + d/2 = a + (b - a)/2 = (a + b)/2$. The advantage of the splitting into two steps is that we can overwrite b with d and a with s, requiring no auxiliary storage. A C-like implementation is given by

```
b -= a; a += b/2;
```

after which b contains the difference and a the average. The computations can be done in-place. Moreover we can immediately find the inverse without formally solving a 2×2 system: simply run the above code backwards! (i.e., change the order and flip the signs.) Assume a contains the average and b the difference. Then

```
a -= b/2; b += a;
```

recovers the values a and b in their original memory locations. This particular scheme of writing a transform is a first, simple instance of the *lifting scheme*.

1.4 The Lifting Scheme

In this section we describe the lifting scheme in more detail. Consider a signal s_j with 2^j samples which we want to transform into a coarser signal s_{j-1} and a detail signal d_{j-1}. A typical case

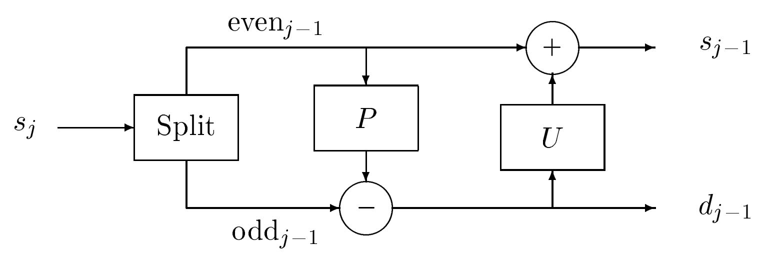

Figure 1.4: *The lifting scheme, forward transform: first compute the detail as the failure of a prediction rule, then use that detail in an update rule to compute the coarse signal.*

of a wavelet transform built through lifting consists of three steps: split, predict, and update. Let us discuss each stage in more detail.

- **Split:** This stage does not do much except for splitting the signal into two disjoint sets of samples. In our case one group consists of the even indexed samples s_{2l} and the other group consists of the odd indexed samples s_{2l+1}. Each group contains half as many samples as the original signal. The splitting into even and odds is a called the *Lazy wavelet transform*. We thus built an operator so that

$$(\text{even}_{j-1}, \text{odd}_{j-1}) := \text{Split}(s_j)$$

  Remember that in the previous example a was an even sample while b was an odd sample.

- **Predict:** The even and odd subsets are interspersed. If the signal has a local correlation structure, the even and odd subsets will be highly correlated. In other words given one of the two sets, it should be possible to predict the other one with reasonable accuracy. We always use the even set to predict the odd one. In the Haar case the prediction is particularly simple. An odd sample $s_{j,2l+1}$ will use its left neighboring even sample $s_{j,2l}$ as its predictor. We then let the detail $d_{j-1,l}$ be the difference between the odd sample and its prediction:

$$d_{j-1,l} = s_{j,2l+1} - s_{j,2l},$$

  which defines an operator P such that

$$d_{j-1} = \text{odd}_{j-1} - P(\text{even}_{j-1}).$$

As we already argued, it should be possible to represent the detail more efficiently. Note that if the original signal is a constant, then all details are exactly zero.

- **Update:** One of the key properties of the coarser signals is that they have the same average value as the original signal, i.e., the quantity

$$S = 2^{-j} \sum_{l=0}^{2^j-1} s_{j,l}$$

is independent of j. This results in the fact that the last coefficients $s_{0,0}$ is the DC component or overall average of the signal. The update stage ensures this by letting

$$s_{j-1,l} = s_{j,2l} + d_{j-1,l}/2.$$

Substituting this definition we easily verify that

$$\sum_{l=0}^{2^{j-1}} s_{j-1,l} = \sum_{l=0}^{2^{j-1}} (s_{j,2l} + d_{j-1,l}/2) = 1/2 \sum_{l=0}^{2^{j-1}} (s_{j,2l} + s_{j,2l+1}) = 1/2 \sum_{l=0}^{2^j} s_{j,l},$$

which defines an operator U of the form

$$s_{j-1} = \text{even}_{j-1} + U(d_{j-1})$$

All this can be computed in-place: the even locations can be overwritten with the averages and the odd ones with the details. An abstract implementation is given by:

$$\begin{aligned} (\text{odd}_{j-1}, \text{even}_{j-1}) \quad &\texttt{:=} \quad \text{Split}(s_j); \\ \text{odd}_{j-1} \quad &\texttt{-=} \quad P(\text{even}_{j-1}); \\ \text{even}_{j-1} \quad &\texttt{+=} \quad U(\text{odd}_{j-1}); \end{aligned}$$

These three stages are depicted in a wiring diagram in Figure 1.4.

We can immediately build the inverse scheme, see the wiring diagram in Figure 1.5. Again we have three stages:

- **Undo update:** Given d_j and s_j we can recover the even samples by simply subtracting the update information:

$$\text{even}_{j-1} = s_{j-1} - U(d_{j-1}).$$

In the case of Haar, we compute this by letting

$$s_{j,2l} = s_{j-1,l} - d_{j-1,l}/2.$$

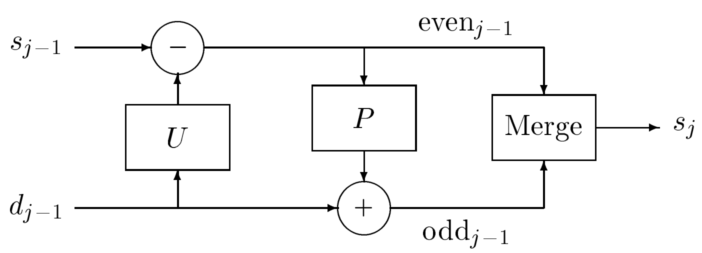

Figure 1.5: *The lifting scheme, inverse transform: first undo the update and recover the even samples, then add the prediction to the details and recover the odd samples.*

- **Undo predict:** Given even_{j-1} and d_{j-1} we can recover the odd samples by adding the prediction information
$$\text{odd}_{j-1} = d_{j-1} + P(\text{even}_{j-1}).$$
In the case of Haar, we compute this by letting
$$s_{n,2l+1} = d_{n-1,l} + s_{n,2l}.$$

- **Merge:** Now that we have the even and odd samples we simply have to zipper them together to recover the original signal. This is the inverse Lazy wavelet:
$$s_j = \text{Merge}(\text{even}_{j-1}, \text{odd}_{j-1}).$$

Assuming that the even slots contain the averages and the odd ones contain the difference, the implementation of the inverse transform is:

$$\begin{aligned} \text{even}_{j-1} \quad &\texttt{-=} \quad U(\text{odd}_{j-1}); \\ \text{odd}_{j-1} \quad &\texttt{+=} \quad P(\text{even}_{j-1}); \\ s_j \quad &\texttt{:=} \quad \text{Merge}(\text{odd}_{j-1}, \text{even}_{j-1}) \end{aligned}$$

The inverse transform is thus always found by reversing the order of the operations and flipping the signs.

The lifting scheme has a number of algoritmic advantages

- **In-place:** all calculations can be performed in-place which can be an important memory savings.

- **Efficiency:** in many cases the number of floating point operations needed to compute both smooth and detail parts is reduced since subexpressions are reused.

- **Parallelism:** "unrolling" a wavelet transform into a wiring diagram exhibits its inherent SIMD parallelism at all scales, with single write and multiple read semantics.

But perhaps more importantly lifting has some structural advantages which are both theoretically and pratically relevant:

- **Inverse Transform:** writing the wavelet transform as a sequence of elementary predict and update (lifting) steps, it is immediately obvious what the inverse transform is: simply run the code backwards. In the classical setting, the inverse transform can typically only be found with the help of Fourier techniques.

- **Generality:** this is the most important advantage. Since the design of the transform is performed without reference to Fourier techniques it is very easy to extend it to settings in which, for example, samples are not placed evenly or constraints such as boundaries need to be incorporated. It also carries over directly to curves, surfaces and volumes.

It is for these reasons that we built our exposition entirely around the lifting scheme.

1.5 The Linear Wavelet Transform

One way to build other wavelet transforms is through the use of different predict and/or update steps. What is the incentive behind improving predict and update? The Haar transform uses a predictor which is correct in case the original signal is a constant. It eliminates zeroth order correlation. We say that the order of the predictor is one. Similarly the order of the update operator is one as it preserves the average or zeroth order *moment*. In many cases it is desirable to have predictors which can exploit coherence beyond zeroth order correlation and similarly it is often desirable to preserve higher order moments beyond the zeroth in the successively coarser versions of the function.

In this section we build a predictor and update which are of order two. This means that the predictor will be exact in case the original signal is a linear and the update will preserve the average and the first moment. This turns out to be fairly easy. For an odd sample $s_{j,2l+1}$ we let the predictor be the average of the neighboring sample on the left ($s_{j,2l}$) and the neighboring

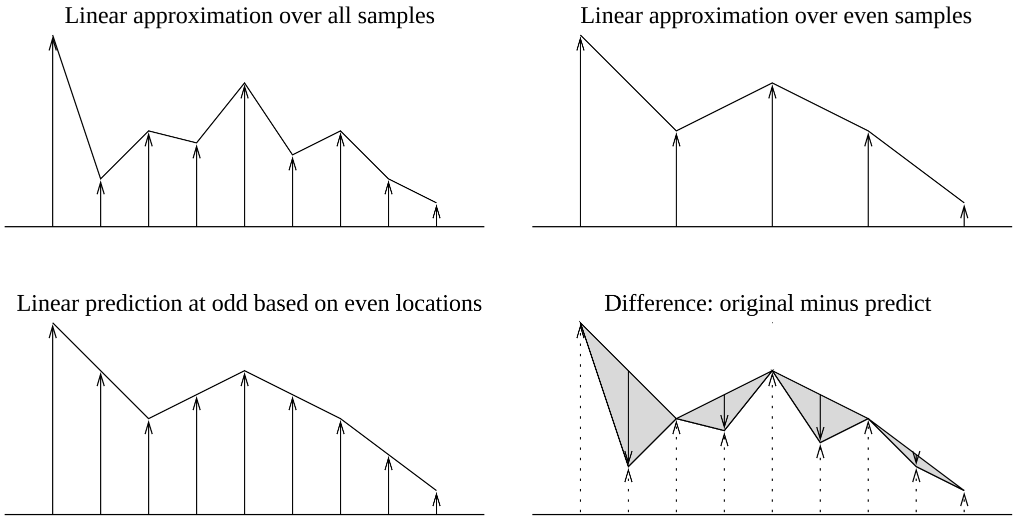

Figure 1.6: *Example of linear prediction. On the top left the original signal with a piecewise linear approximation. To its right is the coarser approximation based only on the even samples. Using the even samples to predict values at the odd locations based on a linear predictor is shown in the bottom left. The detail coefficients are defined as the difference between the prediction and the actual value at the odd locations (bottom right.) We may think of this as the failure of the signal to be locally like a first degree polynomial.*

sample on the right ($s_{j,2l+2}$.) The detail coefficient is given by

$$d_{j,l} = s_{j,2l+1} - 1/2(s_{j,2l} + s_{j,2l+2}).$$

Figure 1.6 illustrates this idea. Notice that if the original signal was a first degree polynomial, i.e., if $s_l = \alpha\, l + \beta$ for some α and β, this prediction is always correct and all details are zero, i.e., $N = 2$. In other words, the detail coefficients measure to which extent the original signal *fails to be linear*. The expected value of their magnitudes is small. In terms of frequency content, the detail coefficients capture high frequencies present in the original signal.

In the update stage, we first assure that the average of the signal is preserved or

$$\sum_l s_{j-1,l} \ = \ 1/2 \sum_l s_{j,l}.$$

We therefore update the even samples $s_{j,2l}$ using the previously computed detail signals $d_{j-1,l}$.

Again we use the neighboring wavelet coefficients and propose an update of the form:

$$s_{j-1,l} = s_{j,2l} + A\,(d_{j-1,l-1} + d_{j-1,l}).$$

To find A we compute the average:

$$\sum_l s_{j-1,l} = \sum_l s_{j,2l} + 2\,A \sum_l d_{j-1,l} = (1 - 2\,A) \sum_l s_{j,2l} + 2\,A \sum_l s_{j,2l+1}.$$

From this we get $A = 1/4$ as the correct choice to maintain the average. Because of the symmetry of the update operator we also preserve the first order moment or

$$\sum_l l\, s_{j-1,l} \;=\; 1/2 \sum_l l\, s_{j,l}.$$

One step in the wavelet transform is shown in the scheme in Figure 1.7. By iterating this scheme we get a complete wavelet transform. The inverse is as easy to compute, letting

$$s_{j,2l} = s_{j-1,l} - 1/4\,(d_{j-1,l-1} + d_{j-1,l}),$$

to recover the even and

$$s_{j,2l+1} = d_{j,l} + 1/2(s_{j,2l} + s_{j,2l+2}),$$

to recover the odd samples.

Remarks:

- In this section we have not mentioned anything about what should happen at the edges of the signal. For now one can assume that signals are either periodic or infinite. In a later section, we show how lifting can be used to correctly deal with edge effects.

- The wavelet transform presented above is the biorthogonal (2,2) of Cohen-Daubechies-Feauveau [5]. One might not immediately recognize this, but by substituting the predict in the update, one can check that the coarser coefficients are given by

  $$s_{j-1,l} = -1/8\, s_{j,2l-2} + 1/4\, s_{j,2l-1} + 3/4\, s_{j,2l} + 1/4\, s_{j,2l+1} - 1/8\, s_{j,2l+2}.$$

  Note that, when written in this form (which is not lifting,) the transform cannot be computed in-place. Also to find the inverse transform one would have to rely on Fourier techniques. This is much less intuitive and does not generalize to irregular settings.

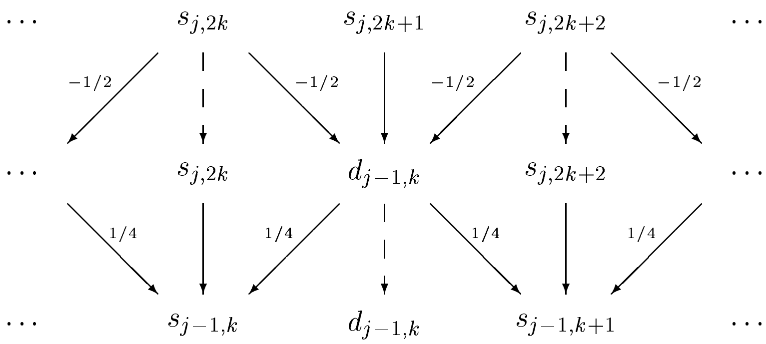

Figure 1.7: *At the top the initial vector of coefficients. As a first step all odd locations have* $1/2$ *of their neighboring even locations subtracted. In the second step all even locations get a contribution of* $1/4$ *of their neighboring odd locations leaving the smooth and detail coefficients. Now one can recurse by repeating the same set of operations with a memory stride of 2, 4, 8, and so forth. In the end the entire transform sits in the original memory locations.*

1.6 Subdivision Methods

In the previous section we used the ideas of *predict* and *update* to build wavelet transforms. As examples of prediction steps we saw constant prediction and linear prediction. In this section we will focus on *subdivision*, which is a powerful paradigm to build predictors. Subdivision is used extensively in *CAGD* to generate curves and surfaces. In that context it is used to refine a given mesh (1D or 2D) through a simple local procedure. Choosing this procedure carefully will result in an ever better approximation of some smooth limit curve or surface. Spline methods with the *de Casteljau* and *de Boor* algorithms, as well as certain interpolating subdivisions, such as the method of *Deslauriers-Dubuc*, fall into this category.

Considering subdivision methods as sources of predictors corresponds to focusing on the design of various forms of P function box(es) in wavelet transform wiring diagrams such as shown in Figures 1.4 and 1.5. Later on we will see how to construct suitable U function boxes, but for now we will work in the setting without a U box, see Figure 1.8. Equivalently, we may think of subdivision as an inverse wavelet transform with no detail coefficients. In this context, subdivision is often referred to as "the cascade algorithm." In later sections, we will see how for a given subdivision scheme one can define different ways to compute the detail coefficients.

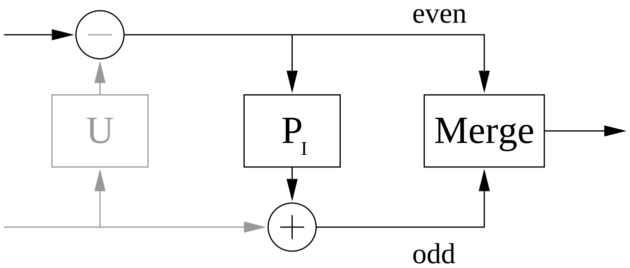

Figure 1.8*: The simplest subdivision and by implication inverse wavelet transform is interpolating subdivision. Values at odd locations are computed as a function of some set of neighboring even locations. The even locations do not change in the process.*

We begin by describing interpolating subdivision which is often useful if one is interested in constructions which interpolate a given data set. This corresponds to constructions with the simplest form of the P function box as shown in Figure 1.8. Here subdivision corresponds to predicting new values at the odd positions $s_{j+1,2k+1}$ at the next finer level, while all old values $s_{j+1,2k} = s_{j,k}$ remain the same.

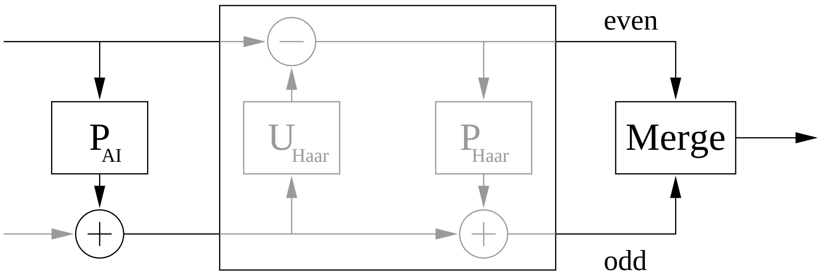

Figure 1.9*: Average-interpolating subdivision can be used to enhance an existing inverse Haar transform. Based on a number of even neighbors a Haar detail signal is computed and added to the odd locations. Applying the usual inverse Haar transform yields average-interpolating functions.*

Next we will consider a subdivision which is based on *average* interpolation and is related to the Haar transform and, through differentiation, to the interpolating transform. As the name suggests this method interpolates local averages, rather than samples of a function. This method will result in a P box which will be put in front of the inverse Haar transform as shown in Figure 1.9.

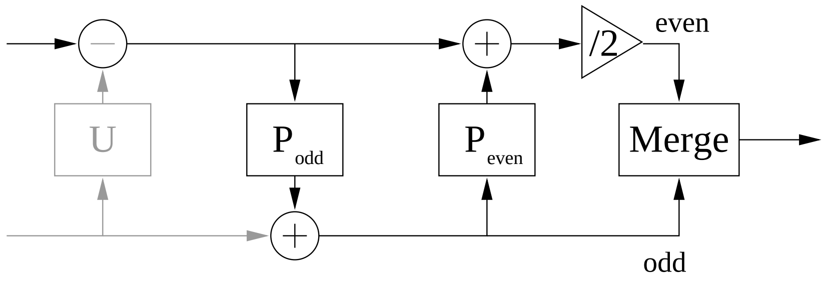

Figure 1.10: *B-splines can also be built with lifting. For cubic B-splines this diagram results. It is an instance of multiple P stages and also introduces a new element, scaling.*

Finally we discuss cubic B-splines as another example of building predictors. In this case we will see how powerful predictors can be built by allowing prediction to consist of several elementary stages including rescales (see the division by 2 on the top wire in Figure 1.10.)

1.6.1 Interpolating Subdivision

Interpolating subdivision starts by considering the problem of building an interpolant for a given data sequence. For example, we might be given a sequence of samples of some unknown function at regular intervals and the task to fill in intermediate values in a smooth fashion. Deslauriers and Dubuc attacked this problem by defining a recursive procedure for finding the value of an interpolating function at all dyadic points [10, 11]. This algorithm proceeds by inserting a new *predicted coefficient* inbetween each pair of existing coefficients. Since none of the already existing coefficients will get changed interpolation of the original data is assured.

Perhaps the simplest way to set up such an interpolating subdivision scheme is the following. Let $\{s_{0,k}\}$ with $k \in \mathbf{Z}$ be the original sample values. Now define a refined sequence of sample values recursively as

$$\begin{aligned} s_{j+1,2k} &= s_{j,k} \\ s_{j+1,2k+1} &= 1/2\,(s_{j,k} + s_{j,k+1}), \end{aligned}$$

and place the $s_{j,k}$ at locations $x_{j,k} = k\,2^{-j}$. Or in words, new values are inserted halfway between old values by linearly interpolating the two neighboring old values (see Figure 1.11.)

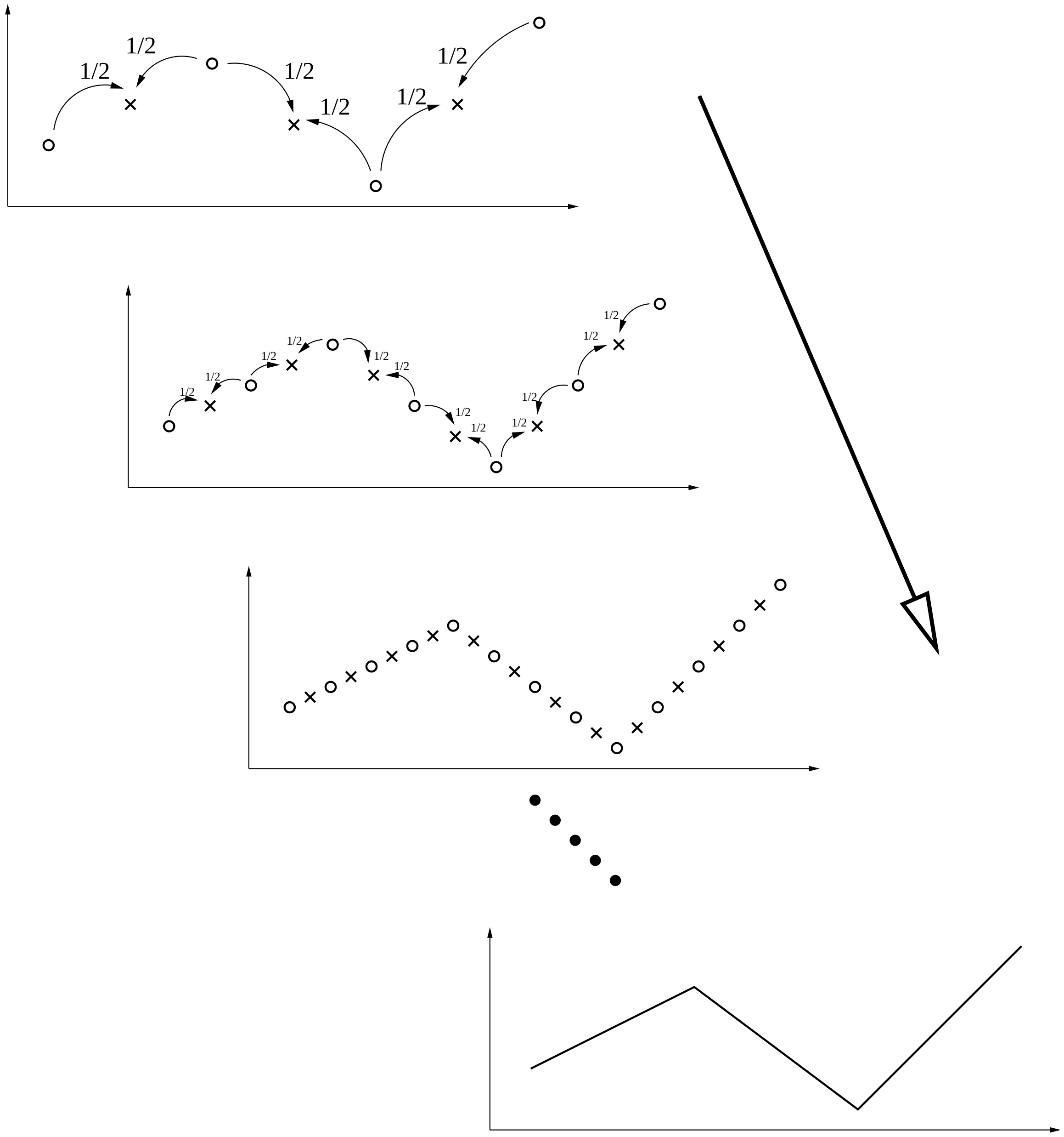

Figure 1.11: *The linear subdivision, or prediction, step inserts new values inbetween the old values by averaging the two old neighbors. Repeating this process leads in the limit to a piecewise linear interpolation of the original data.*

This subdivision rule was already used in the linear prediction transformation in Section 1.5. In the limit, values at all dyadic points will be defined and the function can be extended to a continuous function over all real numbers. The result is a piecewise linear interpolation of the original sample values. Suppose the initial sample values given to us were actually samples of a linear polynomial. In that case our subdivision scheme will exactly reproduce that linear polynomial. We then say that the *order* of the subdivision scheme is 2. The order of *polynomial reproduction* is important in quantifying the quality of a subdivision (or prediction) scheme.

To see how to build more powerful versions of such subdivisions we look at the procedure above in a slightly different light. Instead of thinking of it as averaging we can describe it via the construction of an interpolating polynomial. Given data $s_{j,k}$ and $s_{j,k+1}$ at $x_{j,k} = k\,2^j$ and $x_{j,k+1} = (k+1)\,2^j$ determine the unique linear polynomial which interpolates this data. Now define the new coefficient $s_{j+1,2k+1}$ as the value that this polynomial takes on at $x_{j+1,2k+1} = (2k+1)\,2^{j+1}$, which is halfway between $k\,2^j$ and $(k+1)\,2^j$. Given this way of looking at the problem an approach to build higher order predictors immediately suggests itself: instead of using only immediate neighbors to build a linear interpolating polynomial, use more neighbors on either side to build higher order interpolating polynomials and define the new value by evaluating the resulting polynomial at the new midpoint.

For example, we can use two neighboring values on either side and define the (unique) cubic polynomial $p(x)$ which interpolates those four values

$$\begin{aligned} s_{j,k-1} &= p(x_{j,k-1}) \\ s_{j,k} &= p(x_{j,k}) \\ s_{j,k+1} &= p(x_{j,k+1}) \\ s_{j,k+2} &= p(x_{j,k+2}). \end{aligned}$$

The new coefficient at $x_{j+1,2k+1}$ is defined to be the value that this cubic polynomial takes on at the midpoint. With the old samples untouched we get

$$\begin{aligned} s_{j+1,2k} &= s_{j,k} \\ s_{j+1,2k+1} &= p(x_{j+1,2k+1}). \end{aligned}$$

Note that the polynomial is in general a different one for each successive set of 4 old values. Figure 1.12 illustrates these ideas with the linear prediction on the left side and the cubic prediction on the right.

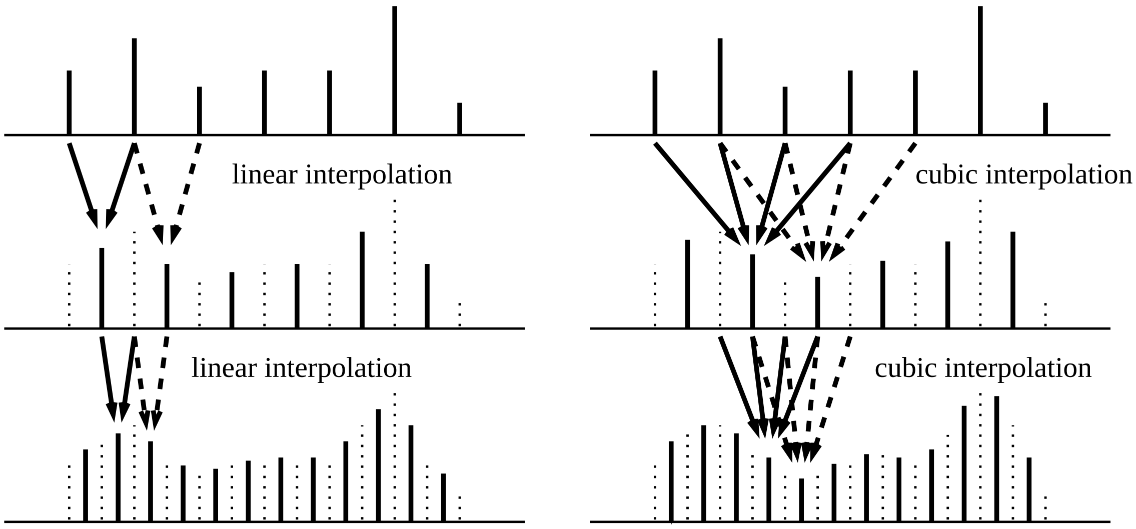

Figure 1.12: *On the left, the linear subdivision (or prediction) step inserts new values inbetween the old values by averaging the two old neighbors. On the right cubic polynomials are used for every quad of old values to determine a new inbetween value.*

Observing that this unique cubic interpolating polynomial is itself a linear function of the values $\{s_{j,k-1}, s_{j,k}, s_{j,k+1}, s_{j,k+2}\}$ we find, after some algebraic manipulation—and under the assumption that the associated $x_{j,k}$ are equally spaced—the new value $s_{j+1,2k+1}$ to be a simple weighting of $\{s_{j,k-1}, s_{j,k}, s_{j,k+1}, s_{j,k+2}\}$ with weights $\{-1/16, 9/16, 9/16, -1/16\}$. This stencil is well known in the CAGD literature as the 4-point scheme [16].

In general we use N ($N = 2D$ even) samples and build a polynomial of degree $N - 1$. For each group of $N = 2D$ coefficients $\{s_{j,k-D+1}, \ldots, s_{j,k}, \ldots, s_{j,k+D}\}$, it involves two steps:

1. Construct a polynomial p of degree $N - 1$ so that

$$s_{j,k+l} = p(x_{j,k+l}) \text{ for } -D+1 \leq l \leq D.$$

2. Calculate one coefficient on the next finer level as the value of this polynomial at $x_{j+1,2k+1}$

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

We say that the *order* of the subdivision scheme is N.

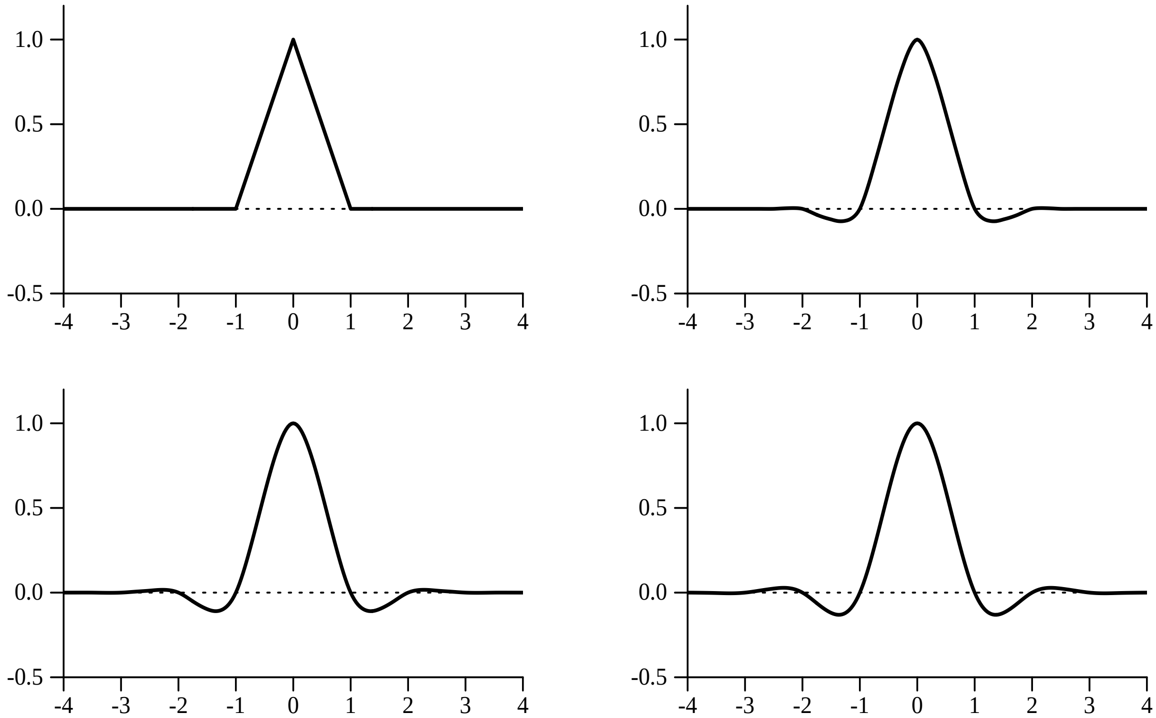

Figure 1.13: *Scaling functions resulting from interpolating subdivision. Going from left to right, top to bottom the order N of the subdivision is* 2, 4, 6, *and* 8. *Each function takes on the value one at the origin and zero at all other integers. Note that the functions have support, i.e., they are non-zero, over somewhat larger intervals than shown here.*

What makes interpolating subdivision so attractive from an implementation point of view is that we only need a routine which can evaluate an interpolating polynomial at a single location given some number of sample values and locations. The new sample value is defined through evaluation of this polynomial at the new, refined location. A particularly efficient (and stable) procedure for this is Neville's algorithm [24, 21]. If all samples are evenly spaced this polynomial and the associated weights need to be computed only once and can be used from then on. Notice also that nothing in the definition of this procedure requires the original samples to be located at integers. This feature can be used to define scaling functions over irregular subdivisions. Interval boundaries for finite sequences are also easily accommodated. We will come back to these observations in Chapter 2.

1.6.2 Interpolating Scaling Functions

Here we formally define the notion of *scaling functions*. Each coefficient $s_{j,k}$ has one scaling function denoted as $\varphi_{j,k}(x)$ associated with it. This scaling function is defined as follows: set all $s_{j,l}$ on level j equal to zero except for $s_{j,k}$ which is set to 1. Now run the interpolating subdivision scheme starting from level j ad infinitum. The resulting limit function is $\varphi_{j,k}(x)$. Consider some original sequence of sample values $s_{j,k}$ at level j. Simply using linear superposition and starting the subdivision scheme at level j, yields a limit function $f(x)$ of the form

$$f(x) = \sum_k s_{j,k}\, \varphi_{j,k}(x).$$

If the sample locations are regularly spaced ($x_{j,k} = k\, 2^{-j}$,) it is easy to see that all scaling functions are translates and dilates of one fixed function $\varphi(x) = \varphi_{0,0}(x)$:

$$\varphi_{j,k}(x) = \varphi(2^j x - k).$$

This function is also called the *fundamental* solution of the subdivision scheme. Figure 1.13 shows the scaling functions $\varphi_{0,0}$ which result from the interpolating subdivision of order 2, 4, 6, and 8 (left to right, top to bottom.) In the case of linear interpolation it is easy to see that the fundamental solution is the well known piecewise linear hat function. Perhaps surprisingly the fundamental solution of the cubic scheme described above is *not* a (piecewise) cubic polynomial. However, it is still true that the subdivision process will reproduce all cubic polynomials: if the initial sequence of samples came from a cubic polynomial $P(x)$ then all refinement steps will use exactly that polynomial $p(x) = P(x)$ (due to uniqueness) to define new intermediate values. As a consequence all new points will be samples of the original polynomial, in the limit reproducing the original cubic. Equivalently we say that any cubic polynomial can be written as a linear combination of scaling functions.

The properties of $\varphi(x)$ in the general case are:

1. **Compact support:** $\varphi(x)$ is exactly zero outside the interval $[-N+1, N-1]$. This easily follows from the locality of the subdivision scheme.

2. **Interpolation:** $\varphi(x)$ is interpolating in the sense that $\varphi(0) = 1$ and $\varphi(k) = 0$ for $k \neq 0$. This immediately follows from the construction.

3. **Polynomial reproduction:** Polynomials up to degree $N-1$ can be expressed as linear combinations of scaling functions. More precisely:

$$\sum_k (k2^{-j})^p \, \varphi_{j,k}(x) = x^p \text{ for } 0 \le p < N.$$

This can be seen by starting the scheme on level j with the sequence $x^p_{j,k}$ and using the fact that the subdivision definition insures the reproduction of polynomials up to degree $N-1$.

4. **Smoothness:** Typically $\varphi_{j,k} \in C^\alpha$ where $\alpha = \alpha(N)$. We know that $\alpha(4) < 2$ and $\alpha(6) < 2.830$ (strict bounds.) Also, for large N the smoothness increases linearly as $.2075N$. This fact is much less trivial than the previous ones. We refer to [10, 11] and [9, p. 226].

5. **Refinability:** This means the scaling function satisfies a *refinement relation* of the form

$$\varphi(x) = \sum_{l=-N}^{N} h_l \, \varphi(2x - l).$$

This can be seen as follows. Do one step in the subdivision starting from $s_{0,k} = \delta_{k,0}$. Call the result $h_l = s_{1,l}$. It is easy to see that only $2N+1$ coefficients h_l are non-zero. Now start the subdivision scheme from level 1 with these values $s_{1,l}$. The refinement relation follows from the fact that this should give the same result as starting from level 0 with the values $s_{0,k}$. Also because of interpolation, it follows that $h_{2l} = \delta_{0,l}$. We refer to the h_l as *filter coefficients*. With a change of variables in the refinement relation we get

$$\varphi_{j,k}(x) = \sum_l h_{l-2k} \, \varphi_{j+1,l}(x). \tag{1.2}$$

In the case of linear subdivision, the filter coefficients are $h_l = \{1/2, 1, 1/2\}$. The associated scaling function is the familiar linear B-spline "hat" function. The cubic case leads to the filter $h_l = \{-1/16, 0, 9/16, 1, 9/16, 0, -1/16\}$. For those familiar with the more traditional treatment of wavelets, the h_l coefficients describe the impulse response sequence of the low-pass filter used in the inverse wavelet transform. Once we have the filter coefficients, we can write the subdivision as

$$s_{j+1,l} = \sum_k h_{l-2k} \, s_{j,k}.$$

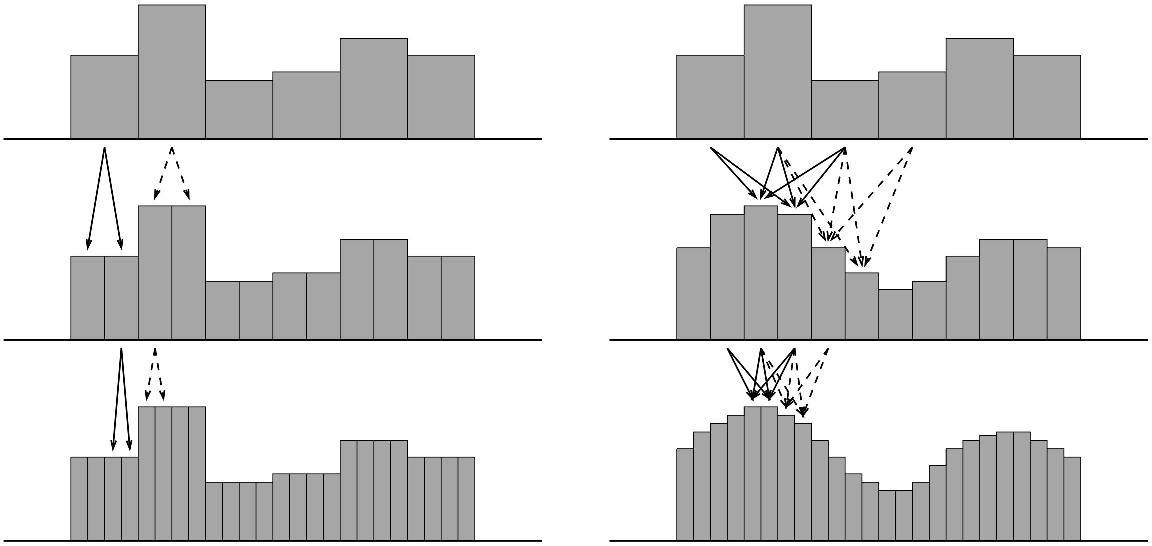

Figure 1.14: *Examples of average-interpolation. On the left a diagram showing the constant average-interpolation scheme. Each subinterval gets the average of a constant function defined by the parent interval. This is what happens in the inverse Haar transform if the detail coefficients are zero. On the right the same idea is applied to higher order average-interpolation using a neighboring interval on either side. The unique quadratic polynomial which has the correct averages over one such triple is used to compute the averages over the subintervals of the middle interval. This process is repeated an infinitum to define the limit function.*

We see that because $h_{2l} = \delta_{0,l}$, the subdivision scheme is interpolating, i.e., even indexed samples remain unchanged. Note that the h_l are used in a refinement relation to go from finer level *scaling functions* to coarser level *scaling functions*, while during subdivision the same h_k are used to go from coarser level *samples* to finer level *samples*.

1.6.3 Average-Interpolating Subdivision

In contrast to the interpolating subdivision scheme of Deslauriers-Dubuc we now consider another subdivision scheme: average-interpolation as introduced by Donoho [12]. To begin with we focus on the basic idea, i.e., how to produce the new values at the finer level directly from the old values at the coarser level. At the end of this section, we will fit average-interpolation into lifting as shown in the wiring diagram of Figure 1.9.

The starting point of interpolating subdivision was a set of samples of some function. Average-interpolation can be motivated similarly. Suppose that instead of samples we are given averages of some unknown function over intervals

$$s_{0,k} = \int_k^{k+1} f(x)\,dx.$$

For purposes of exposition we will assume that the intervals are all unit sized, a restriction which will be removed in the second generation setting (see Chapter 2.)

Such values might arise from a physical device which does not perform point sampling, but integration, as is done for example, by a CCD cell (to a first approximation.) How can we use such values to define a function whose averages are the measurement values given to us? One obvious answer is to use these values to define a piecewise constant function which takes on the value $s_{0,k}$ for $x \in [k, k+1]$. This corresponds to the following constant average-interpolation scheme

$$\begin{aligned} s_{j+1,2k} &= s_{j,k} \\ s_{j+1,2k+1} &= s_{j,k}. \end{aligned}$$

This is the inverse Haar transform with all detail coefficients equal to zero. Cascading this procedure ad infinitum we get a function which is defined everywhere and is piecewise constant. Furthermore its averages over intervals $[k, k+1]$ match the observed averages. The disadvantage of this simple scheme is that the limit function is not smooth. In order to understand how to increase the smoothness of such a reconstruction we define a general *average*-interpolating procedure.

One way to think about the previous scheme is to describe it as follows. We assume that the (unknown) function we are dealing with is a *constant polynomial* over the interval $[k, k+1]$. The values of $s_{j+1,2k}$ and $s_{j+1,2k+1}$ then follow as the averages of this polynomial over the respective subintervals. The diagram on the left side of Figure 1.14 illustrates this scheme.

Just as before we can extend this idea to higher order polynomials. The next natural choice is quadratic. For a given interval consider the intervals to its left and right. Define the (unique) quadratic polynomial $p(x)$ such that

$$\begin{aligned} s_{j,k-1} &= \int_{(k-1)2^{-j}}^{k\,2^{-j}} p(x)\,dx \\ s_{j,k} &= \int_{k\,2^{-j}}^{(k+1)2^{-j}} p(x)\,dx \end{aligned}$$

$$
s_{j,k+1} = \int_{(k+1)2^{-j}}^{(k+2)2^{-j}} p(x)\, dx.
$$

Now compute $s_{j+1,2k}$ and $s_{j+1,2k+1}$ as the average of this polynomial over the left and right subintervals of $[k\,2^{-j}, (k+1)2^{-j}]$

$$
\begin{aligned}
s_{j+1,2k} &= 2\int_{k\,2^{-j}}^{(k+1/2)2^{-j}} p(x)\, dx \\
s_{j+1,2k+1} &= 2\int_{(k+1/2)2^{-j}}^{(k+1)2^{-j}} p(x)\, dx.
\end{aligned}
$$

Figure 1.14 (right side) shows this procedure.

It is not immediately clear what the limit function of this process will look like, but it easy to see that the procedure will reproduce quadratic polynomials. The argument is the same as in the interpolating case: assume that the initial averages $\{s_{0,k}\}$ were averages of a given quadratic polynomial $q(x)$. In that case the unique polynomial $p(x)$ which has the prescribed averages over each triple of intervals will always be that same polynomial $q(x)$ which gave rise to the initial set of averages. Since the interval sizes go to zero and the averages over the intervals approach the value of the underlying function in the limit the original quadratic polynomial $q(x)$ will be reproduced.

We can define the scaling function exactly the same way as in the interpolating subdivision case. In general we use N intervals (N odd) to construct a polynomial of degree $N-1$. The order of the subdivision scheme is given by N. Figure 1.15 shows the scaling functions of order 1, 3, 5, and 7 (left to right, top to bottom.)

This scheme also has the virtue that it is very easy to implement. The conditions on the integrals of the polynomial result in an easily solvable linear system relating the coefficients of $p(x)$ to the $s_{j,k}$. In its simplest form (we will see more general versions later on) we can streamline this computation even further by taking advantage of the fact that the integral of a polynomial $p(x)$ is itself another polynomial $P(x) = \int_0^x p(y)\, dy$. This leads to another interpolation problem

$$
\begin{aligned}
0 &= P((k-1)2^{-j}) \\
s_{j,k} &= P(k\,2^{-j}) \\
s_{j,k} + s_{j,k+1} &= P((k+1)2^{-j}) \\
s_{j,k} + s_{j,k+1} + s_{j,k+2} &= P((k+2)2^{-j}).
\end{aligned}
$$

Given such a polynomial $P(x)$ the finer averages become

$$
s_{j+1,2k} = 2(P((k+1/2)2^{-j}) - P(k\,2^{-j}))
$$

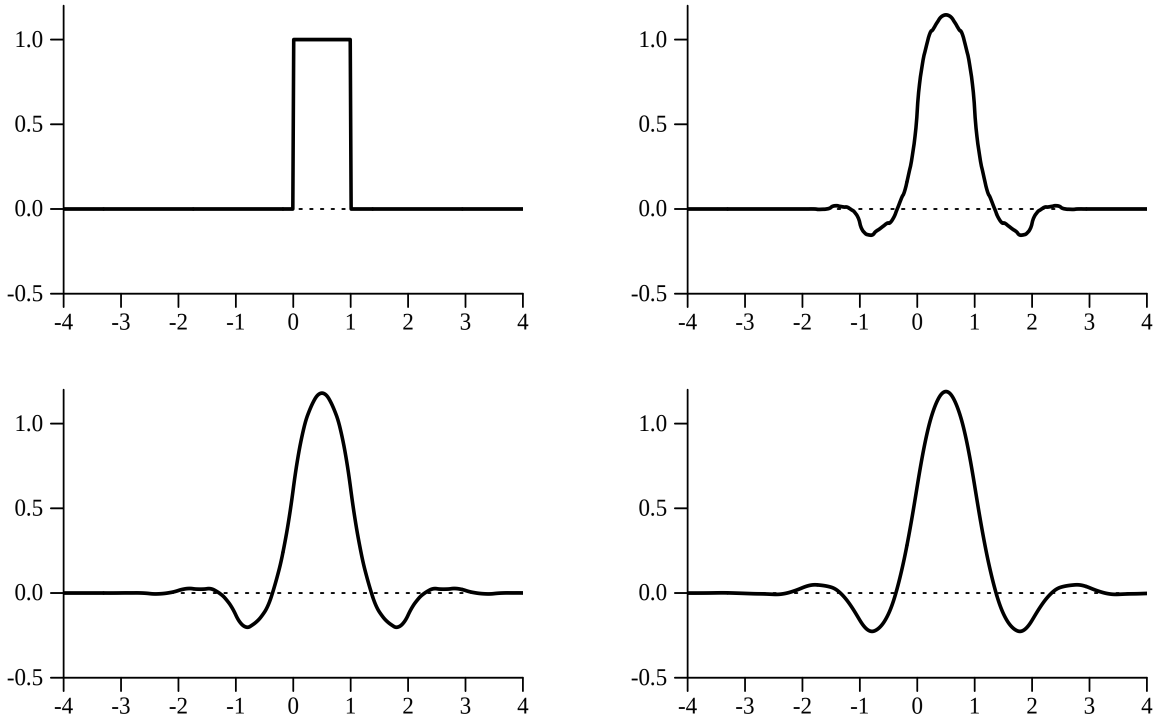

Figure 1.15: *Scaling functions which result from average-interpolation. Going from left to right, top to bottom orders of the respective subdivision schemes were* 1, 3, 5, *and* 7.

$$s_{j+1,2k+1} \;=\; 2(P((k+1)2^{-j}) - P((k+1/2)2^{-j})).$$

This computation, just like the earlier interpolating subdivision, can be implemented in a stable and efficient way with Neville's algorithm.

More generally we define the average-interpolating subdivision scheme of order N as follows. For each group of $N = 2D + 1$ coefficients $\{s_{j,k-D}, \ldots, s_{j,k}, \ldots, s_{j,k+D}\}$, it involves two steps:

1. Construct a polynomial $p(x)$ of degree $N - 1$ so that
$$s_{j,k+l} = \int_{(k+l)2^{-j}}^{(k+l+1)2^{-j}} p(x)\,dx \;\text{ for }\; -D \le l \le D.$$

2. Calculate two coefficients on the next finer level as
$$s_{j+1,2k} \;=\; 2\int_{k\,2^j}^{(k+1/2)2^j} p(x)\,dx$$

$$s_{j+1,2k+1} \;=\; 2\int_{(k+1/2)2^{-j}}^{(k+1)2^{-j}} p(x)\,dx.$$

These definitions will also work if the intervals are not on a dyadic grid but have unequal size. In that case one has to carefully account for the given interval width when computing the averages. However, nothing fundamental changes, even in the presence of boundaries or weighted measures. We will turn to these generalizations in Chapter 2.

There is one issue we have not yet addressed (see the remark in the first paragraph of this section.) The equations we have given so far do not fit into the lifting scheme framework, since one cannot simply overwrite $s_{j,k}$ with $s_{j+1,2k}$ as is desirable. Instead we use an already existing inverse Haar transform and use the average-interpolating subdivision as a P box *before* entering the inverse Haar transform. The diagram in Figure 1.9 illustrates this setup. Instead of computing $s_{j+1,2k}$ and $s_{j+1,2k+1}$ directly we will compute their difference $d_{j,k} = s_{j+1,2k+1} - s_{j+1,2k}$ and feed this as a difference signal into the inverse Haar transform. Given that the average of $s_{j+1,2k}$ and $s_{j+1,2k+1}$ is $s_{j,k}$, it follows that the inverse Haar transform when given $s_{j,k}$ and $d_{j,k}$ will compute $s_{j+1,2k}$ and $s_{j+1,2k+1}$ as desired. This leads to a transform with three lifting steps—the average-interpolating prediction, the Haar update, the Haar prediction—and all the advantages of lifting, such as in-place and easy invertibility remain.

1.6.4 Average-Interpolating Scaling Functions

Scaling functions are defined exactly the same way as in the interpolating case. If the samples are regularly spaced, they are translates and dilates of one fixed function $\varphi(x)$. The limit function $f(x)$ of the subdivision scheme is given by

$$f(x) = \sum_k s_{j,k}\,\varphi_{j,k}(x).$$

The properties of the associated scaling function are as follows:

1. **Compact support:** $\varphi(x)$ is exactly zero outside the interval $[-N+1, N]$. This easily follows from the locality of the subdivision scheme.

2. **Average-interpolation:** $\varphi(x)$ is average-interpolating in the sense that

   $$\int_k^{k+1} \varphi(x)\,dx = \delta_{k,0}.$$

   This immediately follows from the definition.

3. **Polynomial reproduction:** $\varphi(x)$ reproduces polynomials up to degree $N-1$. In other words
$$\sum_k 1/(p+1)\left((k+1)^{p+1}-k^{p+1}\right)\varphi(x-k) = x^p \text{ for } 0 \leq p < N.$$
This can be seen by starting the scheme with this particular coefficient sequence and using the fact that the subdivision reproduces polynomials up to degree $N-1$.

4. **Smoothness:** $\varphi(x)$ is continuous of order R, with $R = R(N) > 0$. One can show that $R(3) \geq .678$, $R(5) \geq 1.272$, $R(7) \geq 1.826$, $R(9) \geq 2.354$, and asymptotically $R(N) \approx .2075N$ [12].

5. **Refinability:** $\varphi(x)$ satisfies a refinement relation of the form
$$\varphi(x) = \sum_{l=-N+1}^{N} h_l\,\varphi(2x-l).$$
This follows from similar reasoning as in the interpolating case starting from $s_{0,k} = \delta_{0,k}$.

As before subdivision can be written as:
$$s_{j+1,l} = \sum_k h_{l-2k}\,s_{j,k}.$$
In the case of quadratic subdivision, the filter coefficients are $h_l = \{-1/8, 1/8, 1, 1, 1/8, -1/8\}$. For quartic subdivision $h_l = \{3/128, -3/128, -11/64, 11/64, 1, 1, 11/64, -11/64, -3/128, 3/128\}$ results. These constructions are part of the biorthogonal family of scaling functions described by Cohen-Daubechies-Feauveau [5], where they are named (1,3) and (1,5) respectively.

An interesting connection between Deslauriers-Dubuc interpolation and average-interpolation was pointed out by Donoho [12, Lemma 2.2]: Given some sequence $\{s_{0,k}\}$ apply interpolating subdivision of order $N = 2D$ to it. Comparing this to the sequence that results from applying average-interpolation of order $N' = 2D-1$ to the sequence $\{s'_{0,k} = s_{0,k+1} - s_{0,k}\}$ we find $s_{j,k} = 2^j s'_{j,k}$ (once again assuming equal sized intervals.) This observation follows directly from the fact that the average of the derivative of an interpolating polynomial p is simply the difference of the values that the polynomial takes on at the end and start of an interval, divided by the size of the interval. As a consequence we have
$$\frac{d}{dx}\varphi^I(x) = \varphi^{AI}(x+1) - \varphi^{AI}(x), \tag{1.3}$$

where the superscripts stand for I-nterpolating scaling function and A-verage I-nterpolating scaling function.

This provides a simple recipe for computing the derivative of a function $f(x)$ which results from interpolation of some sequence $\{s_{0,k}\}$: simply take successive differences first and then apply the average-interpolating subdivision of one order less. Obviously, one could also take the differences and then apply an interpolating subdivision method, not necessarily average-interpolation. In that case the limit function would be an approximation of the derivative, while Equation 1.3 holds exactly.

1.6.5 B-Spline Subdivision

B-splines and in particular cubic B-splines are a very common primitive in computer graphics. Their popularity stems from the fact that they are C^2, which is important in many rendering applications to ensure smooth shading. They also possess the convex hull property and are variation diminishing which makes them a favorite for curve and surface modeling tasks.

In this section we will show how cubic B-splines can be used as a predictor in wavelet constructions. In order to keep the exposition simple we will only consider cardinal cubic B-splines here, i.e., all control points are associated with integer parameter positions. We will also assume a bi-infinite sequence for now and only discuss the boundary case later.

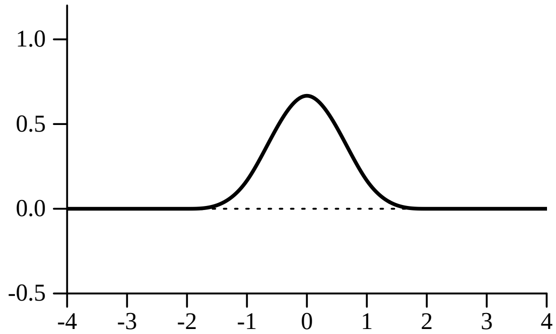

Figure 1.16: *Cubic B-spline scaling function.*

Given a set of cubic B-spline control points at the integers $\{s_{0,k}\}$ subdivision tells us how to find a set of control points at the half integers which describe the same underlying B-spline curve. Repeating this process as before the control points become dense and converge to the actual curve (see Figure 1.16.) Typically this subdivision rule is described through a convolution with the sequence $\{h_k\} = 1/8\,\{1, 4, 6, 4, 1\}$, i.e.,

$$\begin{aligned} s_{j+1,2k} &= 1/8\,(s_{j,k-1} + 6s_{j,k} + s_{j,k+1}) \\ s_{j+1,2k+1} &= 1/8\,(4s_{j,k} + 4s_{j,k+1}). \end{aligned}$$

As stated this implementation of cubic B-spline subdivision does not fit into our wiring diagram framework, since it cannot be executed in place: the memory location for $s_{j,k}$ and $s_{j+1,2k}$ are identical. Computing $s_{j+1,2k}$ will leave us with the wrong values needed for the computation of $s_{j+1,2k+2}$, for example!

This is the first instance of a subdivision method which requires a *sequence* of elementary P function boxes, as well as a new primitive: scaling. Scaling fits into the lifting philosophy as it can be done in-place and is trivially inverted. There are a number of advantages to implementing the cubic B-spline subdivision in that way. Aside from allowing us to do the computation in place, it will be maximally parallel and, as we will see later, make it trivial to derive elementary wavelets (and through further lifting an entire class of wavelets.)

We begin by averaging even coefficients into odd locations

$$s_{j+1,2k+1} = 1/2\,(s_{j,k} + s_{j,k+1}).$$

This is the P_{odd} box in the diagram of Figure 1.10. Next apply a P_{even} function box to get

$$s_{j+1,2k} = s_{j,k} + 1/2\,(s_{j+1,2k-1} + s_{j+1,2k+1}).$$

Finally dividing all even locations be 2

$$s_{j+1,2k} \mathrel{/}= 2$$

we get our desired sequence. Substitution of the definition of the odd locations immediately verifies that the result is as intended

$$s_{j+1,2k} = 1/2\,(1/4 s_{j,k-1} + 6/4 s_{j,k} + 1/4 s_{j,k+1}).$$

We have succeeded in building cubic B-spline scaling functions with a simple inplace computation which only involves immediate neighbors. As we will see later, choosing an appropriate U function box to put before this inverse transform (see Figure 1.10) can give us associated spline wavelets.

1.6.6 The Next Step

We have just seen a number of ways to generate scaling functions which fit well with the overall philosophy of lifting and give the practitioner a rich set of constructions from which to choose. Before we discuss the construction of the associated wavelets, and how they "drop out of" the wiring diagrams we first consider the notion of a Multiresolution Analysis more formally in the following section.

1.7 Multiresolution Analysis*

In this section, we will go into some more mathematical detail about multiresolution analysis as originally conceived by Mallat and Meyer [20, 19]. In the previous section we defined the notion of a scaling function $\varphi(x)$ and saw how all scaling functions are simply translates and dilates of one fixed function:

$$\varphi_{j,l}(x) = \varphi(2^j x - l).$$

In this section we will use these functions to build a multiresolution analysis.

Assume we start a subdivision scheme of the previous section on level j from a sequence $\{s_{j,l}\}$. Because of linearity, we can write the resulting limit function $s_j(x)$ as

$$s_j(x) = \sum_l s_{j,l}\, \varphi_{j,l}(x).$$

The scaling functions have compact support, so that for a fixed x the summation only involves a finite number of terms. We next define the space V_j of all limit functions obtained from starting the subdivision on level j. This is the linear space spanned by the scaling functions $\varphi_{j,l}$ with $l \in \mathbf{Z}$:

$$V_j = \operatorname{span}\{\varphi_{j,l}(x) \mid l \in \mathbf{Z}\}.$$

For example, if $\varphi(x)$ is the Haar scaling function, the V_j is the space of functions which are piecewise constant on the intervals $[k2^{-j}, (k+1)2^{-j})$ with $k \in \mathbf{Z}$. If $\varphi(x)$ is the piecewise linear hat (or tent) function, V_j is the space of continuous functions which are piecewise linear on the intervals $[k2^{-j}, (k+1)2^{-j})$ with $k \in \mathbf{Z}$.

The different V_j spaces satisfy the following properties which make them a multiresolution analysis:

1. **Nestedness:** $V_j \subset V_{j+1}$.

2. **Translation:** if $f(x) \in V_j$ then $f(x + k2^{-j}) \in V_j$.

3. **Dilation:** if $f(x) \in V_j$ then $f(2x) \in V_{j+1}$.

4. **Completeness:** every function $f(x)$ of finite energy ($\in \mathrm{L}_2$) can be approximated with arbitrary precision with a function from V_j for suitably high j.

The nestedness property follows immediately from the refinement relation. If we can write $\varphi(x)$ as a linear combination of the $\varphi(2x - k)$, then we can also write $\varphi_{j,l}$ as a linear combination of the $\varphi_{j+1,k}$ and thus $V_j \subset V_{j+1}$. The translation and dilation properties follow from the definition of V_j. The proof of the completeness property is beyond the scope of this tutorial. We refer to [9] for more details.

A very important property of a multiresolution analysis is the *order*. We say that the order of a multiresolution analysis is N in case every polynomial $p(x)$ of degree strictly less than N can be written as a linear combination of scaling functions of a given level. In other words polynomials of degree less than N belong to all spaces V_j. For example, in case of the Haar multiresolution analysis, $N = 1$. Constant functions belong to all V_j spaces. In case of the hat function, $N = 2$ because all linear functions belong to all V_j spaces. Note that the order of a multiresolution analysis is the same as the order of the predictor used to build the scaling functions.

This concludes the discussion of subdivision, scaling functions, and multiresolution. Remember that the original motivation for treating subdivision was that it provides predictors which readily fit into the lifting framework. In the next section we turn to the other main component of lifting: update.

1.8 Update Methods

We have seen several examples of wiring diagram constructions. In the very beginning of this chapter we started with the Haar forward and inverse transform. In that case it was clear from inspection what the update box had to do, given that we wanted the coarser signal to be an average of the finer signal. For the linear transform, finding the update box required some algebraic manipulations. Generally update boxes are designed to ensure that the coarser signal has the same average as the higher resolution signal.

In the section on subdivision methods we discussed various ways of realizing predictors in the context of an inverse transform with, effectively, zero wavelet coefficients. The diagrams in

Figures 1.8, 1.9, and 1.10 show where the update boxes go, but we have not yet explained how they should be designed.

That is the subject matter of this section. We begin by making some simple observations about forward and inverse transforms and give a number of design criteria for update boxes. As in the section on subdivision we will only consider the regular setting and postpone generalizations to Chapter 2.

1.8.1 Forward and Inverse Transforms

One of the nice features of the lifting formalism is the ease with which one finds the inverse transform: simply flip the diagram left to right, switch additions and subtractions, and switch multiplications and divisions. In this way forward and inverse transform are equivalent from a design point of view. For some questions, it is easier to consider one rather than the other. For example, in the case of borrowing predictors from subdivision methods it is most natural to first consider the inverse transform, and later derive the forward transform. These relationships between forward and inverse "wiring diagrams" lead directly to important consequences.

Let us consider the case of subdivision more closely. It is a linear transformation which takes in m degrees of freedom on the top wire, say coefficients associated with the integers, and outputs $(2m - 1)$ degrees of freedom, i.e., coefficients associated with the half integers due to interpolating subdivision. We are a bit sloppy here when ignoring issues such as the boundary, but for purposes of our argument we may do so for now (later on when we discuss the generalization to boundaries we will see that the present argument is solid.) Figure 1.17 indicates this idea on the left.

A question that immediately arises is whether we can characterize the remaining $m-1$ degrees of freedom in a simple and useful form. This is an important aspect of multiresolution analysis where we have the spaces $V_j \subset V_{j+1}$ and we may ask how to characterize the difference between two such consecutive spaces: what is the difference in "resolution" between the two spaces? What are we loosing when we go from a finer resolution to a coarser resolution?

Since subdivision is a linear transformation we may think of it as a $((2m-1) \times m)$ matrix. In that language the question of characterizing the extra degrees of freedom, is to ask for a set of columns to add to this matrix so that it becomes square, is banded, and its inverse is banded as well. At first sight these requirements appear to be rather hard to satisfy. However, we already know the answer to this question!

Suppose we have the inverse transform diagram with the detail wire added (on the right

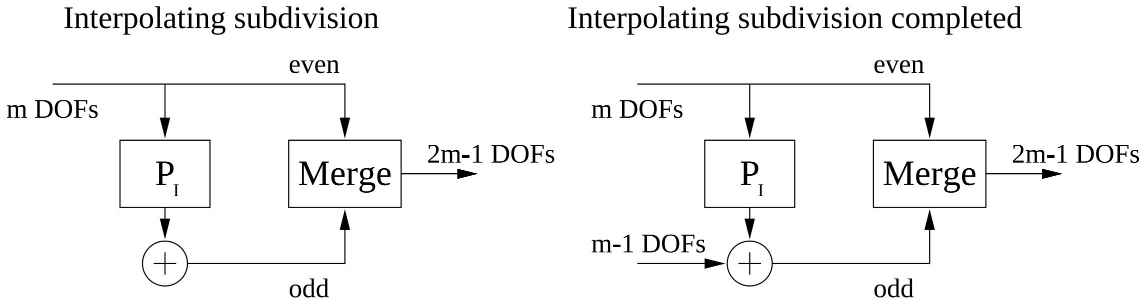

Figure 1.17: *On the left pure interpolating subdivision is indicated: odd samples are computed using a predictor based on even samples. Beginning with m degrees of freedom, $(2m-1)$ result after one step (in the bounded interval case.) Simply extending the lower wire to the left is a proper completion, i.e., it describes the extra degrees of freedom added by one subdivision step. The proof consists of observing that the completed diagram is invertible (as opposed to the pure subdivision diagram which is not.) Given that it is invertible the bottom wire must represent the remaining DOFs. In the language of matrices—all operations are linear—this simply states that subdivision can be completed in such a way that the resulting matrix is banded and its inverse is banded as well.*

side of Figure 1.17,) then we can immediately build a diagram with the forward transform ("running everything backwards.") Concatenating the two we get the identity by construction. Since the whole operation is linear we have just convinced ourselves that the linear operator represented by the inverse transform is invertible, *if we also consider the lower detail wire.* In other words, writing subdivision in our somewhat nonstandard way, we can immediately read off a representation of the additional degrees of freedom introduced as one goes from V_j to V_{j+1}: simply put the delta sequence $\delta_{0,l}$ on the detail wire. In the case of interpolating subdivision (Figure 1.18 shows the example of linear interpolating subdivision) the result is a delta sequence $\delta_{1,2l+1}$. If we continue the subdivision process ad infinitum the functions shown result. Note how the functions due to (successive) smooth coefficients overlap. The functions due to detail coefficients, the actual wavelets, are the same shape in this case, but smaller and sit inbetween.

A more complicated (and interesting) example is provided by the subdivision diagram for the cubic B-splines. Figure 1.19 illustrates the behavior of this subdivision diagram as a function of putting successive delta sequences on the smooth and detail wire respectively. For the smooth wire $\delta_{0,k}$ results in the well known sequence $1/8\,\{1,4,6,4,1\}$ of refinement coefficients. Putting

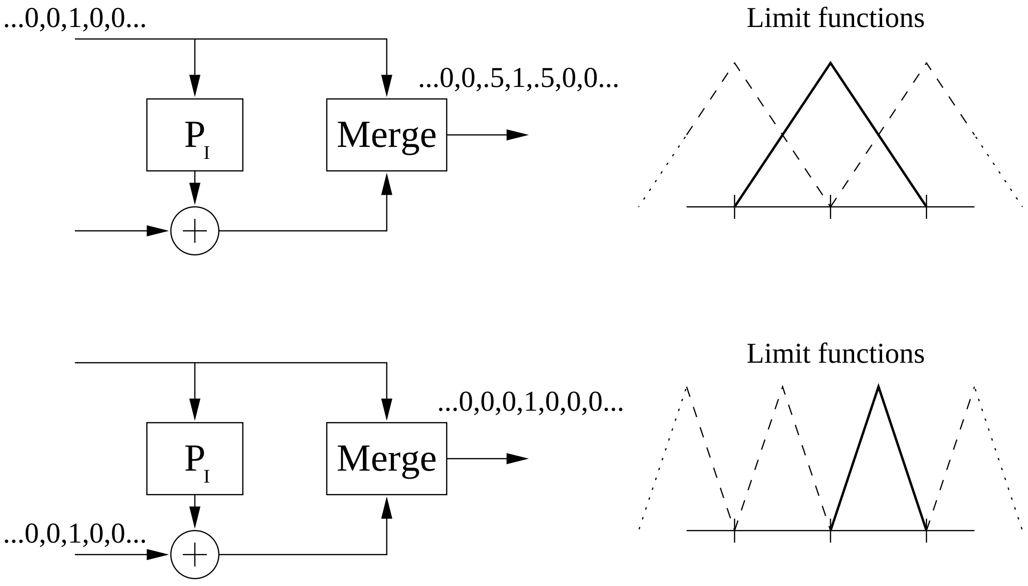

Figure 1.18: *The result of putting a delta sequence on the top (smooth) or bottom (detail) wire of an interpolating subdivision (inverse transform) in the case of the linear predictor. On the right the resulting functions, after subdivision ad infinitum, for a sequence of consecutive delta inputs. Note how the detail functions sit "inbetween" the scaling functions. The picture is essentially the same for higher order interpolating predictors: the detail functions are inbetween the scaling functions and dilated (by a factor of 2) versions of the scaling functions.*

$\delta_{0,l}$, on the detail wire results in the sequence $1/4\,\{1,4,1\}$. From our considerations above we know that this sequence must be a *completion* of the space spanned by the $1/8\,\{1,4,6,4,1\}$ and in this way captures the degrees of freedom inbetween V_j and V_{j+1}.

In principle we could stop here and would not have to worry about update boxes. The problem is that there are many possible completions and the ones we get "for free" from the subdivision diagram are not necessarily the only ones or the best ones. Note that the detail functions which we got in the linear (Figure 1.18) and cubic B-spline case (Figure 1.19) do not have a zero integral, a condition we need to ensure that the coarser version of the signal has the same integral as the finer version (see the next section.)

Designing update boxes is all about manipulating the representation of these differences, of "inbetween" degrees of freedom. That this should be so is easy to see. The update box on the

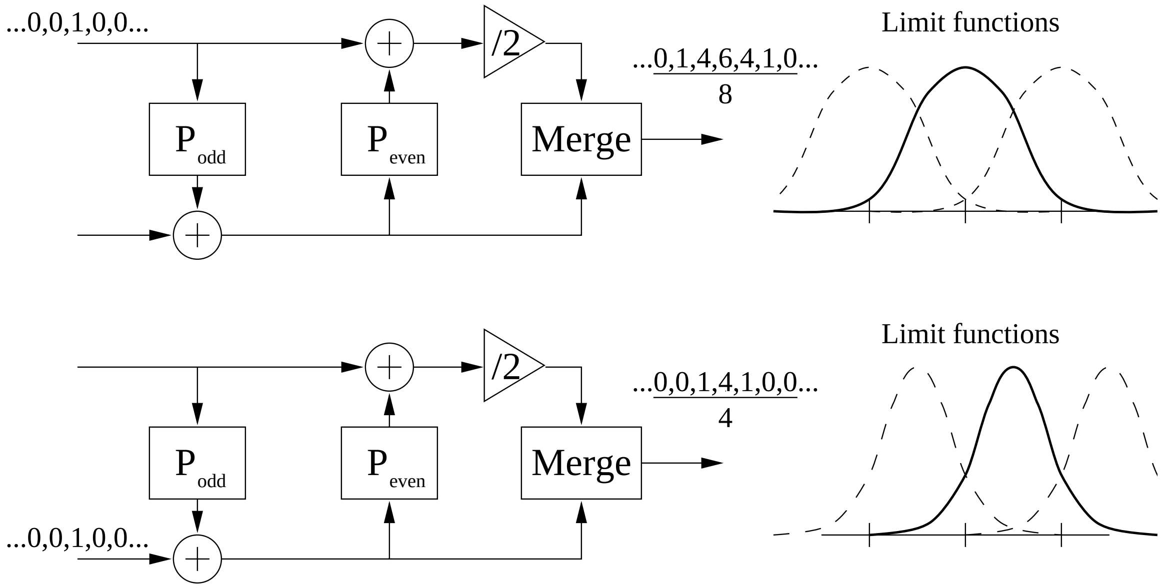

Figure 1.19: *The result of putting a delta sequence on the top (smooth) or bottom (detail) wire of a cubic B-spline subdivision (inverse transform.) On the right the resulting functions, after subdivision ad infinitum, for a sequence of consecutive delta inputs.*

inverse transform side makes a contribution to the smooth wire based on a signal on the odd wire (see Figures 1.8, 1.9, and 1.10.) Considering the sequence $\delta_{0,l}$ on the detail wire we can see how having a U box will alter the sequence generated at the end (see Figure 1.20.)

From the examples of the Haar and Linear transform we remember that the main purpose of the update box is to ensure that the average of the coarser level signals is maintained, i.e,

$$2^{-j} \sum_{l=0}^{2^j-1} s_{j,l} \tag{1.4}$$

is independent of j. From this we see that there is no need for an extra update box in the case of wavelet transforms built from average-interpolation. Given that we can write the forward transform as a Haar transform followed by one average-interpolation prediction operation, the update box which is part of the Haar transform already assures (1.4). We next discuss how to design update boxes for interpolating subdivision and cubic B-spline subdivision.

1.8.2 Update for Interpolation

We already encountered one update step for the linear example in the very beginning. It consisted of computing the coarser level coefficients as

$$s_{j-1,l} = s_{j,2l} + 1/4\,(d_{j-1,l-1} + d_{j-1,l}).$$

We show here how such an update can be derived in general. The main purpose of the update step is to ensure that the average is maintained. Saying that the average of the signal s_j is equal to the average of the signal s_{j-1} is equivelent to saying that the average of the detail or difference signal d_{j-1} is zero. Given that the detail signal is nothing else but a linear combination of complementary sequences as derived in the previous section, it is sufficient to construct complementary sequences with zero average. We consider Figure 1.18 again. The complementary sequence, i.e., the result from putting a 1 on the odd wire is $\{0, 0, 1, 0, 0\}$. Obviously this does not have a zero average. Therefore we use the update box from the odd wire to the even wire (see Figure 1.20.) The result of putting a 1 on the even wire is $\{1/2, 1, 1/2, 0, 0\}$ or $\{0, 0, 1/2, 1, 1/2\}$

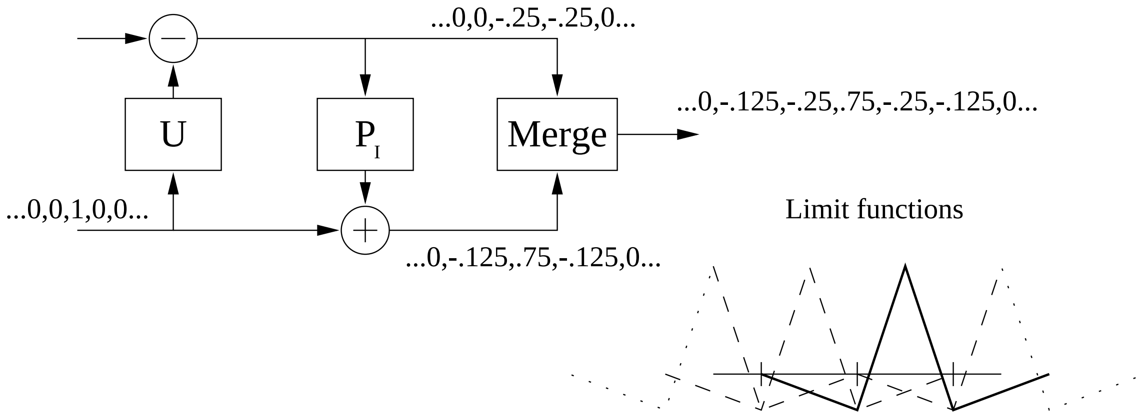

Figure 1.20: *Starting with the delta sequence on the bottom wire, the update box make a contribution to the even wire, which in turn causes further changes in the odd wire after prediction. The result is the sequence at the end which corresponds to the indicated limit functions. The average of the coefficients is zero, as expected.*

depending on the position of the original 1. The update box now combines these sequences to build a complementary sequence with average zero. We propose an update which adds a fraction

A of the odd element into the two neighboring evens. This leads to a complementary sequence of the form

$$\{0, 0, 1, 0, 0\} - A\,\{1/2, 1, 1/2, 0, 0\} - A\,\{0, 0, 1/2, 1, 1/2\}.$$

Choosing $A = 1/4$ leads to the complementary sequence

$$\{-1/8, -1/4, 3/4, -1/4, -1/8\},$$

which has average zero as desired.

Other types of update can be used as well. They ensure that not only the average but also the first $\tilde{N}$ moments of the sequences are preserved, i.e.,

$$m_p = 2^{-j} \sum_{l=0}^{2^j-1} l^p\, s_{j,l} \tag{1.5}$$

is independent of j for $0 \leq p < \tilde{N}$. The update weights can be found easily as well. In [27] it is shown that as long as $\tilde{N} \leq N$ one can simply take the *same* weights for update as one used for prediction, divided by two. For example, in case the predictor is of order $N \geq 4$, one can get an update with $\tilde{N} = 4$ by letting

$$s_{j-1,l} = s_{j,2l} - 1/32\, d_{j-1,l-2} + 9/32\, d_{j-1,l-1} + 9/32\, d_{j-1,l} - 1/32\, d_{j-1,l+1}.$$

1.8.3 Update for cubic B-splines

The update box for the B-splines can be found using the same reasoning as in the previous section. The complementary sequence now is

$$\{0, 0, 1/4, 1, 1/4, 0, 0\} - A/8\,\{1, 4, 6, 4, 1, 0, 0\} - A/8\,\{0, 0, 1, 4, 6, 4, 1\},$$

which leads to $A = 3/8$.

1.9 Wavelet Basis Functions*

In this section we formally establish the relationship between detail coefficients and the wavelet functions. In the earlier section on multiresolution analysis (Section 1.7) we described spaces V_j which are spanned by the fundamental solutions of the subdivision process, the scaling functions $\varphi_{j,l}$. Here we will examine the differences $V_{j+1} \backslash V_j$ and the wavelets $\psi_{j,l}(x)$ which span these differences.

Consider an initial signal $s_n = \{s_{n,l} \mid l \in \mathbf{Z}\}$. We can associate a function $s_n(x)$ in V_n with this signal:

$$s_n(x) = \sum_l s_{n,l}\varphi_{n,l}(x).$$

Calculate one level of the wavelet transform as described in an earlier section. This yields coarser coefficients $s_{n-1,l}$ and coefficients $d_{n-1,l}$. The coarser coefficients $s_{n-1,l}$ corresponds to a new function in V_{n-1}:

$$s_{n-1}(x) = \sum_l s_{n-1,l}\varphi_{n-1,l}(x).$$

Recall that $\varphi_{n-l,l}(x)$ is the function that results if we run the inverse transform on the sequence $\delta_{n-1,l}$. With this the above equation expresses the fact that the function $s_{n-1}(x)$ is the result of "assembling" the functions associated with a 1 in each of the positions $(n-1, l)$, each weighted by $s_{n-1,l}$. At first sight it is unclear which function the signal d_{n-1} corresponds to. Somehow we feel that it corresponds to the "difference" of the signals $s_n(x)$ and $s_{n-1}(x)$. To find the solution we need to define the wavelet function.

Consider a detail coefficient $d_{0,0} = 1$ and set all other detail coefficients $d_{0,k}$ with $k \neq 0$ to zero. Now compute one level of an inverse wavelet transform. This corresponds to what we did in Figures 1.18 and 1.19 when putting a single 1 on the lower wire. After a single step of an inverse transform this yields coefficients g_l of a function in V_1, e.g., $g_l = \{0, 1, 0\}$ in case of linears without lifting in Figure 1.18 or $g_l = \{-1/8, -1/4, 3/4, -1/4, -1/8\}$ in the case of linears with lifting. We define this function to be the wavelet function $\psi(x) = \psi_{0,0}(x)$. Thus:

$$\psi_{0,0}(x) = \sum_l g_l\, \varphi_{1,l}(x) = \sum_l g_l\, \varphi(2x - l).$$

This wavelet function is often called the *mother wavelet*. All other wavelets $\psi_{j,k}$ are obtained by setting $d_{j,k} = 1$ and $d_{j,l}$ with $l \neq k$ to zero, calculating one level of the inverse wavelet transforms and then subdividing ad infinitum to get the corresponding function in V_{j+1}. Using an argument similar to the case of scaling functions, one can show that all wavelets are translates and dilates of the mother wavelet:

$$\psi_{j,k}(x) = \psi(2^j x - k).$$

Figure 1.21 shows several mother wavelets coming from interpolation, Figure 1.22 shows wavelets resulting from average-interpolation, and Figure 1.23 shows a wavelet associated with cubic B-spline scaling functions.

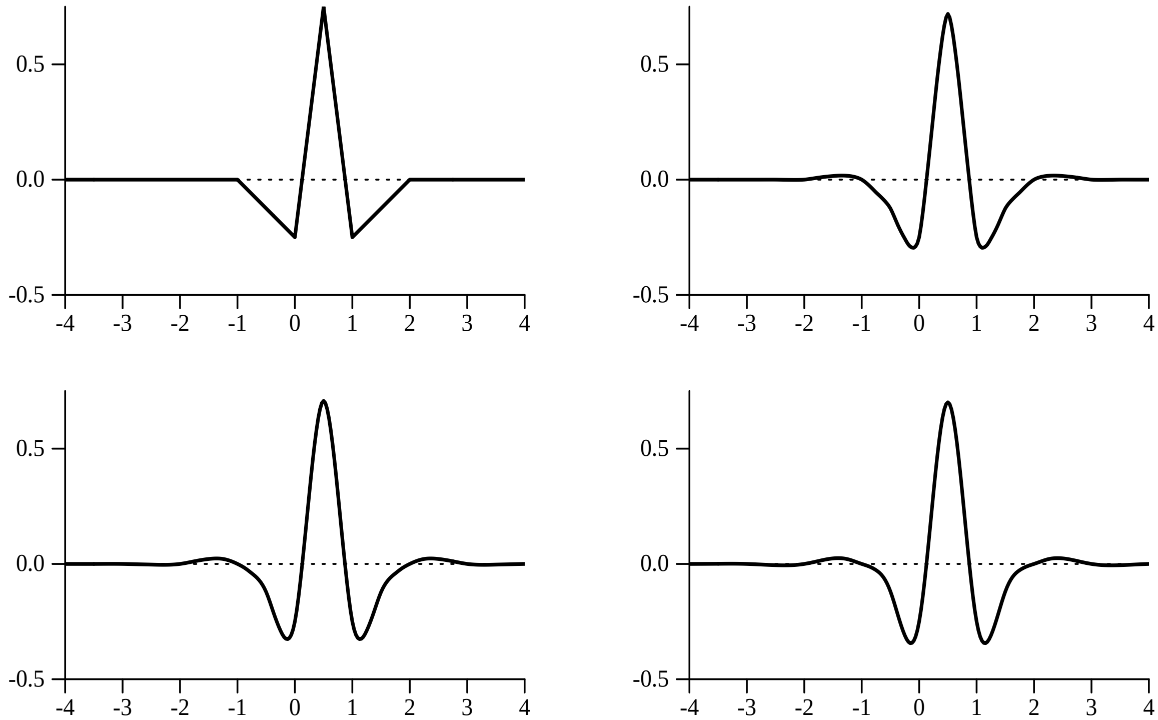

Figure 1.21: *Wavelets from interpolating subdivision. Going from left to right, top to bottom are the wavelets of order $N = 2$, 4, 6, and 8. Each time $\tilde{N} = 2$.*

Now we can answer the question from the start of this section. We first define the detail function $d_{n-1}(x)$ to be

$$d_{n-1}(x) = \sum_l d_{n-1,l}\, \psi_{n-1,l}.$$

Because of linear superposition of the wavelet transform, it follows that

$$s_n(x) = s_{n-1}(x) + d_{n-1}(x) = \sum_l s_{n-1,l}\, \varphi_{n-1,l}(x) + \sum_l d_{n-1,l}\, \psi_{n-1,l}.$$

In words, the function defined by the sequence $s_{n,l}$ is equal to the result of performing one forward transform step, computing $s_{n-1,l}$ (upper wire) and $d_{n-1,l}$ (lower wire,) and then using these coefficients in a linear superposition of the elementary functions which result when running an inverse transform on a single 1 on the top or bottom wire, in the respective position.

So what we expected is true: the detail function $d_{n-1}(x)$ is nothing but the difference between the original function and the coarser version. As we will see later, there are many different ways

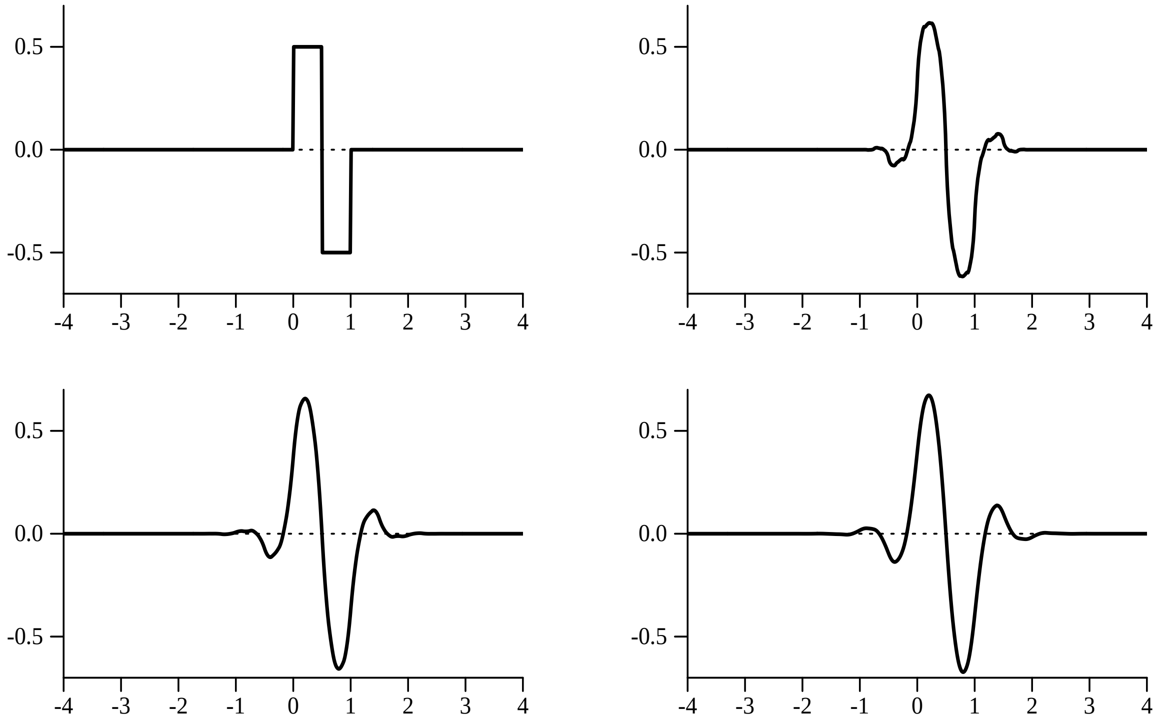

Figure 1.22: *Wavelets from average-interpolation. Going from left to right, top to bottom are the wavelets which correspond to the orders $N = 1$, 3, 5, and 7. Each time $\tilde{N} = 1$.*

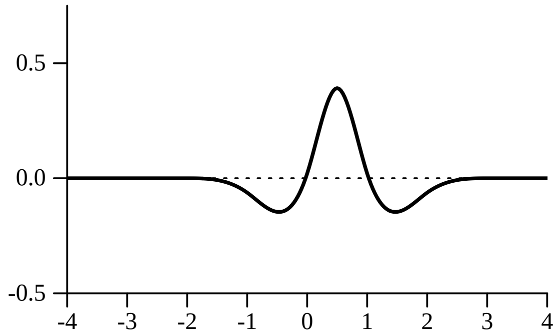

Figure 1.23: *Cubic B-spline wavelet*

to define a detail function d_{n-1}. They typically depend on two things: first, the detail coefficients $d_{n-1,l}$ (which are computed in the forward transform) and secondly the wavelet functions $\psi_{n-1,l}$

(which are built during the inverse transform.)

The n-level wavelet decomposition of a function $s_n(x)$ is defined as

$$s_n(x) = s_0(x) + \sum_{j=0}^{n-1} d_j(x).$$

The space W_j is defined to be the space that contains the difference functions:

$$W_j = \operatorname{span}\{\psi_{j,l}(x) \mid l \in \mathbf{Z}\}.$$

It then follows that W_j is a space complementing V_j in V_{j+1}

$$V_{j+1} = V_j \oplus W_j.$$

In an earlier section we defined the notion of order of a multiresolution analysis. If the order is N, then the wavelet transform started from any polynomial $p(x) = s_n(x)$ of degree less than N will only yield zero wavelet coefficients $d_{j,l}$. Consequently all detail functions $d_j(x)$ are zero and all $s_j(x)$ with $j < n$ are equal to $p(x)$.

In this section we introduce the *dual order* $\tilde{N}$ of a multiresolution analysis. We say that the dual order is $\tilde{N}$ in case the wavelets have $\tilde{N}$ vanishing moments:

$$\int_{-\infty}^{+\infty} x^p \, \psi_{j,l}(x) \, dx = 0 \ \text{ for } \ 0 \leq p < \tilde{N}.$$

Because of translation and dilation, if the mother wavelet has $\tilde{N}$ vanishing moments, then all wavelets do. As a results all detail functions $d_j(x)$ in a wavelet representation also have $\tilde{N}$ vanishing moments and all coarser versions $s_j(x)$ of a function $s_n(x)$ have the first $\tilde{N}$ moments independent of j:

$$\int_{-\infty}^{+\infty} x^p \, s_j(x) \, dx = \int_{-\infty}^{+\infty} x^p \, s_{j+1}(x) \, dx.$$

Remark: Readers who are familiar with the more traditional signal processing introduction to wavelets will note that order and dual order relate to the localization of the signals in frequency. Typically the coarser function $s_{n-1}(x)$ will contain the lower frequency band while the detail function $d_{n-1}(x)$ contains the higher frequency band. The order of a MRA is related to the smoothness of the scaling functions and thus to how much aliasing occurs from the lower band to the higher band. The dual order corresponds to the cancellation of the wavelets and thus to how much aliasing occurs from the higher band to the lower. In the lifting scheme, the predict part ensures a certain order, while the update part ensures a particular dual order.

Chapter 2

Second Generation Wavelets

2.1 Introduction

In the first chapter we only considered the regular setting, i.e., all samples are equally spaced, and subdivision always puts new samples in the middle between old samples. Consequently a sample value $s_{j,k}$ "lives" at the location $k\,2^{-j}$ and all the scaling functions and wavelets we constructed were dyadic translates and dilates of one fixed "mother" function. We refer to these as *first generation wavelets*. We described the construction of wavelets with the help of the lifting scheme, which only uses techniques of the spatial domain. However, historically first generation wavelets were always constructed in the frequency domain with the help of the Fourier transform, see e.g. [9]. All the wavelets and scaling functions we described in the first chapter can be derived with these classical methods. Using the lifting scheme though makes it very straightforward to build wavelets and scaling functions in much more general settings in which the Fourier transform is not applicable anymore as a construction tool.

In the following sections we consider more general settings such as boundaries, irregular samples, and arbitrary weight functions. These cases do not allow for wavelets which are translates and dilates of one fixed function, i.e., the wavelets at an interval boundary are not just translates of nearby wavelets. This lack of translation and dilation invariance requires new construction tools, such as lifting, to replace the Fourier transform. Lifting constructions are performed entirely in the spatial domain and can be applied in the more general, irregular settings. Even though the wavelets which result from using the lifting scheme in the more general settings will not be translates and dilates of one function anymore, they still have all the powerful properties of first generation wavelets: fast transforms, localization, and good approximation. We therefore refer to them as *Second Generation Wavelets* [26].

The purpose of the remainder of this chapter is to show that subdivision, interpolation, and lifting taken together result in a versatile and straightforward to implement second generation wavelets toolkit. All the algorithms can be derived via simple arguments involving little more than the manipulation of polynomials (as already seen in the first chapter.)

We focus on three settings which lead to Second Generation Wavelets:

- **Intervals:** When working with finite data it is desirable to have basis functions adapted to life on an interval. This way no awkward solutions such as zero padding, periodization, or reflection are needed. We point out that many wavelet constructions on the interval already exist, see [1, 6, 3], but we would like to use the subdivision schemes adapted to boundaries since they lead to more straightforward constructions and implementations.

- **Irregular samples:** In many practical applications, the samples do not necessarily live on a regular grid. Resampling is fraught with pitfalls and may even be impossible. A basis and transform adapted to the irregular grid is desired.

- **Weighted inner products:** Often one needs a basis adapted to a weighted inner product instead of the regular L_2 inner product. A weighted inner product of two functions f and g is defined as
$$\langle f, g \rangle = \int w(x)\, f(x)\, g(x)\, dx,$$
where $w(x)$ is some positive function. Weighted wavelets are very useful in the solution of boundary value ODEs, see [25]. Also, as we will see later, they are useful in the approximation of functions with singularities.

Obviously, we are also interested in combinations of these settings. Once we know how to handle each separately, combined settings can be dealt with easily.

2.2 Interpolating Subdivision and Scaling Functions

2.2.1 Interval Constructions

Recall that interpolating subdivision assembles $N = 2D$ coefficients $s_{j,k}$ in each step. These uniquely define a polynomial $p(x)$ of degree $N-1$. This polynomial is then used to generate

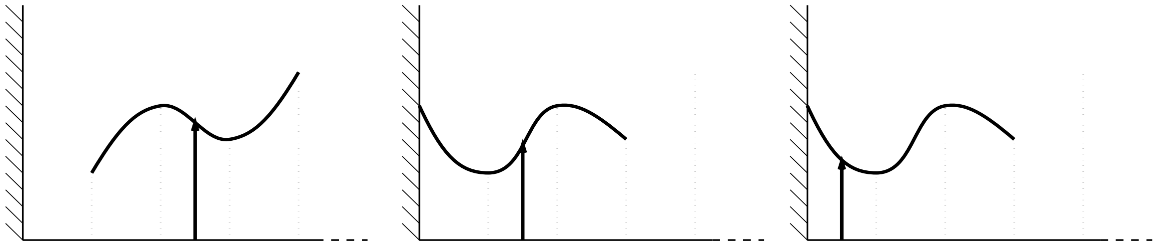

Figure 2.1: *Behavior of the cubic interpolating subdivision near the boundary. The midpoint samples between $k = 2, 3$ and $k = 1, 2$ are unaffected by the boundary. When attempting to compute the midpoint sample for the interval $k = 0, 1$ we must modify the procedure since there is no neighbor to the left for the cubic interpolation problem. Instead we choose 3 neighbors to the right. Note how this results in the same cubic polynomial as used in the definition of the midpoint value $k = 1, 2$, except this time it is evaluated at $1/2$ rather than $3/2$. The procedure clearly preserves the cubic reconstruction property even at the interval boundary and is thus the natural choice for the boundary modification.*

one new coefficient $s_{j+1,l}$. The new coefficient is located in the middle of the N old coefficients. When working on an interval the same principle can be used as long as we are sufficiently far from the boundary. Close to the boundary we need to adapt this scheme. Consider the case where one wants to generate a new coefficient $s_{j+1,l}$, but is unable to find the same number of old samples $s_{j,k}$ to the left as to the right of the new sample, simply because they are not available. The basic idea is then to choose, from the set of available samples $s_{j,k}$, those N which are closest to the new coefficient $s_{j+1,l}$.

To be concrete, take the interval $[0, 1]$. We have $2^j + 1$ coefficients $s_{j,k}$ at locations $k\,2^{-j}$ for $0 \leq k \leq 2^j$. The left most coefficient $s_{j+1,0}$ is simply $s_{j,0}$. The next one, $s_{j+1,1}$ is found by constructing the interpolating polynomial to the points $(x_{j,k}, s_{j,k})$ for $0 \leq k < N$ and evaluating it at $x_{j+1,1}$. For $s_{j+1,2}$ we evaluate the same polynomial p at $x_{j+1,2}$. Similar constructions work for the other N boundary coefficients and the right side. Figure 2.1 shows this idea for a concrete example.

2.2.2 Irregular Samples

The case of irregular samples can also be accommodated by observing that interpolating subdivision does not require the samples to be on a regular grid. We can take an arbitrarily spaced

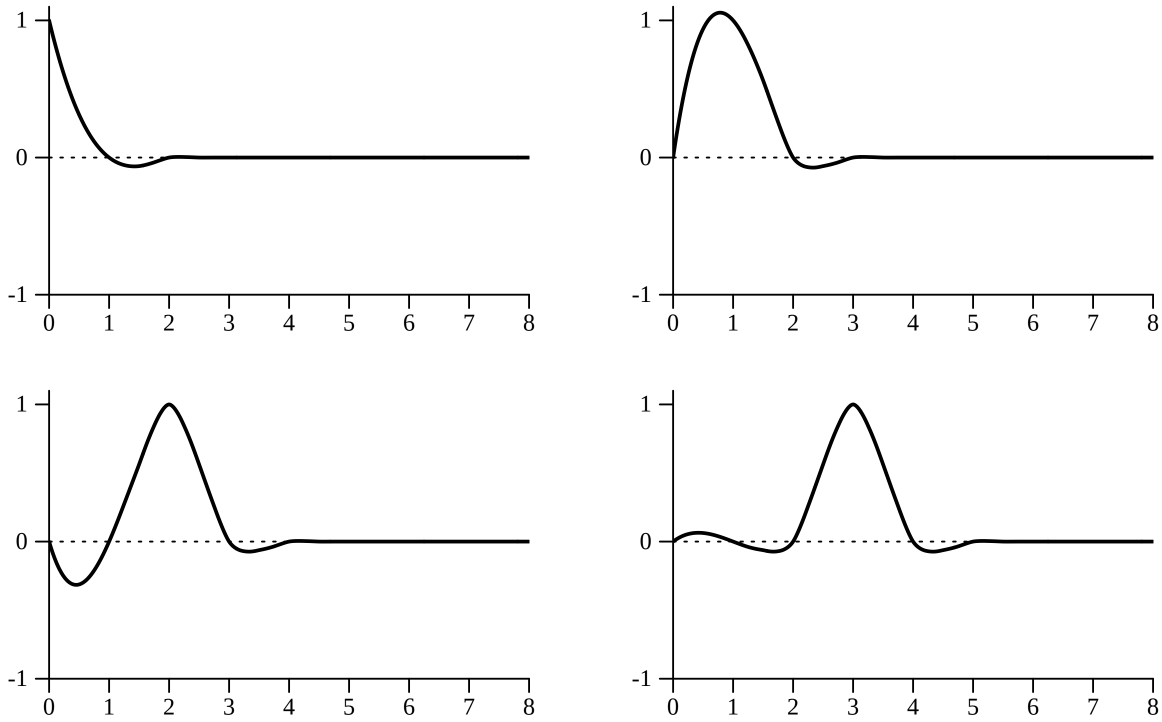

Figure 2.2: *Examples of scaling functions affected by a boundary. Left to right top to bottom scaling functions of cubic ($N = 4$) interpolation with $k = 0, 1, 2, 3$. Note how the boundary scaling functions are still interpolating as one would expect.*

set of points $x_{j,k}$ with $x_{j+1,2k} = x_{j,k}$ and $x_{j,k} < x_{j,k+1}$. A coefficient $s_{j,k}$ lives at the location $x_{j,k}$. The subdivision schemes can now be applied in a straightforward manner.

2.2.3 Weighted Inner Products

Interpolating subdivision does not involve an inner product, hence a weighted inner product does not change the subdivision part. However, the update part does change since it involves integrals of scaling functions, as we shall see soon. We postpone the details of this until after the general section on computing update weights.

2.2.4 Scaling Functions

As in the first generation case, we define the scaling function $\varphi_{j,k}$ to be the result of running the subdivision scheme ad infinitum starting from a sequence $s_{j,k'} = \delta_{j,k'}$. The main difference with the first generation case is that, because of the irregular setting, the scaling functions are not necessarily translates and dilates of each other. For example, Figure 2.2 shows the scaling functions affected by the boundary. The main feature of the second generation setting is that the powerful properties such as approximation order, the refinement relation, and the connection with wavelets remain valid. We summarize the main properties:

- The limit function of the subdivision scheme started at level j with coefficients $s_{j,k}$ can be written as

$$f = \sum_k s_{j,k}\, \varphi_{j,k}.$$

- The scaling functions are compactly supported.

- The scaling functions are interpolating:

$$\varphi_{j,k}(x_{j,l}) = \delta_{k,l}.$$

- Polynomials upto degree $N-1$ can be written as linear combinations of the scaling functions at level j. Because of the interpolating property, the coefficients are samples of the polynomial:

$$\sum_k x_{j,k}^p\, \varphi_{j,k} = x^p \text{ for } 0 \leq p < N.$$

- Very little is known about the smoothness of the resulting scaling functions in the irregular case. Recall that they are defined as the limit of a fully nonstationary subdivision scheme. Work in progress though suggests that with some very reasonable conditions on the weight function or the placement of the sample locations, one can obtain roughly the same smoothness as in the regular case.

- They satisfy refinement relations. Start the subdivision on level j with $s_{j,k} = \delta_{j,k}$. We know that the subdivision scheme converges to $\varphi_{j,k}$. Now do only one step of the subdivision scheme. Call the resulting coefficients $h_{j,k,l} = s_{j+1,l}$. Only a finite number are non zero.

Since starting the subdivision scheme at level $j+1$ with the $\{h_{j,k,l} \mid l\}$ coefficients also converges to $\varphi_{j,k}$, we have that

$$\varphi_{j,k}(x) = \sum_l h_{j,k,l}\, \varphi_{j+1,l}(x). \tag{2.1}$$

Note how in this case the coefficients of the refinement relation are (in general) different for each scaling function. Compare this with the first generation setting of Equation (1.2), where $h_{j,k,l} = h_{l-2k}$. In that setting the $h_{j,k,l}$ are translation and dilation invariant.

Depending on the circumstances there are two basic ways of performing this subdivision process. The more general way is to always construct the respective interpolating polynomials on the fly using an algorithm such as Neville. This has the advantage that none of the sample locations have to be known beforehand.

However, in case the sample locations are fixed and known ahead of time, one can precompute the subdivision, or filter, coefficients. These will be the same as the ones from the refinement relation: assume we are given the samples $\{s_{j,k} \mid k\}$ and we want to compute the $\{s_{j+1,l} \mid l\}$. Given that the whole process is linear, we can simply use superposition. A 1 at location (j,k) would, after subdivision, give the sequence $\{h_{j,k,l} \mid l\}$. Superposition now immediately leads to:

$$s_{j+1,l} = \sum_k h_{j,k,l}\, s_{j,k}. \tag{2.2}$$

This is an equivalent formulation of the subdivision. It requires the precomputation of the $h_{j,k,l}$. These can be found once offline by using the polynomial interpolation algorithm of Neville.

Compare Equations (2.1) and (2.2) which use the same coefficients $h_{j,k,l}$. In (2.1) the summation ranges over l and the equation allows us to go from a fine level scaling function to a coarse level scaling function. In (2.2) the summation ranges over k and the equation allows us to go from coarse level samples to fine level samples.

2.3 The Unbalanced Haar Transform

Before we discuss how average-interpolation works in the second generation setting, we first take a look at an example. The *Unbalanced Haar transform* is the generalization of the Haar wavelet to the second generation setting [17]. The first difference with the interpolating case is that a coefficient $s_{j,k}$ does not "live" at location $x_{j,k}$ any more, but rather on the interval $[x_{j,k}, x_{j,k+1}]$.

We define the generalized length of an interval $[x_{j,k}, x_{j,k+1}]$ as

$$I_{j,k} = \int_{x_{j,k}}^{x_{j,k+1}} w(x)\, dx,$$

where $w(x)$ is some weight function. From the definition, it immediately follows that

$$I_{j-1,k} = I_{j,2k} + I_{j,2k+1}. \tag{2.3}$$

With this definition $I_{j,k}$ measures the weight given to a particular interval and its associated coefficient $s_{j,k}$. Given a signal s_j, the important quantity to preserve is not so much the average of the coefficients, but their *weighted* average:

$$\sum_k I_{j,k}\, s_{j,k}$$

With this in mind we can define the generalization of the Haar transform, which is called the *Unbalanced Haar Transform*. The detail wavelet coefficient is still computed as before:

$$d_{j-1,k} = s_{j,2k+1} - s_{j,2k},$$

but the average is now computed as a weighted average:

$$s_{j-1,k} = \frac{I_{j,2k}\, s_{j,2k} + I_{j,2k+1}\, s_{j,2k+1}}{I_{j-1,k}}.$$

Defining the transform this way assures two things:

1. If the original signal is a constant, then all detail coefficients are zero and all coarser versions are constants as well. This follows from the definition of the transform and Equation (2.3).

2. The weighted average of all coarser signals is the same, i.e.

   $$\sum_l I_{j,l}\, s_{j,l}$$

   does not depend on j. This follows from computing the coarser signals as weighted averages.

Later we will see that the order of the Unbalanced Haar MRA as well as its dual order are one.

Next we have to cast this in the lifting framework of split, predict, and update. The split divides the signal in even and odd indexed coefficients. The prediction for an odd coefficient $s_{j,2k+1}$ is its left neighboring even sample $s_{j,2k}$ which leads to the detail coefficient:

$$d_{j-1,k} = s_{j,2k+1} - s_{j,2k}.$$

In the update step, the coarser level is computed as [26]

$$s_{j-1,k} = s_{j,2k} + \frac{I_{j,2k+1}}{I_{j-1,k}}\, d_{j-1,k}.$$

Using Equation (2.3), one can see that this is equivalent to the weighted average computation.

2.4 Average-Interpolating Subdivision

In this section we discuss how average-interpolation works in the second generation case as introduced in [25]. The setting is very similar to the first generation case. We first describe the average-interpolating subdivision scheme and then show how this fits into the lifting strategy.

We start by assuming that we are given weighted averages of some unknown function over intervals

$$s_{n,l} = \frac{1}{I_{n,l}} \int_{x_{n,l}}^{x_{n,l+1}} w(x)\, f(x)\, dx.$$

Just as before we define average-interpolating subdivision through the use to higher order polynomials, with the first interesting choice being quadratic. For a given interval consider the intervals to its left and right. Define the (unique) quadratic polynomial $p(x)$ so that

$$\begin{aligned}
s_{j,k-1} &= \frac{1}{I_{j,k-1}} \int_{x_{j,k-1}}^{x_{j,k}} w(x)\, p(x)\, dx \\
s_{j,k} &= \frac{1}{I_{j,k}} \int_{x_{j,k}}^{x_{j,k+1}} w(x)\, p(x)\, dx \\
s_{j,k+1} &= \frac{1}{I_{j,k+1}} \int_{x_{j,k+1}}^{x_{j,k+2}} w(x)\, p(x)\, dx.
\end{aligned}$$

Now compute $s_{j+1,2k}$ and $s_{j+1,2k+1}$ as the average of this polynomial over the left and right subintervals of $[x_{j,k}, x_{j,k+1}]$

$$\begin{aligned}
s_{j+1,2k} &= \frac{1}{I_{j+1,2k}} \int_{x_{j+1,2k}}^{x_{j+1,2k+1}} w(x)\, p(x)\, dx \\
s_{j+1,2k+1} &= \frac{1}{I_{j+1,2k+1}} \int_{x_{j+1,2k+1}}^{x_{j+1,2k+2}} w(x)\, p(x)\, dx.
\end{aligned}$$

It is easy to see that the procedure will reproduce quadratic polynomials. Assume that the initial averages $\{s_{0,k}\}$ were weighted averages of a given quadratic polynomial $P(x)$. In that case the unique polynomial $p(x)$ which has the prescribed averages over each triple of intervals will always be that same polynomial $P(x)$ which gave rise to the initial set of averages. Since the interval sizes go to zero and the averages over the intervals approach the value of the underlying function in the limit the original quadratic polynomial $P(x)$ will be reproduced.

Higher order schemes can be constructed similarly. We construct a polynomial p of degree $N-1$ (where $N = 2D+1$) so that

$$s_{j,k+l} = \frac{1}{I_{j,k+l}} \int_{x_{j,k+l}}^{x_{j,k+l+1}} w(x)\, p(x)\, dx \ \text{ for } \ -D \leq l \leq D,$$

Then we calculate two coefficients on the next finer level as

$$\begin{aligned} s_{j+1,2k} &= \frac{1}{I_{j+1,2k}} \int_{x_{j+1,2k}}^{x_{j+1,2k+1}} w(x)\, p(x)\, dx \\ s_{j+1,2k+1} &= \frac{1}{I_{j+1,2k+1}} \int_{x_{j+1,2k+1}}^{x_{j+1,2k+2}} w(x)\, p(x)\, dx. \end{aligned}$$

The computation is very similar to the first generation case, except for the fact that the polynomial problem cannot be recast into a Neville algorithm any longer since the integral of a polynomial times the weight function is not necessarily a polynomial.

These algorithms take care of the weighted setting and the irregular samples setting. In the case of an interval construction, we follow the same philosophy as in the interpolating case. We need to assemble N coefficients to determine an average-interpolating polynomial. In case we cannot align them symmetrically around the new samples, as at the end of the interval, we simply take more from one side than the other. This idea is illustrated in Figure 2.3.

Next we cast average-interpolation into the lifting framework. We use the average-interpolating subdivision as a P box *before* entering the inverse Unbalanced Haar transform. The diagram in Figure 1.9 illustrates this setup. Instead of computing $s_{j+1,2k}$ and $s_{j+1,2k+1}$ directly we will compute their difference $d_{j,k} = s_{j+1,2k+1} - s_{j+1,2k}$ and feed this as a difference signal into the inverse Unbalanced Haar transform. Given that the weighted average of $s_{j+1,2k}$ and $s_{j+1,2k+1}$, as computed by average-interpolation is $s_{j,k}$, it follows that the inverse Unbalanced Haar transform when given $s_{j,k}$ and $d_{j,k}$ will compute $s_{j+1,2k}$ and $s_{j+1,2k+1}$ as desired.

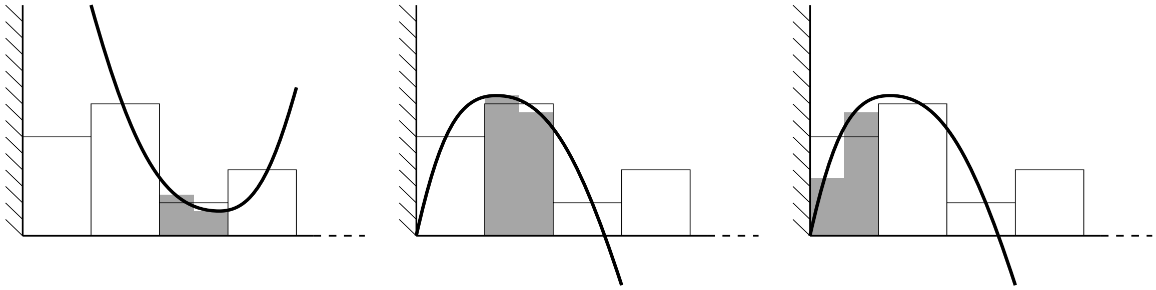

Figure 2.3: *Behavior of the quadratic average-interpolation process near the boundary. The averages for the subintervals $k = 2$ and $k = 1$ are unaffected. When attempting to compute the finer averages for the left most interval the procedure needs to be modified since no further average to the left of $k = 0$ exists for the average-interpolation problem. Instead we use 2 intervals to the right of $k = 0$, effectively reusing the same average-interpolating polynomial constructed for the subinterval averages on $k = 1$. Once again it is immediately clear that this is the natural modification to the process near the boundary, since it insures that the crucial quadratic reproduction property is preserved.*

2.5 Average-Interpolating Scaling Functions

As before an average-interpolating scaling function $\varphi_{j,k}(x)$ is defined as the limit function of the subdivision process started on level j with the sequence $\delta_{j,k}$. We here list the main properties:

- The limit function of the subdivision scheme started at level j with coefficients $s_{j,k}$ can be written as

$$f = \sum_k s_{j,k}\, \varphi_{j,k}.$$

- The scaling functions are compactly supported.

- The scaling functions are average-interpolating:

$$\frac{1}{I_{j,l}} \int_{x_{j,l}}^{x_{j,l+1}} w(x)\, \varphi_{j,k}(x)\, dx = \delta_{k,l}.$$

- The scaling functions of level j reproduce polynomials upto degree $N-1$.

$$\sum_k c_{j,k}\, \varphi_{j,k} = x^p \text{ for } 0 \le p < N$$

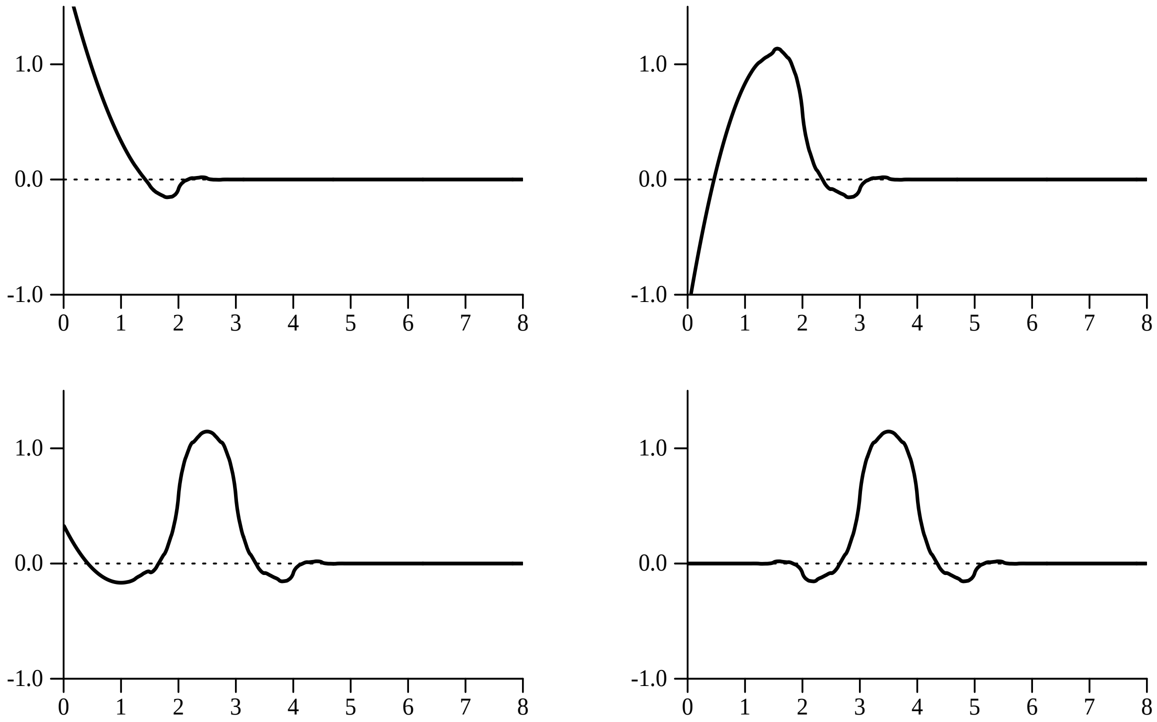

Figure 2.4: *Examples of scaling functions affected by a boundary. Left to right, top to bottom scaling functions of quadratic ($N = 3$) average-interpolation at $k = 0, 1, 2, 3$. The functions continue to have averages over each integer subinterval of either* 0 *or* 1.

with coefficients

$$c_{j,k} = \frac{1}{I_{j,k}} \int_{x_{j,k}}^{x_{j,k+1}} w(x)\, x^p\, \varphi_{j,k}(x)\, dx.$$

- The scaling functions satisfy refinement relations:

$$\varphi_{j,k} = \sum_l h_{j,k,l}\, \varphi_{j+1,l}. \tag{2.4}$$

Figure 2.4 shows the average-interpolating boundary functions.

2.6 Cubic B-spline Scaling Functions

In the case of cubic B-splines we need to worry about the endpoints of a finite sized interval. Because of their support the scaling functions close to the endpoints would overlap the outside of the interval. This issue can be addressed in a number of different ways. One treatment, used by Chui and Quak [3], uses multiple knots at the endpoints of the interval. The appropriate subdivision weights then follow from the evaluation of the de Boor algorithm for those control points. The total number of scaling functions at level j becomes $2^j + 3$ in this setting. Consequently it is not so easy anymore to express everything in a framework which is based on insertion of new control points inbetween old ones. We used a different treatment which preserves this property.

The P_{odd} box remains as before at the boundary—every odd location still has an even neighbor on either side—but we change the P_{even} box since the left- (right-) most even position has only one odd neighbor. In this case the P_{even} box makes no contribution to the boundary control point and furthermore the boundary control point does not get rescaled. This leads to endpoint interpolating piecewise cubic polynomial scaling functions as shown in Figure 2.5.

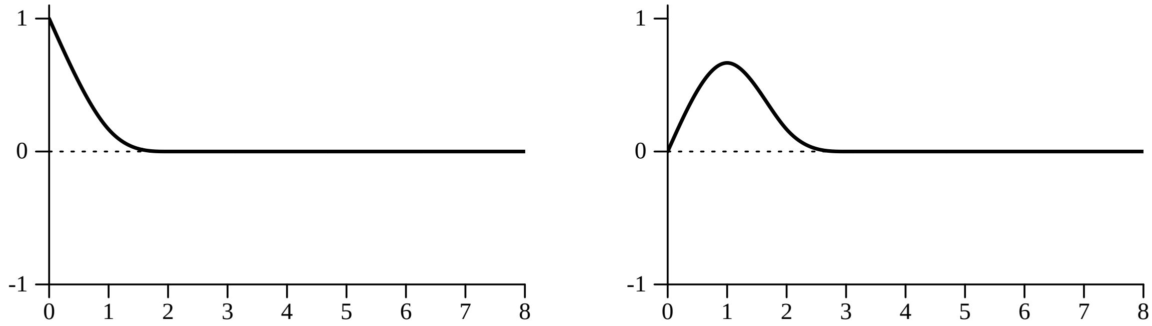

Figure 2.5: *In the case of cubic B-splines only the two leftmost splines change for the particular adaptation to the boundary which we chose.*

2.7 Multiresolution Analysis

Now that we have defined subdivision and scaling functions in the second generation setting, it is a small step to multiresolution analysis. Remember that the result of the subdivision algorithm started from level j can always be written as a linear combination of scaling functions.

$$s_j(x) = \sum_k s_{j,k}\, \varphi_{j,k}(x).$$

The definition of the V_j spaces is now exactly the same as in the first generation case:

$$V_j = \operatorname{span}\{\varphi_{j,k} \mid 0 \le k < K_j\}.$$

We assume K_j scaling functions on level j. It follows from the refinement relations (2.1) that the spaces are nested:

$$V_j \subset V_{j+1}.$$

Again we want that any function of finite energy ($\in \mathrm{L}_2$) can be approximated arbitrarily closely with scaling functions. Mathematically we write this as

$$\bigcup_{j>0} V_j \text{ is dense in } \mathrm{L}_2.$$

The *order* of the MRA is defined similarly to the first generation case. We say that the order is N in case every polynomial of degree less than N can be written as a linear combination of scaling functions on a given level. The subdivision schemes we saw in the previous sections have order N where N is odd for interpolating subdivision and even for average-interpolating subdivision.

Integrals of Scaling Functions We will later see that in order to build wavelets, it is important to know the integral of each scaling function. In the first generation case this is not an issues. Due to translation and dilation the integral of $\varphi_{j,l}(x)$ is always 2^{-j}. In the second generation case, the integrals are not given by such a simple rule. Therefore we need an algorithm to compute them. We define:

$$M_{j,k} := \int_{-\infty}^{+\infty} w(x)\, \varphi_{j,k}(x)\, dx.$$

The computation goes in two phases. We first approximate the integrals on the finest level n numerically using a simple quadrature formula. The ones on the coarser levels $j < n$ can be computed iteratively. From integrating the refinement relation (2.1) it immediately follows that

$$M_{j,k} = \sum_l h_{j,k,l}\, M_{j+1,l}.$$

Once we computed the $h_{j,k,l}$ coefficients, the recursive computation of the integrals of the scaling functions is straightforward. We will later need them in the update stage.

2.8 Lifting and Interpolation

In this section, we discuss how to use the lifting scheme to compute second generation wavelet transforms. The steps will be exactly the same as in the first generation case: split, predict, update.

The split stage again is the *Lazy wavelet transform*. It simply consists of splitting the samples $s_{j,l}$ into the even indexed samples $s_{j,2k}$ and the odd indexed samples $s_{j,2k+1}$.

In the predict stage we take the even samples and use interpolating subdivision to predict each odd sample. The detail coefficient is the difference of the odd sample and its predicted value. Suppose we use a subdivision scheme of order $N = 2D$ to build the predictor. The detail coefficient is then computed as

$$d_{j-1,k} := s_{j,2k+1} - p(x_{j,2k+1}),$$

where $p(x)$ is the interpolating polynomial of degree $N-1$ which interpolates the points $(x_{j-1,k+l}, s_{j-1,k+1})$ with $-D+1 \leq l \leq D$. Thus if the original signal is a polynomial of degree strictly less than N, the detail signal is exactly zero.

The purpose of the update stage is to preserve the weighted average on each level; we want that

$$\int_{-\infty}^{+\infty} \sum_k s_{j,k}\, \varphi_{j,k}(x)\, dx = \sum_k s_{j,k}\, M_{j,k}$$

does not depend on the level j. According to the lifting scheme we do this using the detail computed in the previous step. We propose an update step of the form:

$$s_{j-1,k} := s_{j,2k} + A_{j,k-1}\, d_{j,k-1} + A_{j,k}\, d_{j,k}. \tag{2.5}$$

In order to find the $A_{j.k}$, assume we run the inverse transform from level $j-1$ to j starting with all $s_{j-1,l}$ zero and only one $d_{j-1,k}$ non-zero. Then undoing the update and running the subdivision scheme should result in a function with integral zero. Undoing the update will result in two non zero even coefficients, $s_{j,2k}$ and $s_{j,2k+2}$. Undoing the update involves computing:

$$\begin{aligned} s_{j,2k} &:= s_{j-1,k} - A_{j,k-1}\, d_{j,k-1} - A_{j,k}\, d_{j,k} \\ s_{j,2k+2} &:= s_{j-1,k+1} - A_{j,k}\, d_{j,k} - A_{j,k+1}\, d_{j,k+1}. \end{aligned}$$

Given that only $d_{j,k}$ is non zero, we have that $s_{j,2k} = -A_{j,k}$ and $s_{j,2k+2} = -A_{j,k}$. Now running the subdivision scheme results in a function given by

$$\varphi_{j,2k+1}(x) - A_{j,k}\, \varphi_{j-1,k}(x) - A_{j,k}\, \varphi_{j-1,k+1}(x). \tag{2.6}$$

This function has to have integral zero. Thus:

$$A_{j,k} = \frac{M_{j,2k+1}}{M_{j-1,k} + M_{j-1,k+1}}.$$

This shows us how to choose $A_{j,k}$.

One can also build more powerful update methods, which not only assure that the integral of the $s_j(x)$ functions is preserved, but also their first (generalized) moment:

$$\int_{-\infty}^{+\infty} w(x)\, x\, s_j(x)\, dx.$$

This requires the calculation of the first order moments of the scaling functions, which can be done analogously to the integral calculations. The lifting (2.6) then has different lifting weights for $\varphi_{j-1,k}$ and $\varphi_{j-1,k+1}$ (as opposed to $A_{j,k}$ for both) which can be found by solving a 2×2 linear system.

2.9 Wavelet functions

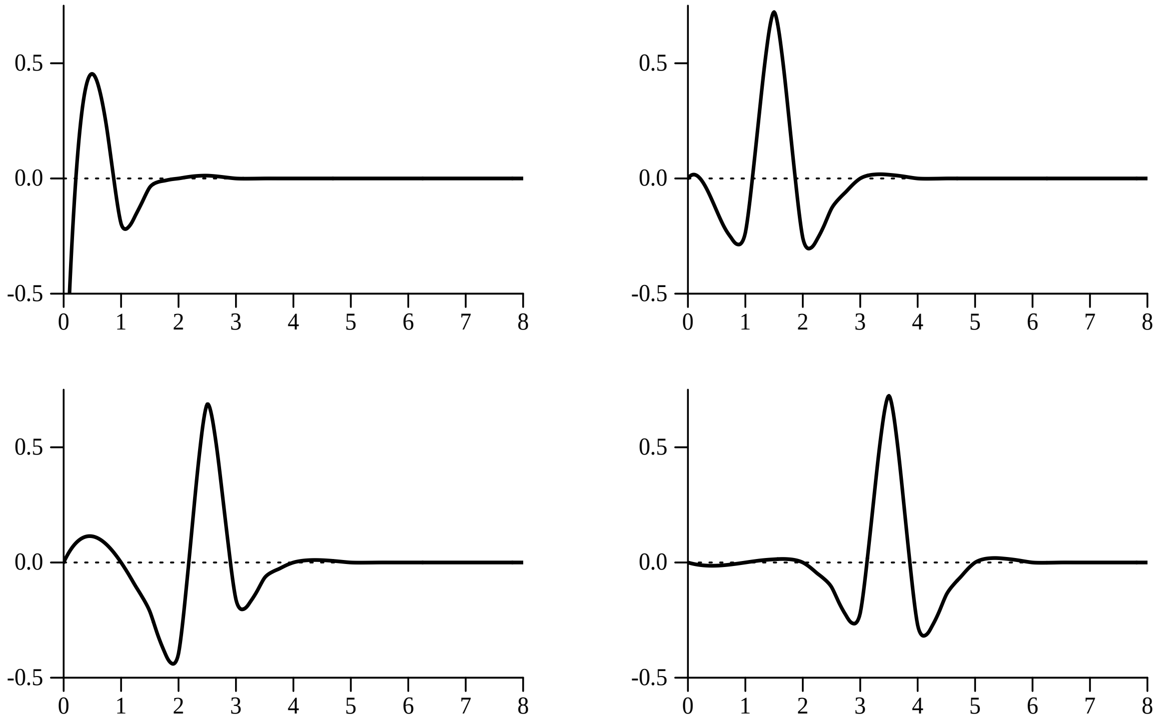

Figure 2.6: *Examples of wavelets affected by a boundary. Going left to right, top to bottom wavelets with $\tilde{N} = 2$ vanishing moments and order $N = 4$ for $k = 0, 1, 2, 3$.*

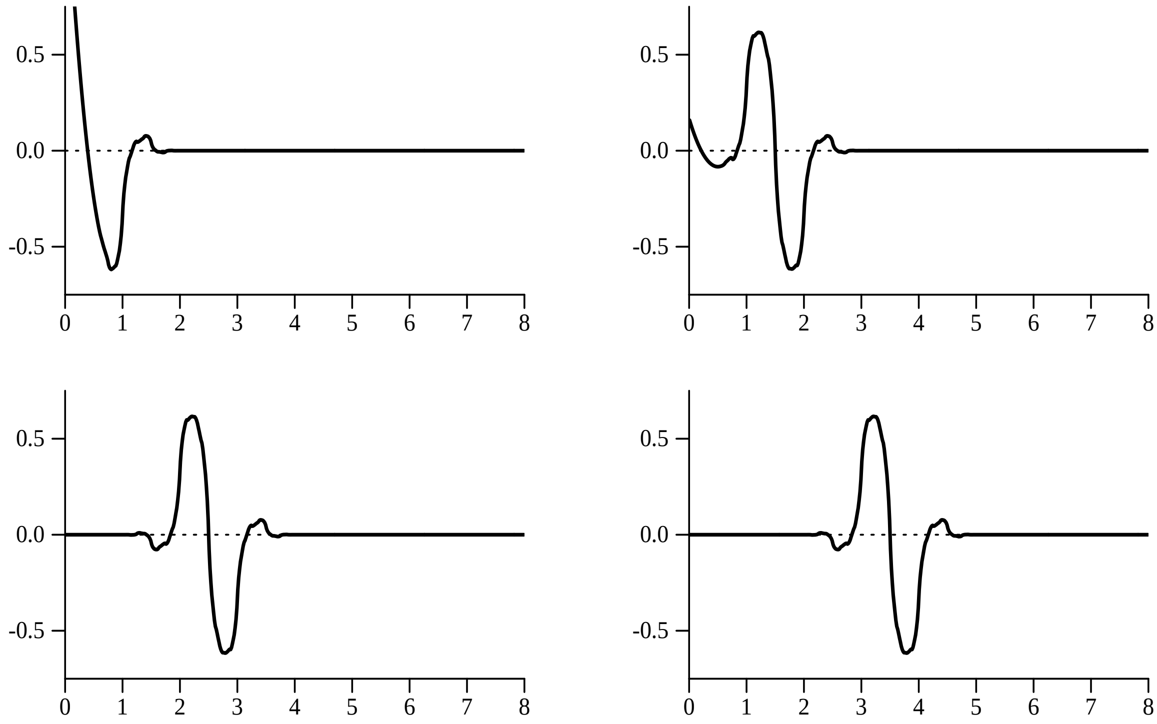

Figure 2.7: *Examples of wavelets affected by a boundary. Going left to right, top to bottom wavelets with* $\tilde{N} = 1$ *vanishing moment and order* $N = 3$ *for* $k = 0, 1, 2, 3$.

We can now define wavelet functions exactly the same way as in the first generation case. Compute an inverse wavelet transform from level j to level $j+1$ with all coarse scale coefficients $s_{j,l}$ set to zero and only one detail coefficient $d_{j,k}$ set to one while the other detail coefficients are zero. Call the resulting sequence $\{g_{j,k,l} \mid l\}$. Then run the subdivision algorithm. The resulting function is the wavelet $\psi_{j,k}$:

$$\psi_{j,k} = \sum_l g_{j,k,l}\, \varphi_{j+1,l}. \tag{2.7}$$

Figures 2.6, 2.7, and 2.8 show the wavelets affected by the boundary in the interpolating, average-interpolation, and cubic B-spline case respectively. In the interpolating case $N = 4$ and the wavelets, built with lifting, have $\tilde{N} = 2$ vanishing moments. In the average-interpolation case $N = 3$ and the wavelets have $\tilde{N} = 1$. For the B-splines the wavelets have $\tilde{N} = 2$ vanishing moments.

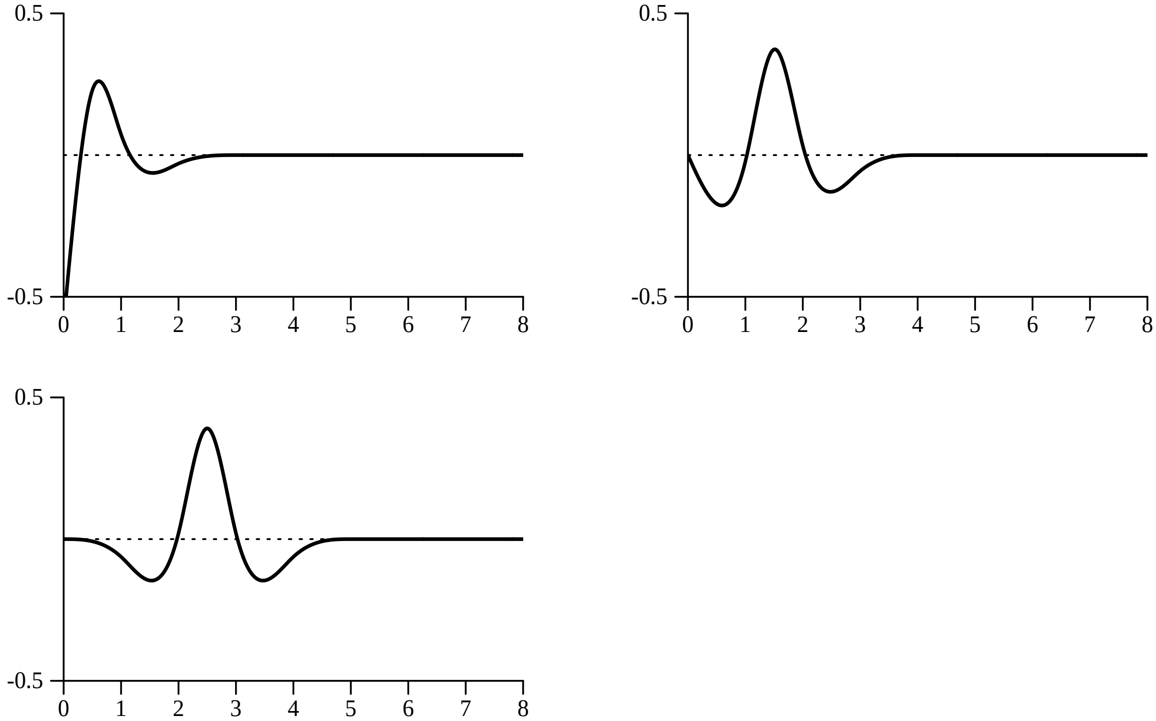

Figure 2.8: *The two left most (top row) wavelets are influenced by the boundary. Thereafter they default to the usual B-spline wavelets (bottom left).*

Define the detail function

$$d_j(x) = \sum_k d_{j,k}\, \psi_{j,k}(x).$$

It then follows that

$$s_{j+1}(x) = s_j(x) + d_j(x).$$

The multiresolution representation of a function $s_n(x)$ can now be written as

$$s_n(x) = s_0(x) + d_0(x) + d_1(x) + \ldots d_{n-1}(x) = s_0(x) + \sum_{j=0}^{n-1} \sum_k d_{j,k}\, \psi_{j,k}.$$

The main advantage of the wavelet transform is the fact that the expected value of the detail coefficient magnitudes is much smaller than the original samples. This is how we obtain a more compact representation of the original signal.

Remember that the order of a multiresolution analysis is N if the multiresolution representation of any polynomial of degree strictly less than N yields only zero detail signals. In other words if $s_n(x) = p(x)$ is a polynomial of degree less than N, then $s_n(x) = s_0(x)$ and all details are identically zero.

Another quantity which characterizes a multiresolution analysis is its *dual order*. We say that the dual order is $\tilde{N}$ in case all wavelets have $\tilde{N}$ vanishing moments or

$$\int_{-\infty}^{+\infty} w(x)\, x^p\, \psi_{j,k}(x)\, dx = 0 \text{ for } 0 \leq p < \tilde{N}.$$

We only consider the case where $\tilde{N} = 1$. The same techniques can be used for the more general case if so needed. Consequently all detail signals have a vanishing integral,

$$\int_{-\infty}^{+\infty} w(x)\, d_j(x)\, dx = 0.$$

This is equivalent to saying that

$$\int_{-\infty}^{+\infty} w(x)\, s_j(x)\, dx$$

is independent of the level j.

We define the subspaces W_j as

$$W_j = \text{span}\{\psi_{j,k} \mid 0 \leq k < K_{j+1} - K_j\},$$

then

$$V_{j+1} = V_j \oplus W_j.. \tag{2.8}$$

The dimension of W_j is thus the dimension of V_{j+1} (K_{j+1}) minus the dimension of V_j (K_j).

2.10 Applications

In this section we describe results of some experiments involving the ideas presented earlier. The examples were generated with a simple C code whose implementation is a direct transliteration of the algorithms described above. The only essential piece of code imported was an implementation of Neville's algorithm from Numerical Recipes [21]. All examples were computed on the unit interval, that is all constructions are adapted to the boundary as described earlier. The only code modification to accommodate this is to insure that the moving window of coefficients does not cross the left or right end point of the interval. The case of a weight function required somewhat more machinery which we describe in that section.

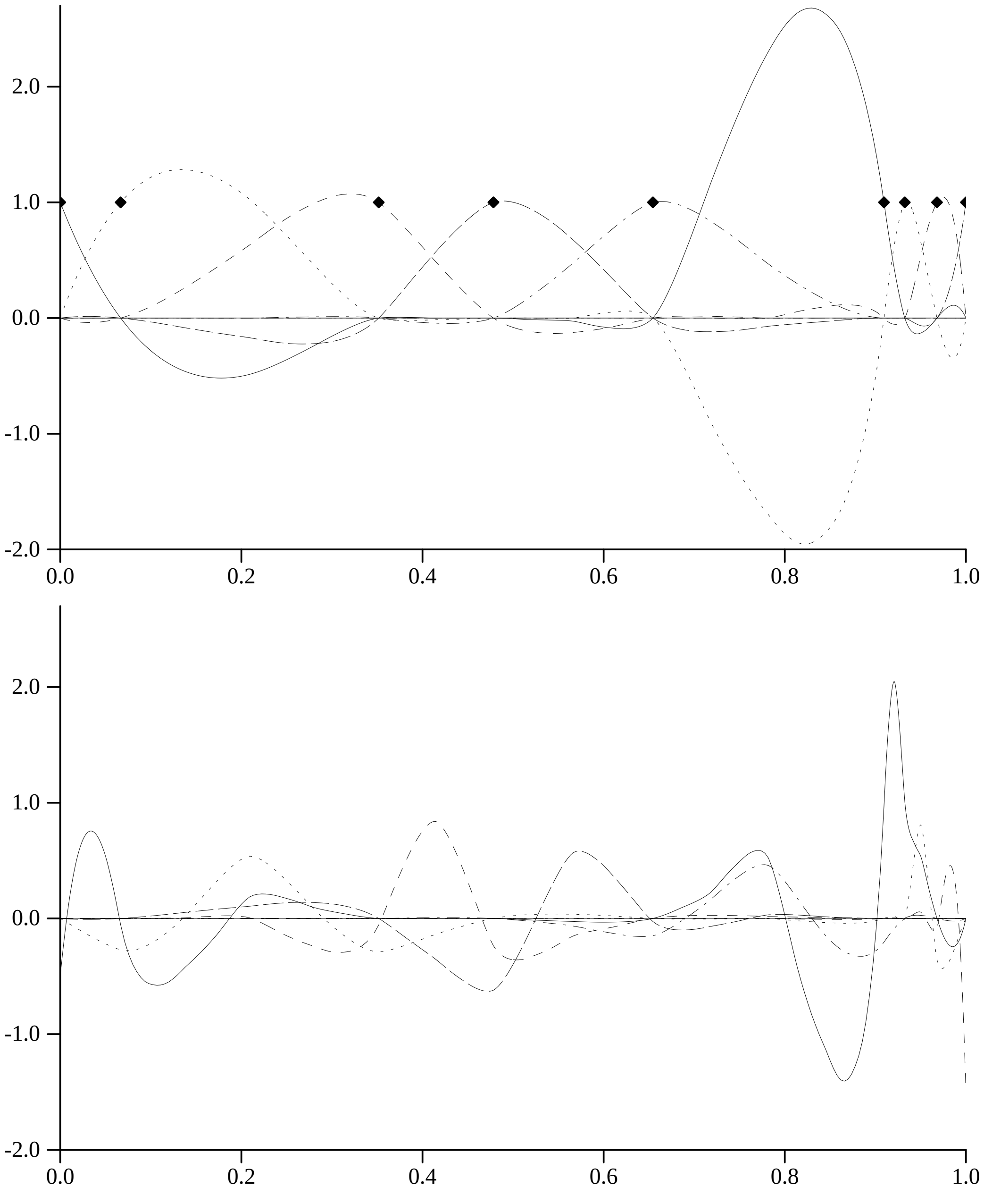

Figure 2.9: *Example of scaling functions (top) with $N = 4$ (interpolating subdivision) and wavelets (bottom) with $\tilde{N} = 2$ vanishing moments adapted to irregular sample locations. The original sample locations $x_{3,k}$ are highlighted with diamond marks. Note that one of the scaling functions is 1 at each of the marks, while all others are 0.*

2.10.1 Interpolation of Randomly Sampled Data

The first and simplest generalization concerns the use of $x_{j0,k}$ placed at random locations. Figure 2.9 shows the scaling functions (top) and wavelets (bottom) which result for such a set of random locations. The scaling functions are of order $N = 4$ (interpolating subdivision) and the wavelets have $\tilde{N} = 2$ vanishing moments. In this case we placed 7 uniformly random samples between $x_{3,0} = 0$ and $x_{3,8} = 1$. These locations are discernible in the graph as the unique points at which all scaling functions have a root save for one which takes on the value 1 (indicated by solid diamond marks). Sample points at finer levels were generated recursively by simply adding midpoints, i.e., $x_{j+1,2k+1} = 1/2\,(x_{j,k} + x_{j,k+1})$ for $j > 3$.

An interesting question is how the new sample points should be placed. A disadvantage of always adding midpoints is that imbalances between the lengths of the intervals are maintained. A way to avoid this is to place new sample points only in intervals whose length is larger than the average interval length. Doing so repeatedly will bring the ratio of largest to smallest interval length ever closer to 1.

Another possible approach would add new points so that the length of the intervals varies in a smooth manner, i.e., no large intervals neighbor small intervals. This can be done by applying an interpolating subdivision scheme, with integers as sample locations, to the $x_{j,k}$ themselves to find the $x_{j+1,2k+1}$. This would result in a smooth mapping from the integers to the $x_{j,k}$. After performing this step the usual interpolating subdivision would follow. Depending on the application one of these schemes may be preferable.

Next we took some random data over a random set of 16 sample locations and applied linear ($N = 2$) and cubic ($N = 4$) interpolating subdivision to them. The resulting interpolating functions are compared on the right side of Figure 2.10. These functions can be thought of as a linear superposition of the kinds of scaling functions we constructed above for the example $j = 3$.

Note how sample points which are very close to each other can introduce sharp features in the resulting function. We also note that the interpolation of order 4 exhibits some of the overshoot behavior one would expect when encountering long and steep sections of the curve followed by a reversal of direction. This behavior gets worse for higher order interpolation schemes. These experiments suggest that it might be desirable to enforce some condition on the ratio of the largest to the smallest interval in a random sample construction.

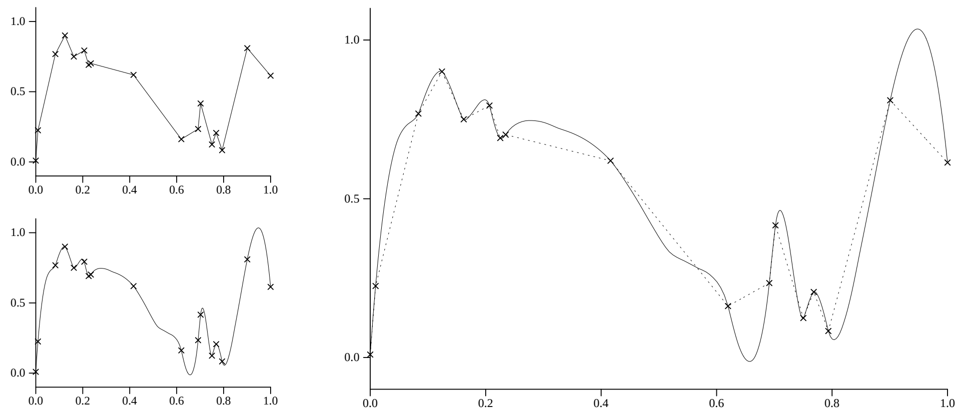

Figure 2.10: *Example of data defined at random locations $x_{4,k}$ on the unit interval and interpolated with interpolating subdivision of order $N = 2$ and 4 respectively.*

2.10.2 Smoothing of Randomly Sampled Data

A typical use for wavelet constructions over irregular sample locations is smoothing of data acquired at such locations. As an example of this we took 512 uniformly random locations on the unit interval (V_9) and initialized them with averages of $sin(3/4\pi x)$ with $\pm 20\%$ additive white noise. The resulting function is plotted on the top left of Figure 2.11 at level 9. The scaling functions used were based on average-interpolation with $N = 5$ and $\tilde{N} = 1$. Smoothing was performed by going to coarser spaces (lower index), setting all wavelet coefficients to zero and subdividing back out. From left to right, top to bottom the coarsest level used in the transform is V_9, V_7, V_5, and V_3.

We hasten to point out that this is is a very simple and naive smoothing technique. Depending on the application and knowledge of the underlying processes much more powerful smoothing operators can be constructed [15, 14]. This example merely serves to suggest that such operations can also be performed over irregular samples.

2.10.3 Weighted Inner Products

When we discussed the construction of scaling functions and wavelets we pointed out how a weight function in the inner product can be incorporated in the transform. The only change in the code is due to the fact that we cannot cast the average-interpolation problem into the form of a Neville interpolation algorithm anymore, since in general the integral of a polynomial

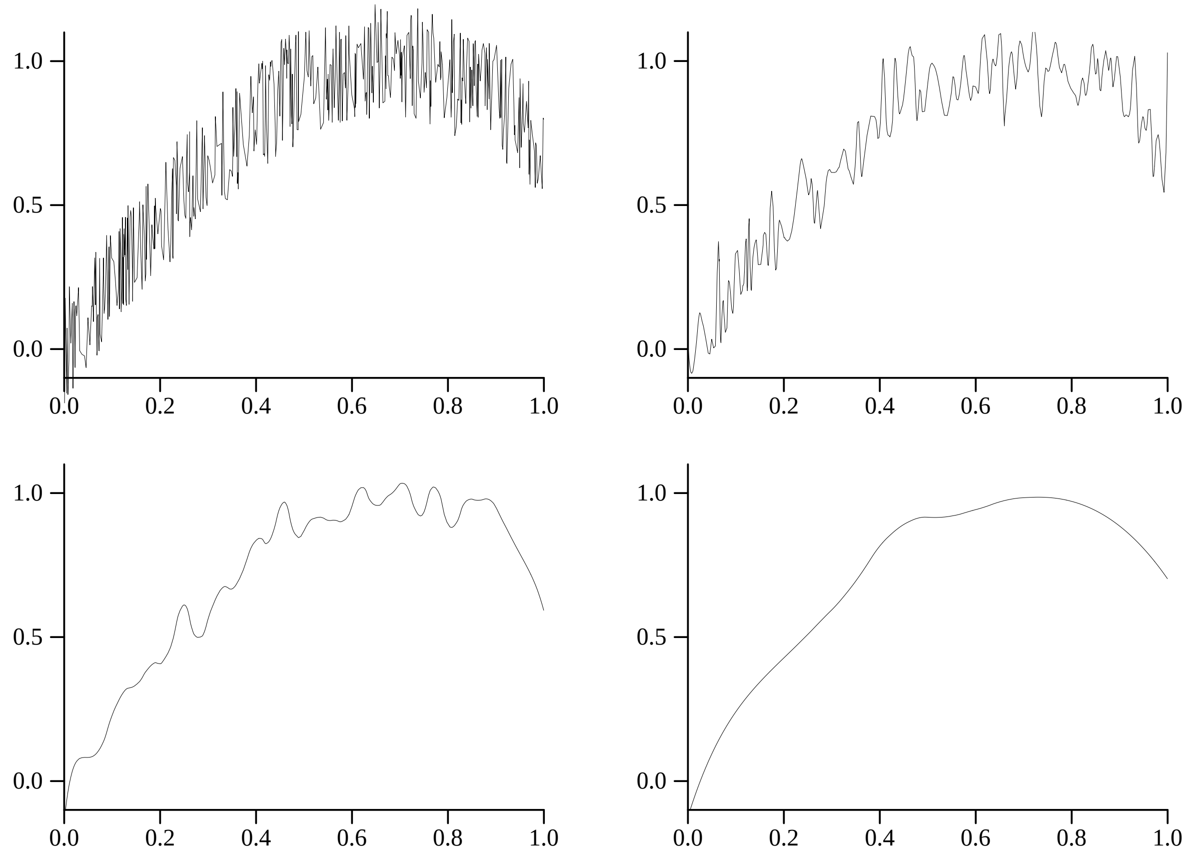

Figure 2.11: *A sine wave with additive noise sampled at uniformly distributed random locations in the unit interval and reconstructed with quintic average-interpolation (upper left). Successive smoothings are performed by going to coarser resolutions and cascading back out (upper right and bottom row).*

times the weight function is not another polynomial. Instead we first explicitly construct the polynomial p in the subdivision and use it to find the filter coefficients. This implies solving the underlying linear system which relates the coefficients of $p(x)$ to the observed weighted averages. Similarly, when lifting the interpolating wavelets to give them 2 vanishing moments the weighted moments of the scaling function enter. In both of these cases the construction of weighted bases requires additional code to compute moments and solve the linear systems involved in finding the filters. We saw in Section 2.7 how moment and integral calculations can

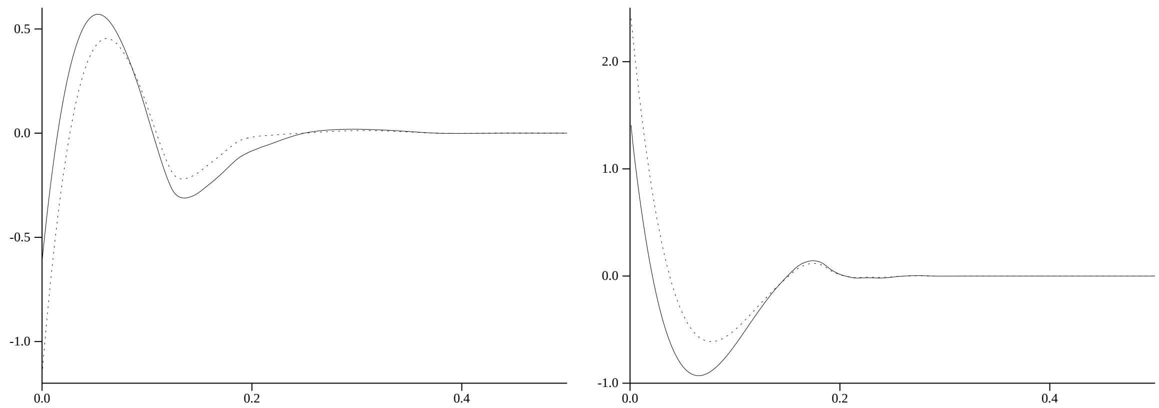

Figure 2.12: *Comparison of weighted (solid line) and unweighted (dotted line) wavelets at the left endpoint of the interval where the weight function $x^{-1/2}$ becomes singular. On the left $N = 4$ and $\tilde{N} = 2$; on the right $N = 5$ and $\tilde{N} = 1$. Note how the weighted wavelets take on smaller values at zero in order to adapt to the weight function whose value tends to infinity.*

be performed recursively from the finest level on up by using the refinement relationship for the scaling function. Without going into any further detail we point out that moment calculations and the solution of the linear system to find $p(x)$ can be numerically delicate. The stability depends on which polynomial basis is used. For example, we found the linear systems that result when expressing everything with respect to global monomial moments so ill-conditioned as to be unsolvable even in double precision. The solution lies in using a local polynomial, i.e., a basis which changes for each interval. A better choice might be a basis of local orthogonal polynomials.

In our experiments we used the weight function $x^{-1/2}$ which is singular at the left interval boundary. To compute the moments we used local monomials, resulting in integrals for which analytic expressions are available.

Figure 2.12 shows some of the resulting wavelets. In both cases we show the left most wavelet, which is most impacted by the weight function. Weighted and unweighted wavelets further to the right become ever more similar. Part of the reason why they look similar is the normalization. For example, both weighted and unweighted scaling functions have to satisfy $\sum_k \varphi_{j,k} = 1$. The images show wavelets with $N = 4$ (interpolating) and $\tilde{N} = 2$ vanishing moments on the left and wavelets $N = 5$ (average-interpolation) and $\tilde{N} = 1$ vanishing moment on the right. In both cases the weighted wavelet is shown with a solid line and the unweighted case with a dotted line.

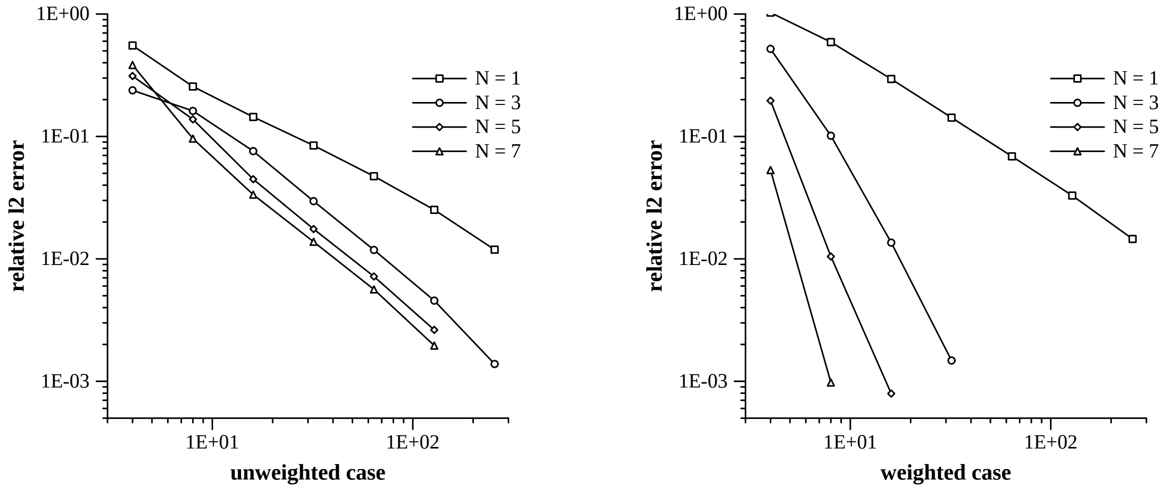

Figure 2.13: *Comparison of approximation error when expanding the function* $\sin(4\pi x^{1/2})$ *over* $[0, 1/2]$ *using wavelets constructed with respect to an unweighted inner product (left) and a weighted inner product with weight* $x^{-1/2}$ *(right). Here* $N = 1$, 3, 5, *and* 7.

The weighted and unweighted wavelets are only slightly different in shape. However, when applied to the expansion of some function they can make a dramatic difference. As an example we applied both types of wavelets to the function $f(x) = \sin(4\pi x^{1/2})$, which has a divergent derivative at zero. With unweighted wavelets the convergence will be slow close to the singularity, typically $O(h)$ with $h = 2^{-j}$ independent of N. In other words, there is no gain in using higher order wavelets. However, if we build weighted wavelets for which the weight function times f is an analytic function, we can expect $O(h^N)$ behavior everywhere again. For our example we can take $w(x) = x^{-1/2}$. This way the weighted wavelets are adapted to the singularity of the function f. Figure 2.13 shows the error in the resulting expansions with order $N = 1$, 3, 5, and 7 and dual order $\tilde{N} = 1$. For unweighted wavelets higher order constructions only get better by a constant factor, while the weighted wavelets show higher order convergence when going to higher order wavelets.

2.11 Warning

Like every "do it yourself at home" product this one comes with a warning. Most of the techniques we presented here are straightforward to implement and before you know it you will be generating wavelets yourself. However, we did not discuss most of the deeper underlying mathematical properties which assure that everything works like we expect it to. These address issues such as: What are the conditions on the subdivision scheme so that it generates smooth functions? or: Do the resulting scaling functions and wavelets generate a stable, i.e., Riesz basis? These questions are not easily answered and require some heavy mathematics. One of the fundamental questions is how properties, such as convergence of the subdivision algorithm, Riesz bounds, and smoothness, can be related back to properties of the filter sequences. This is a very hard question and at this moment no general answer is available to our knowledge.

We restrict ourselves here to a short description of the extent to which these questions have been answered. In the classical case, i.e., regular samples and no weight function, everything essentially works. The regularity of the basis functions varies linearly with N. In the case of the interval, regular samples, and no weight function, again the same results hold. This is because the boundary basis functions are finite linear combinations of the ones from the real line. In the case of regular samples with a weight function, it can be shown that with some minimal conditions on the weight function, the basis functions have the same regularity as in the unweighted case. In the case of irregular samples, little is known at this moment. Everything essentially depends on how irregular the samples are. It might be possible to obtain results under the conditions that the irregular samples are not too far from the regular samples, but this has to be studied in detail in the future.

Recent results concerning general multiscale transforms and their stability were obtained by Wolfgang Dahmen and his collaborators. They have been working (independently from [26, 27]) on a scheme which is very similar to the lifting scheme [2, 8]. In particular, Dahmen shows in [7] which properties in addition to invertibility of the transform are needed to assure stable bases. Whether this result can be applied to the bases constructed here needs to be studied in the future.

2.12 Outlook

So far we have only discussed the construction of second generation wavelets on the real line or the interval. Most of the techniques presented here such as polynomial subdivision and lifting extend easily to much more general sets. In particular domains in $\mathbf{R}^n$, curves, surfaces, and manifolds.

One example is the construction of wavelets on the sphere [22]. There we use the lifting scheme to construct locally supported, biorthogonal spherical wavelets and their associated fast transforms. The construction starts from a recursive triangulation of the sphere and is parameterization independent. Since the construction does not rely on any specific properties of the sphere it can be generalized to other surfaces. The only question which needs to be addressed is what the right replacement for polynomials is. Polynomials restricted to a sphere are still a natural choice because of the connection with spherical harmonics, but on a general surface this is no longer the case.

Bibliography

[1] L. Andersson, N. Hall, B. Jawerth, and G. Peters. Wavelets on closed subsets of the real line. In [23], pages 1–61.

[2] J. M. Carnicer, W. Dahmen, and J. M. Peña. Local decompositions of refinable spaces. Technical report, Insitut für Geometrie und angewandete Mathematik, RWTH Aachen, 1994.

[3] C. Chui and E. Quak. Wavelets on a bounded interval. In D. Braess and L. L. Schumaker, editors, *Numerical Methods of Approximation Theory*, pages 1–24. Birkhäuser-Verlag, Basel, 1992.

[4] C. K. Chui, L. Montefusco, and L. Puccio, editors. *Conference on Wavelets: Theory, Algorithms, and Applications.* Academic Press, San Diego, CA, 1994.

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

[6] A. Cohen, I. Daubechies, and P. Vial. Multiresolution analysis, wavelets and fast algorithms on an interval. *Appl. Comput. Harmon. Anal.*, 1(1):54–81, 1993.

[7] W. Dahmen. Stability of multiscale transformations. Technical report, Institut für Geometrie und angewandete Mathematik, RWTH Aachen, 1994.

[8] W. Dahmen, S. Prössdorf, and R. Schneider. Multiscale methods for pseudo-differential equations on smooth manifolds. In [4], pages 385–424.

[9] 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.

[10] G. Deslauriers and S. Dubuc. Interpolation dyadique. In *Fractals, dimensions non entières et applications*, pages 44–55. Masson, Paris, 1987.

[11] G. Deslauriers and S. Dubuc. Symmetric iterative interpolation processes. *Constr. Approx.*, 5(1):49–68, 1989.

[12] D. L. Donoho. Smooth wavelet decompositions with blocky coefficient kernels. In [23], pages 259–308.

[13] D. L. Donoho. Interpolating wavelet transforms. Preprint, Department of Statistics, Stanford University, 1992.

[14] D. L. Donoho and I. M. Johnstone. Ideal spatial adaptation via wavelet shrinkage. *Biometrika*, to appear, 1994.

[15] D. L. Donoho and I. M. Johnstone. Adapting to unknown smoothness via wavelet shrinkage. 1995.

[16] N. Dyn, J. A. Gregory, and D. Levin. A four-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, (4):257–268, 1987.

[17] M. Girardi and W. Sweldens. A new class of unbalanced Haar wavelets that form an unconditional basis for l_p on general measure spaces. Technical Report 1995:2, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina, 1995*.

[18] M. Lounsbery, T. D. DeRose, and J. Warren. Multiresolution surfaces of arbitrary topological type. Department of Computer Science and Engineering 93-10-05, University of Washington, October 1993. Updated version available as 93-10-05b, January, 1994.

[19] S. G. Mallat. Multiresolution approximations and wavelet orthonormal bases of $L^2(\mathbf{R})$. *Trans. Amer. Math. Soc.*, 315(1):69–87, 1989.

[20] Y. Meyer. *Ondelettes et Opérateurs,* I: *Ondelettes,* II: *Opérateurs de Calderón-Zygmund,* III: (with R. Coifman)*, Opérateurs multilinéaires.* Hermann, Paris, 1990. English translation of first volume, *Wavelets and Operators*, is published by Cambridge University Press, 1993.

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

[22] P. Schröder and W. Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. *Computer Graphics Proceedings, (SIGGRAPH 95)*, pages 161–172, 1995.

[23] L. L. Schumaker and G. Webb, editors. *Recent Advances in Wavelet Analysis.* Academic Press, New York, 1993.

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

[25] W. Sweldens. *Construction and Applications of Wavelets in Numerical Analysis.* PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 1994.

[26] W. Sweldens. The lifting scheme: A construction of second generation wavelets. Technical Report 1995:6, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina, 1995.

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

[28] M. Vetterli and C. Herley. Wavelets and filter banks: theory and design. *IEEE Trans. Acoust. Speech Signal Process.*, 40(9):2207–2232, 1992.

Afternoon Section: Applications