Lossless Compression of 3D MRI and CT Data

Andreas Klappenecker*, Frank U. May†, and Thomas Beth

Institute for Algorithms and Cognitive Systems
University of Karlsruhe
D-76128 Karlsruhe, Germany
wavelet@ira.uka.de

ABSTRACT

We propose a conceptually simple method for lossless compression of medical image and volume data. The method can be divided into three steps: the input data is decomposed into several subbands with the help of nonlinear lifting filters, the resulting subbands are block-sorted according to a method suggested by Burrows and Wheeler, and the redundancy is removed with the help of an adaptive arithmetic coder. Moreover, we suggest a new method to implement (non-linear) lifting filters. We describe these filters with the help of a small filter description language, which is compiled into a shared object file and dynamically loaded at run time. The source code of the program is freely available for testing purposes.

Keywords: Wavelets, lifting filters, filter description language, lossless compression, medical image data.

1. INTRODUCTION

Modern medical diagnostics are often based on X-ray computerized tomography (CT) and magnetic resonance imaging (MRI) techniques. The raw three-dimensional scan data delivered by such imaging devices occupies several mega bytes of disc space per data set. Some diagnostic methods may even require several scans per patient, for example in functional computer tomographic examination of whiplash injuries, cf. [1]. Consequently there is a natural demand for highly efficient lossless compression methods.

Recently, several lossless image compression methods were proposed that are based on subband decomposition, e.g. [2, 3, 4]. These methods combine the compression performance of state of the art lossless compression methods with the advantage of an embedded bitstream, allowing a progressive decoding of the image.

We propose a conceptually simple method for lossless compression of three-dimensional MRI and CT data. We decided to use separable filter banks with lifting filters to reduce the entropy of the signal. Our methods rely on modular arithmetic or nonlinear operations to reduce the range of the resulting coefficients. The subbands are then transformed with the Burrows-Wheeler transform[5] and subsequently encoded by an adaptive arithmetic encoder.

Depending on diagnostic purposes, a three-dimensional MRI or CT data set consists of twenty up to more than 100 two-dimensional slices, and the slice distance may vary between 0.6 and 7 mm. Our experiments showed that even for data sets with a small number of two-dimensional slices and a high slice distance there is a significant gain in compression ratio by compressing the 3D data comparted to compressing the two-dimensional slices separately.

Beside the lossless compression methods we show a new way to implement lifting filters in compiled languages. The traditional way of specifying filters in terms of their coefficients is not sufficient to describe nonlinear lifting steps. In contrast, the lifting filters are described in a filter description language (FDL) and compiled into a shared object file, which is dynamically loaded at run time. The compilation step results in significantly higher performance compared to an FDL interpreter.

*This work was supported by the DFG through SFB 414, projects Q1, Q6.
†This work was partially supported by DFG grant Be 887/6-4.

2. FILTER BANKS

We want to decompose CT or MRI image or volume data with a multirate filter bank. Since our aim is lossless compression, this filter bank should allow for a perfect reconstruction of the signal. Moreover, the implementation of the filter bank should avoid a coefficient swell, that is, each subband coefficient should be representable by a decent number of bits, even after some decomposition steps. We are interested in this property since it is difficult to build a coder that exploits all redundancies in a signal.

In this section we recall the basic idea of one-dimensional two-channel filter banks and their lifting filter realization. We will tensor these filter banks to obtain the corresponding filter banks for higher-dimensional signals.

The pixel values of CT or MRI volume data are typically represented by 12-16 bits. Thus, we may view these values as integers in the range from 0 up to 65535. In each row, column, or along the third axis there are only a finite number of non-zero coefficients. A single row may thus be regarded as an element of the group ring $\mathbf{Z}[\mathbf{Z}/N\mathbf{Z}]$. One possibility would be to use a tensor product of cyclic filter banks[6–8] with two channels and integer valued filters to decompose such a signal. The data is processed by two cyclic convolutions, followed by downsampling, and this process is repeated along each axis. Therefore, the signal is decomposed into eight subbands.

We briefly recall how such a cyclic filter bank works. The group ring $\mathbf{Z}[\mathbf{Z}/N\mathbf{Z}]$ is isomorphic to the "truncated" polynomial ring $\mathbf{Z}[\,z\,]/\langle 1-z^N\rangle$. The input signal $s(z)\in\mathbf{Z}[z]/\langle 1-z^N\rangle$ is processed by two cyclic convolutions with the filters $\widetilde{\alpha}(z)$ and $\widetilde{\beta}(z)$. These convolutions are simply given by the multiplication in $\mathbf{Z}[\,z\,]/\langle 1-z^N\rangle$, that is, by polynomial multiplication modulo $1-z^N$. After these convolution operations the sampling rate is reduced by half. Assuming that N is given by an even positive integer, we obtain in this step two signals $d_\alpha(z) := [\downarrow 2]\,\widetilde{\alpha}(z)s(z)$ and $d_\beta(z) := [\downarrow 2]\,\widetilde{\beta}(z)s(z)$, where the downsampling operation $[\downarrow 2]$ is defined as usual by mapping the signal $s(z)$ to the even part $s_e(z)$ of the polyphase decomposition $s(z) = s_e(z^2) + z s_o(z^2)$, that is, the operation $[\downarrow 2]$ is given by

$$[\downarrow 2] : \begin{cases} \mathbf{Z}[\,z\,]/\langle 1-z^N\rangle & \longrightarrow & \mathbf{Z}[\,z\,]/\langle 1-z^{N/2}\rangle, \\ s(z) & \longmapsto & s_e(z). \end{cases}$$

For reconstruction, the "sampling rate" of the signals $d_\alpha(z)$ and $d_\beta(z)$ is doubled with the help of the upsampling operation $[\uparrow 2]$, which is defined by

$$[\uparrow 2] : \begin{cases} \mathbf{Z}[\,z\,]/\langle 1-z^{N/2}\rangle & \longrightarrow & \mathbf{Z}[\,z\,]/\langle 1-z^{N}\rangle, \\ s(z) & \longmapsto & s(z^2). \end{cases}$$

After the upsampling, the signals $d_\alpha(z^2)$ and $d_\beta(z^2)$ are multiplied with $\alpha(z)$ and $\beta(z)$ respectively and added (again all operations are taken modulo $1-z^N$). This yields $\hat{s}(z) = d_\alpha(z^2)\alpha(z) + d_\beta(z^2)\beta(z) \bmod 1-z^N$.

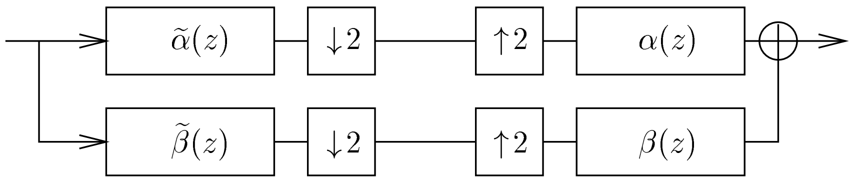

Figure 1. Cyclic filter bank for signals in $\mathbf{Z}[\,z\,]/\langle 1-z^N\rangle$. All convolutions (multiplications) are taken modulo $1-z^N$.

The reconstructed signal $\hat{s}(z)$ will coincide with $s(z)$ for all signals $s(z)\in\mathbf{Z}[\,z\,]/\langle 1-z^N\rangle$ provided the filters $\widetilde{\alpha},\widetilde{\beta}$ and α,β are properly chosen. In principle, we could use such a filter bank for our compression application. However, this approach has two major drawbacks. Although the number of pixels is not expanded due to the cyclic convolution, we have in general annoying border effects. Moreover, the overall size (measured in bits) will increase dramatically, since the output of the convolutions has a significantly wider range of integer values.

We can avoid these drawbacks with the help of the lifting scheme.[2,9–12] The reader should consult these references for a thorough introduction to the philosophy of this method. We will restrict ourselves to those aspects relevant to our application.

As a motivation we will briefly discuss polyphase matrices and their decomposition in ladder structures (for a more detailed discussion of this topic we refer to [13] in these proceedings). Recall that the output $(d_\alpha(z), d_\beta(z))^t$ of the analysis filter bank can be obtained by multiplying the polyphase vector $(s_e(z), s_o(z))^t$ of the input signal $s(z) = s_e(z^2) + z s_o(z^2)$ with the polyphase matrix

$$H_p(z) := \begin{pmatrix} \widetilde{\alpha}_e(z) & \widetilde{\alpha}_o(z) \\ \widetilde{\beta}_e(z) & \widetilde{\beta}_o(z) \end{pmatrix},$$

where the matrix entries are defined by $\widetilde{\alpha}(z) = \widetilde{\alpha}_e(z^2) + z^{-1}\widetilde{\alpha}_o(z^2)$ and $\widetilde{\beta}(z) = \widetilde{\beta}_e(z^2) + z^{-1}\widetilde{\beta}_o(z^2)$. The left-hand side of Figure 2 shows the corresponding implementation.

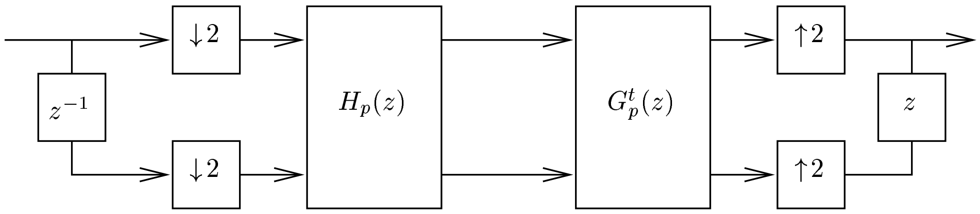

Figure 2. Polyphase implementation of a cyclic filter bank.

The polyphase matrix $H_p(z)$ of a perfect reconstructing filter bank is an element of the group $\mathrm{GL}_2(\mathbf{Z}[\mathbf{Z}/N\mathbf{Z}])$.[13] This group has the special property that it can be generated by elementary transvections, i. e., matrices that differ from the identity matrix by one off-diagonal entry, and invertible diagonal matrices (this non-trivial fact was proved in [14]). Ignoring diagonal matrices, we can write the analysis part as a product of transvections, giving a ladder step implementation. For example, if the polyphase matrix $H_p(z)$ is given by the product

$$H_p(z) = \begin{pmatrix} 1 & 0 \\ b(z) & 1 \end{pmatrix} \begin{pmatrix} 1 & a(z) \\ 0 & 1 \end{pmatrix},$$

then we obtain a ladder step implementation of the analysis filter bank as shown in Figure 3. The advantage of these ladder steps is that they can be easily inverted. Namely, we have to apply the same elementary transvections but in reverse order and with the sign of the off-diagonal entries inverted.

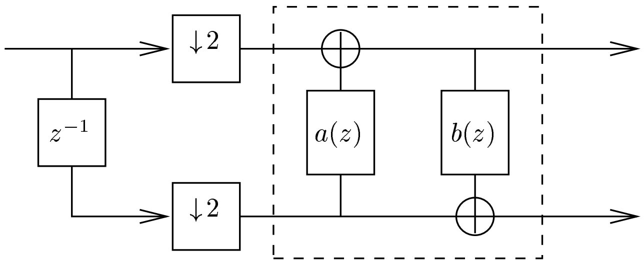

Figure 3. Ladder step implementation.

Wim Sweldens observed that the same simple inversion is possible, if we replace the convolution operators ($x \mapsto a(z)x$ and $x \mapsto b(z)x$ in our preceding example) by more general operations. To see this, observe that the effect of the elementary transvections on a polyphase vector can be described by the mappings

$$(\, s_e(z), s_o(z)\,)^t \longmapsto (\, s_e(z) + a(z)s_o(z), s_o(z)\,)^t \qquad \text{and} \qquad (\, s_e(z), s_o(z)\,)^t \longmapsto (\, s_e(z), s_o(z) + b(z)s_e(z)\,)^t.$$

Assume that we are given operators P and Q that map $\mathbf{Z}[\,z\,]/\langle 1 - z^{N/2}\rangle$ into itself. Then it is immediately clear that the mappings

$$(\, s_e(z), s_o(z)\,)^t \longmapsto (\, s_e(z) + P(s_o(z))\,, s_o(z)\,)^t \qquad \text{and} \qquad (\, s_e(z), s_o(z)\,)^t \longmapsto (\, s_e(z), s_o(z) + Q(s_e(z))\,)^t$$

are invertible as well. We refer to these operations as lifting steps. Note that these lifting steps are a natural generalization of ladder steps.

3. A FILTER DESCRIPTION LANGUAGE

Although the lifting scheme is in principle as simple as a ladder implementation, there arise some practical problems. Whereas a ladder step is already determined by the indices of the off-diagonal entry and the coefficients of the ladder step filter, it is not clear how to specify a general lifting step. We describe in this section a small language for this purpose.

Assume that the input signal is given as an element of the group ring $\mathbf{Z}[\mathbf{Z}/2N\mathbf{Z}]$. In the first step, this signal is split into the even indexed subsequence and the odd indexed subsequence. These subsequences can be viewed as two integer arrays `e` and `o`. We can refer to the elements of these sequences by `e[k]` and `o[k]`. In other words, if the input sequence is given by the samples s_ℓ, with $0 \leq \ell < 2N$, then `e[k]` refers to the element s_{2k} and `o[k]` refers to the element s_{2k+1}.

A lifting step simply takes the elements of one array as input, operates on this data, and adds (or subtracts) the result from the other array. We allow addition `+`, subtraction `-`, multiplication `*`, integer division `/`, left shift `<<`, and right shift `>>` as binary operators. An integer expression can be added to an array element with the operator `+=` and subtracted from an array element with the operator `-=`. For example, the expression `e[k] += 3*o[k]` means that the value of `o[k]` is multiplied by three, added to `e[k]`, and the result is stored in `e[k]`. In addition to the integer operations, we allow floating point operations in intermediate calculations of a lifting step. However, one should note that the assignment operators `+=` and `-=` require an integer expression on the right hand side. The ceiling, floor, and round functions `CEIL`, `FLOOR`, and `ROUND` can be used to convert floating point expressions back to integers. The syntax of our language is summarized in Figure 4.

We explain the main concepts of our language with the help of a few examples. The description of a lifting scheme is always of the form `lifting(` *body* `)`, where *body* consists of several loop statements. For example, a lifting scheme with two lifting steps is shown in the following example:

```
lifting(
   [0;N-1] o[k] -= e[k]; e[k] += o[k]/2;
)
```

The loop index is always given by the special variable `k`. The loop ranges from the index `0` to `N-1`, where `N` denotes the length of the arrays `o` and `e`. In the first lifting step, the array elements `e[k]` are subtracted from `o[k]` and the result is stored in `o[k]`. The second statement makes an integer division by two of `o[k]`, adds the result to `e[k]` and assigns the result of this addition to `e[k]`. In the next example we have replaced the integer division by a floating point division. Since assignments require integer expressions, we need to convert the result of the floating point division to an integer e. g. with the help of the floor function:

```
lifting(
   [0;N-1] o[k] -= e[k]; e[k] += FLOOR(o[k]/2.0);
)
```

This example is an integer version of the Haar transform, which is known as the S-transform, see for example [12].

The next example gives another example of a lifting scheme with two lifting steps. It shows how border effects can be treated:

```
lifting (
   [0]          o[k] -= ((e[k]+e[k+1])/2);      e[k] += (o[k]/2);
   [1; N-2]     o[k] -= ((e[k]+e[k+1])/2);      e[k] += ((o[k-1]+o[k])/4);
   [N-1]        o[k] -= e[k];                   e[k] += ((o[k-1]+o[k])/4);
)
```

In the first lifting step the value of `o[k]` is predicted by `(e[k]+e[k+1])/2`. We have to take care of border effects in this example. We can calculate `o[k] -= (e[k]+e[k+1])/2` in the range `[0;N-1]`. For `o[N-1]` we need a different treatment, for example, we may try to predict this value from `e[N-1]`. In this example we have divided the signal into three regions (left, inner, and right), which is expressed by the first column of loop ranges. The first statement of each line is computed for all `k` in the range. In other words, the first statement in each line refers to the first lifting step. Therefore, we require that all these statements refer to the same target array (in our example the array `o`). The second lifting step has the target array `e`. This array is updated by the second statement in each range.

```
lifting( body )

body     ::= loop
           | body loop

loop     ::= range stmnts

stmnts   ::= stmnt
           | stmnts stmnt

range    ::= [ bound ] | [ bound ; bound ]

bound    ::= number | N | bound + bound | bound - bound

stmnt    ::= lval += iexpr ;
           | lval -= iexpr ;
           | lval +%( modulus )= iexpr ;
           | lval -%( modulus )= iexpr ;

modulus  ::= number | nvals | modulus + modulus | modulus - modulus

lval     ::= e[k] | o[k]

iexpr    ::= e[ index ]
           | o[ index ]
           | number
           | ( iexpr )
           | - iexpr
           | iexpr + iexpr
           | iexpr - iexpr
           | iexpr * iexpr
           | iexpr / iexpr
           | iexpr >> iexpr
           | iexpr << iexpr
           | CEIL ( fexpr )
           | FLOOR ( fexpr )
           | ROUND ( fexpr )
           | MOD ( iexpr , modulus )

fexpr    ::= iexpr
           | number . number
           | ( fexpr )
           | - fexpr
           | fexpr + fexpr
           | fexpr - fexpr
           | fexpr * fexpr
           | fexpr / fexpr

index    ::= k | k + number | k - number

number   ::= [0-9]+
```

Figure 4. The syntax of the filter description language.

Before we introduce more operations it is instructive to generalize the lifting scheme. We used two integer arrays and updated each array with the operations `-=` and `+=`. More generally, we can replace the (additive) group of integers by an arbitrary group. All operations can be done in the same way. As a particular example we can take the residue class ring $\mathbf{Z}/n\mathbf{Z}$ with addition taken modulo n. We know that the pixel values of an image are in a certain range, for example in [0..65535]. Therefore, we can view these values as elements of $\mathbf{Z}/n\mathbf{Z}$, provided that $n \geq 2^{16}$. The analogues of the assignment operators `-=` and `+=` in the group $\mathbf{Z}/n\mathbf{Z}$ are denoted by `+%(` n `)=` and `-%(` n `)=` in our small language.

The next example is a modular version of the previous example. The special variable `nvals` contains the value 2^n, if the input image data is of n bit depth.

```
lifting (
  [0]      o[k] -%(nvals)= ((e[k]+e[k+1])/2);  e[k] +%(nvals)= (o[k]/2);
  [1; N-2] o[k] -%(nvals)= ((e[k]+e[k+1])/2);  e[k] +%(nvals)= ((o[k-1]+o[k])/4);
  [N-1]    o[k] -%(nvals)= e[k];               e[k] +%(nvals)= ((o[k-1]+o[k])/4);
)
```

The interpretation of most constructs of our language should now be clear from the previous examples. As a general rule, we note that the precedence and associativity of the operators is borrowed from the corresponding C operators. Although it is not obvious from the syntax of our language, it should be noted that if an array element `e[k]` (resp. `o[k]`) occurs as an *lval* in a statement, then the *iexpr* of this statement should not contain any reference to an element of the array `e` (resp. `o`). Moreover, if the ith statement in a loop has `e[k]` (resp. `o[k]`) as an *lval*, then it is required to be the *lval* in the ith statement of all other loops. These rules reflect the fact the the ith statement of a loop is part of the description of the ith lifting step.

The description of the lifting filters is translated in our implementation to C code for one-, two-, and three-dimensional data (realizing the non-linear analogues of tensored wavelet filters). Our compression program `lwc` reads such a description, generates the necessary C code, calls the C compiler to generate a shared object file, and dynamically links this object file. This way we obtain a fairly efficient implementation without requiring the user to write C code.

4. BURROWS-WHEELER TRANSFORM

The decomposition of the volume data into subbands reduces the entropy but does not decorrelate completely. We perform a second transform on the subbands for this reason, which is know as the Burrows-Wheeler transform.[5,15]

The Burrows-Wheeler transfora is best understood with the help of an example. Suppose we want to transform the string `abaada`.

$$M := \begin{pmatrix} \texttt{abaada} \\ \texttt{baadaa} \\ \texttt{aadaab} \\ \texttt{adaaba} \\ \texttt{daabaa} \\ \texttt{aabaad} \end{pmatrix} \qquad M' := \begin{pmatrix} \texttt{aabaad} \\ \texttt{abaada} \\ \texttt{baadaa} \\ \texttt{aadaab} \\ \texttt{adaaba} \\ \texttt{daabaa} \end{pmatrix}.$$

We have $M[\tau(i), j] = M'[i, j]$, where τ is given by the permutation $i \mapsto i + 1 \bmod N$

$$N := \begin{pmatrix} \texttt{aabaad} \\ \texttt{aadaab} \\ \texttt{abaada} \\ \texttt{adaaba} \\ \texttt{baadaa} \\ \texttt{daabaa} \end{pmatrix} \qquad N' := \begin{pmatrix} \texttt{daabaa} \\ \texttt{baadaa} \\ \texttt{aabaad} \\ \texttt{aadaab} \\ \texttt{abaada} \\ \texttt{adaaba} \end{pmatrix}$$

$$\sigma = (0\ 2\ 1\ 4\ 5), \qquad \tau = (0\ 1\ 2\ 3\ 4\ 5)$$

$$\beta = (0\ 2\ 4\ 1\ 3\ 5)$$
$$\sigma\tau\sigma^{-1} = (0\ 2\ 1\ 4\ 5)(0\ 1\ 2\ 3\ 4\ 5)(5\ 4\ 1\ 2\ 0) = (0\ 2\ 4\ 1\ 3\ 5) = \beta$$

Figure 5. A slice of an MRI scan. The image shows the knee of our colleague Detlef Zerfowski (shortly after unhealthy marathon training).

Figure 6. Burrows-Wheeler transform of the image shown in Figure 5. (Some people think that the same effect can achieved after infinitely many hours of marathon training...)

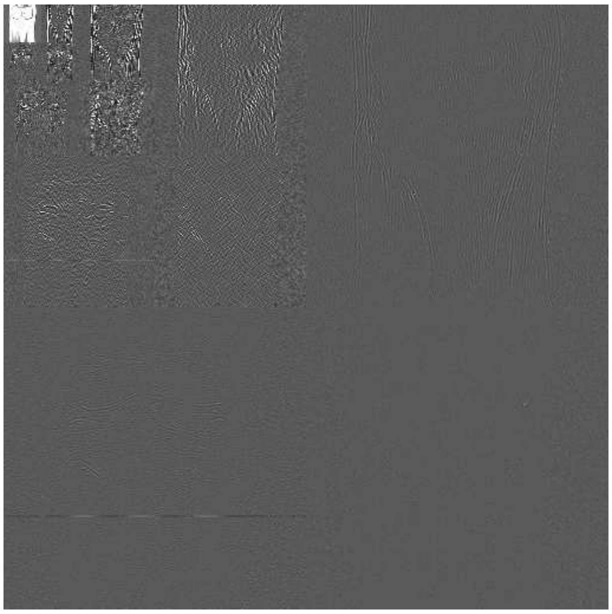

Figure 7. Nonlinear analogue of the wavelet transform. The figure shows a decomposition of the image shown in Figure 5.

5. CONCLUSION

APPENDIX A. OPEN SOFTWARE

A prototype implementation of the method described in this paper is freely available at

`http://iaks-www.ira.uka.de/home/klappi/lossless.html`

The source code of our compression program `lwc` is distributed under the GNU general public license.

REFERENCES

1. J. Dvorak and J. Hayek, "Diagnostik der Instabilität der oberen Halswirbelsäule mittels funktioneller Computertomographie," *Fortschr. Roentgenstr.* **145**(5), pp. 582–585, 1986.
2. A. R. Calderbank, I. Daubechies, W. Sweldens, and B. L. Yeo, "Wavelet transforms that map integer to integers." To appear in ACHA, 1996.
3. A. Klappenecker, A. Nückel, and F. U. May, "Lossless image compression using wavelets over finite rings and related architectures," in *Wavelet Applications in Signal and Image Processing V*, A. Aldroubi, A. F. Laine, and M. A. Unser, eds., vol. 3169, pp. 139–147, SPIE, 1997.
4. N. Memon, X. Wu, and B.-L. Yeo, "Entropy coding techniques for lossless image compression with reversible integer wavelet transforms." IBM Research Report RC 21010, October 1997.
5. M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm." Digital Systems Research Center Report 124, 1994.
6. A. S. Bopardikar, M. R. Raghuveer, and B. S. Adiga, "PRCC filter banks: theory, implementation, and application," in *Wavelet Applications in Signal and Image Processing V*, A. Aldroubi, A. F. Laine, and M. A. Unser, eds., vol. 3169, pp. 410–421, SPIE, 1997.
7. G. Caire, R. L. Grossman, and H. V. Poor, "Wavelet transforms associated with finite cyclic groups," *IEEE Trans. on Information Theory* **39**(4), pp. 1157–1166, 1993.
8. P. P. Vaidyanathan and A. Kirac, "Theory of cyclic filter banks," in *IEEE Int. Conf. on Accoustics, Speech and Signal Processing*, pp. 2449–2452, IEEE, 1997.
9. I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *J. Fourier Anal. Appl.* **4**(3), pp. 245–267, 1998.
10. G. Fernández, S. Periaswamy, and W. Sweldens, "LIFTPACK: A software package for wavelet transforms using lifting," in *Wavelet Applications in Signal and Image Processing IV*, M. Unser, A. Aldroubi, and A. F. Laine, eds., pp. 396–408, Proc. SPIE 2825, 1996.
11. W. Sweldens, "The lifting scheme: a custom-design construction of biorthogonal wavelets," *Appl. Comp. Harmon. Anal.* **3**(2), pp. 186–200, 1996.
12. W. Sweldens and P. Schröder, "Building your own wavelets at home." In "Wavelets in Computer Graphics", ACM SIGGRAPH Course Notes, 1996.
13. A. Klappenecker and M. Holschneider, "A unified view on filter banks." These proceedings, 1998.
14. K. Dennis, B. Magurn, and L. Vaserstein, "Generalized euclidean group rings," *J. reine angew. Math.* **351**, pp. 113–128, 1984.
15. M. Nelson, "Data compression with the burrows-wheeler transform," *Dr. Dobb's Journal* , September 1996.