10. Fast Algorithms of Hypercomplex Fourier Transforms*

Michael Felsberg[1], Thomas Bülow[1], Gerald Sommer[1], and Vladimir M. Chernov[2]

[1] Institute of Computer Science and Applied Mathematics, Christian-Albrechts-University of Kiel
[2] Image Processing System Institute, Russian Academy of Sciences, Samara

10.1 Introduction

In this chapter we consider the computational aspect of the quaternionic Fourier transform (QFT), of the Clifford Fourier transform (CFT), and of the commutative hypercomplex Fourier transform (HFT). We can cover all these transforms with the term *hypercomplex Fourier transforms*, since all mentioned algebras are hypercomplex algebras (see Cha. 9). In order to have a numerical way to evaluate these transforms, we introduce the corresponding discrete transforms by sampling the continuous transforms. Furthermore, we prove the inverse transforms.

The simplest way to create fast algorithms for the discrete transforms (of real n-dimensional signals) is to separate the transforms into $2^n - 1$ transforms so that each transform only effects one spatial coordinate. Therefore, the asymptotic complexity of fast algorithms for n-dimensional transforms

* This work has been supported by German National Merit Foundation and by DFG Grants So-320-2-1, So-320-2-2, and Graduiertenkolleg No. 357.

should not exceed $2^n - 1$ times the complexity of $(2^n - 1)N^n$ FFTs[1], i.e. the complexity should be of order $\mathcal{O}(N^n \log n)$ (or less).

In order to obtain little computational complexities, we formulate several approaches for fast algorithms. We make use of some algebraic properties of hypercomplex algebras in order to optimize the transforms. Some of the algorithms are based on standard algorithms, so that for the implementation there is nearly no additional coding; the standard implementations can be re-used.

All the time and memory complexities of the presented algorithms are estimated, illustrated, and systematically compared. Hence, it should be easy for the reader to choose the algorithm which suits best for his purposes. This decision does not only depend on the purpose but also on the computer environment. We try to stay abreast of this fact by considering cache sizes and memory sizes of today's computers knowing the fact that the usage of swap space (i.e. to swap out data on a hard disc) drives any algorithmetic optimization insane.

10.2 Discrete Quaternionic Fourier Transform and Fast Quaternionic Fourier Transform

In this section we define the discrete quaternionic Fourier transform (DQFT) in a similar way as the discrete complex Fourier transform is defined, i.e. the continuous transform is sampled. We state and prove the inverse discrete QFT, a fast algorithm for the QFT (FQFT) and a fast algorithm for the inverse QFT. Further optimizations of the fast algorithms are presented (and proved) and their complexities are considered in detail.

10.2.1 Derivation of DQFT and FQFT

Starting point for a fast algorithm of the QFT is the discrete quaternionic Fourier transform, of course. It is defined analogously to the discrete Fourier transform by sampling the continuous transform (8.3.4). Formally, the continuous, infinite signal $f(x, y)$ which must be of limited bandwidth (or convolved with a low-pass filter) is convolved with the Shah function (infinite sum of equi-distant Dirac impulses). Afterwards, the new (periodic) signal is sampled so that $f_{m,n} = f(me_x, ne_y)$ where e_x and e_y must be integer divisors of the periods with respect to x and y. Formally, the last step is a pointwise multiplication with a Shah function.

We obtain the discrete transform by two steps. Firstly, we only consider the periodicity. As a result we evaluate the integral of (8.3.4) only in one period (similar to Fourier series). Secondly, we multiply the integral with the Shah function, so the integral changes to a sum:

[1] Note that N indicates the extension of the signal in only one dimension. The total size of the signal is therefore N^n.

Definition 10.2.1 (Discrete QFT). *Let f be a discrete two-dimensional signal of finite size $M \times N$. The discrete quaternionic Fourier transform (DQFT), denoted as $\mathcal{F}_q^D\{f\} = F^q$, is defined by*

$$F^q_{u,v} := \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} e^{-i2\pi uxM^{-1}} f_{x,y} e^{-j2\pi vyN^{-1}}. \tag{10.1}$$

The discrete inverse transform can be derived from the continuous one, too. Due to the fact that a periodic signal in the spatial domain corresponds to a discrete spectrum (Fourier series) and a discrete (infinitely long) signal corresponds to a periodic spectrum, we can simply exchange spatial and frequency domain. This yields the same transform, except for the sign of the exponential term and except for a normalizing factor which must be multiplied in this transform as in the one-dimensional case:

Theorem 10.2.1 (Inverse DQFT). *The inverse discrete quaternionic Fourier transform $\mathcal{F}_q^{D^{-1}}\{F^q\} = f$ reads*

$$f_{x,y} = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} e^{i2\pi uxM^{-1}} F^q_{u,v} e^{j2\pi vyN^{-1}}. \tag{10.2}$$

Proof. Due to the fact that we call (10.2) the inverse transform of (10.1), we must prove that the concatenation of both transforms yields the identity.

Firstly, we define abbreviations for the modulation terms:

$$w_i = e^{i2\pi M^{-1}} \tag{10.3a}$$

$$w_j = e^{j2\pi N^{-1}} \tag{10.3b}$$

Applying formula (10.2) to (10.1) yields:

$$\begin{aligned}
&\mathcal{F}_q^{D^{-1}} \left\{ \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} w_i^{-ux} f_{x,y} w_j^{-vy} \right\} \\
&= \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} w_i^{ux'} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} w_i^{-ux} f_{x,y} w_j^{-vy} w_j^{vy'} \\
&= \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} w_i^{u(x'-x)} f_{x,y} w_j^{v(y'-y)} \\
&= \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} M\delta_{x'-x} f_{x,y} N\delta_{y'-y} \\
&= f_{x',y'}
\end{aligned}$$

So the concatenation of the transforms (10.2) and (10.1) yields the identity and we can call them inverse to each other. Note that the other concatenation need not be proved because the Fourier transform is a 1-1 mapping (it is just a change to another basis of same dimension). □

Now, we are able to calculate the spectrum of a finite, discrete signal. If we use formula (10.1) to implement an algorithm for the DQFT, we obtain a computational complexity of cM^2N^2 (c being a constant). This complexity is quite high. Hence we try to reduce it as it has been done by Cooley and Tuckey when they developed their fast Fourier transform (FFT) algorithm. The idea they used is the *decimation of time* method.

The FFT algorithm has originally been designed for 1-D time signals. The time domain has been divided into two new domains (one consisting of the signal values at even positions and one consisting of the values at odd positions). Hence, it is called the radix-2 method. We will do the same for the DQFT. We apply the decimation method to the spatial domain in order to obtain a recursive algorithm (divide and conquer). If the domain is of the size (1×1) the DQFT is the identity. Otherwise we have the following.

Theorem 10.2.2 (Fast QFT). *Let $(f)_{M\times N}$ be a discrete two–dimensional signal and $M = N = 2^k \geq 2$. Then we have*

$$\mathcal{F}_q^D\{f\} = F^{ee} + w_i^{-u}F^{oe} + F^{eo}w_j^{-v} + w_i^{-u}F^{oo}w_j^{-v} \tag{10.4}$$

where

$$F^{ee} = \mathcal{F}_q^D\{f^{ee}\} = \mathcal{F}_q^D\{f_{2x,2y}\} \tag{10.5a}$$

$$F^{oe} = \mathcal{F}_q^D\{f^{oe}\} = \mathcal{F}_q^D\{f_{2x+1,2y}\} \tag{10.5b}$$

$$F^{eo} = \mathcal{F}_q^D\{f^{eo}\} = \mathcal{F}_q^D\{f_{2x,2y+1}\} \tag{10.5c}$$

$$F^{oo} = \mathcal{F}_q^D\{f^{oo}\} = \mathcal{F}_q^D\{f_{2x+1,2y+1}\} \tag{10.5d}$$

Proof. In order to show that the recursive formula (10.4) is correct we present a constructive proof. We apply the radix-2 method to the definition of the DQFT (10.1) and obtain

$$\mathcal{F}_q^D\{f\} = \sum_{x_1,y_1=0}^{N/2-1} \sum_{x_0,y_0=0}^{1} w_i^{-u(2x_1+x_0)} f_{2x_1+x_0,2y_1+y_0} w_j^{-v(2y_1+y_0)} \quad ,$$

by substituting $u = u_1N/2 + u_0$, $v = v_1N/2 + v_0$

$$\mathcal{F}_q^D\{f\} = \sum_{x_0,y_0=0}^{1} w_i^{-ux_0} \sum_{x_1,y_1=0}^{N/2-1} w_i^{-u_0 2x_1} f_{2x_1+x_0,2y_1+y_0} w_j^{-v_0 2y_1} w_j^{-vy_0}$$

and finally, because $w_i^{-Nu_1x_1} = w_j^{-Nv_1y_1} = 1$

$$\mathcal{F}_q^D\{f\} = \mathcal{F}_q^D\{f_{2x,2y}\} + w_i^{-u}\mathcal{F}_q^D\{f_{2x+1,2y}\} + \mathcal{F}_q^D\{f_{2x,2y+1}\}w_j^{-v} + w_i^{-u}\mathcal{F}_q^D\{f_{2x+1,2y+1}\}w_j^{-v}$$

Note that the signals $f_{2x,2y}, f_{2x+1,2y}, f_{2x,2y+1}$ and $f_{2x+1,2y+1}$ and their quaternionic Fourier transforms have the size $N/2$. □

In practical applications signals will not always have a size of a power of two. In that case the domain may be filled up by zeros. As a consequence, the period of the signal is changed, which has an effect on the reconstruction of the original signal.

The recursive formula (10.4) and the identity for $M = N = 1$ yield an algorithm for calculating the DQFT with complexity $cN^2 \operatorname{ld} N$ (see section 10.2.3). We need $\operatorname{ld} N = k$ recursive calls of (10.4).

A fast algorithm for the inverse transform can be derived in the same way as for the DQFT except for the normalizing factor and the sign in the exponential function. This method is called decimation of frequency. Nevertheless, it is advantageous to develop another algorithm which calculates the inverse of (10.4) in each step. Consequently, we obtain after step n exactly the same sub-spectra as in the FQFT just before the $(k-n)$th application of (10.4). The reason why this method is advantageous will be given in section 10.2.2.

Using the notations above we state the following.

Theorem 10.2.3 (Inverse FQFT). *The inverse of formula (10.4) reads* $(L = N/2)$

$$F^{ee}_{u,v} = \frac{1}{4}\left(F^q_{u,v} + F^q_{u+L,v} + F^q_{u,v+L,} + F^q_{u+L,v+L)}\right) \tag{10.6a}$$

$$F^{oe}_{u,v} = \frac{1}{4} w_i^u \left(F^q_{u,v} - F^q_{u+L,v} + F^q_{u,v+L)} - F^q_{u+L,v+L)}\right) \tag{10.6b}$$

$$F^{eo}_{u,v} = \frac{1}{4}\left(F^q_{u,v} + F^q_{u+L,v} - F^q_{u,v+L)} - F^q_{u+L,v+L)}\right) w_j^v \tag{10.6c}$$

$$F^{oo}_{u,v} = \frac{1}{4} w_i^u \left(F^q_{u,v} - F^q_{u+L,v} - F^q_{u,v+L)} + F^q_{u+L,v+L)}\right) w_j^v \tag{10.6d}$$

and therefore, the recursive execution of (10.6a-10.6d) reconstructs $f_{x,y}$ *from* $F^q_{u,v}$.

Proof. Since the DQFT (and hence each application of (10.4)) is a 1-1 mapping, we only have to show that the successive application of (10.6a)-(10.6d) and (10.4) yields the identity.

$$\begin{aligned}
&F^{ee} + w_i^{-u} F^{oe} + F^{eo} w_j^{-v} + w_i^{-u} F^{oo} w_j^{-v} \\
&= \frac{1}{4}\Big(\Big(F^q_{u,v} + F^q_{u+L,v} + F^q_{u,v+L,} + F^q_{u+L,v+L)}\Big) \\
&\quad + w_i^{-u} w_i^u \left(F^q_{u,v} - F^q_{u+L,v} + F^q_{u,v+L)} - F^q_{u+L,v+L)}\right) \\
&\quad + \left(F^q_{u,v} + F^q_{u+L,v} - F^q_{u,v+L)} - F^q_{u+L,v+L)}\right) w_j^v w_j^{-v} \\
&\quad + w_i^{-u} w_i^u \left(F^q_{u,v} - F^q_{u+L,v} - F^q_{u,v+L)} + F^q_{u+L,v+L)}\right) w_j^v w_j^{-v}\Big) \\
&= F^q_{u,v}
\end{aligned}$$

Consequently, (10.6a)-(10.6d) and (10.4) are inverse mappings. If the DQFT of a signal $f_{x,y}$ (size: $N \times N$) is calculated by (10.4), $\operatorname{ld} N$ levels of sub-spectra

are created. Eq. (10.4) describes the transition from one level to the next. We have proved that (10.6a)-(10.6d) do the inverse of (10.4) and therefore, they describe the transition from one level to the *previous*. Consequently, recursive execution of (10.6a)-(10.6d) yield the iDQFT. □

Due to this theorem, the equations (10.6a)-(10.6d) can be used to implement an inverse fast quaternionic Fourier transform. In each step one step of the (forward) FQFT is inverted. Hence, after k steps we have reconstructed the original signal. In other words, the recursive calls must stop for $N = 1$. In this case, the iDQFT is the identity. This procedure is illustrated in Fig. 10.1.

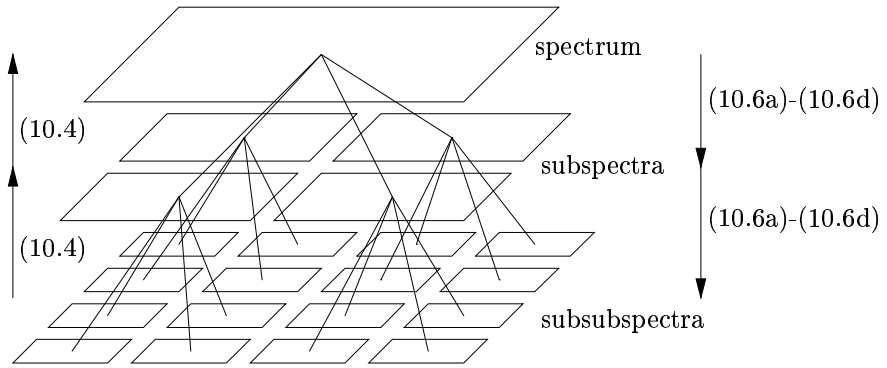

Fig. 10.1. Transitions between the (sub-)spectra

Of course, the performance of the implementation can be increased by e.g. unfolding one or two recursive calls (so the recursive function would stop for $N = 2$ or $N = 4$). However, we only consider principle optimizations in this chapter; further improvements of technical details are left to the programmer. Some general optimizations are given in the following section.

10.2.2 Optimizations by Hermite Symmetry

The first approach for optimizing the fast algorithms does not make use of the specific properties of the spectra of real signals (i.e. Hermite symmetry). The idea is the same as in equations (10.6a)-(10.6d). A phase of π yields only a change of the sign. This property will be called π-*phase* in the following. Hence, we have the same sum (10.4) for $F^q_{u,v}, F^q_{u+L,v}, F^q_{u,v+L}$ and $F^q_{u+L,v+L}$, except for the signs. Consequently, we only need to perform the multiplications once and can use the products four times. Hence, the multiplicative complexity is divided by four.

The second approach makes use of the quaternionic Hermite symmetry: we can

a) calculate four spectra in one step,
b) evaluate three quarters of each sub-spectrum by copying and applying one of the involutions (8.5).

If we choose a) and use the four spectra from the first decimation-step of (10.4), we make use of a method which is called *overlapping* [40].

This procedure is illustrated in the figure 10.2. The 8×8 real signal is mapped onto a 4×4 quaternionic signal. The real, the i-, the j-, and the k-imaginary part are denoted by r, i, j, and k, respectively.

		0		1		2		3	
		0	1	2	3	4	5	6	7
0	0	r	i	r	i	r	i	r	i
	1	j	k	j	k	j	k	j	k
1	2	r	i	r	i	r	i	r	i
	3	j	k	j	k	j	k	j	k
2	4	r	i	r	i	r	i	r	i
	5	j	k	j	k	j	k	j	k
3	6	r	i	r	i	r	i	r	i
	7	j	k	j	k	j	k	j	k

Fig. 10.2. A two-dimensional real signal is mapped onto a quaternionic signal

In (8.5), we have defined three involutions ($q \in \mathbb{H}$):

$$\alpha(q) = -jqj \tag{10.7a}$$
$$\beta(q) = -iqi \tag{10.7b}$$
$$\gamma(q) = -kqk = \alpha(\beta(q)) \tag{10.7c}$$

In order to reconstruct the four spectra from the overlapped one, we simply use the symmetry properties of the signals (see 8.3.6). The symmetries imply the following:

Theorem 10.2.4 (Overlapping). *Let* $\bar{f} = f^{ee} + if^{oe} + jf^{eo} + kf^{oo}$ *and* $\mathcal{F}_q^D\{\bar{f}\} = \bar{F}^q$. *Furthermore let*

$$F^{\alpha}_{u,v} = \alpha(\bar{F}^q_{u,-v}) \tag{10.8a}$$
$$F^{\beta}_{u,v} = \beta(\bar{F}^q_{-u,v}) \tag{10.8b}$$
$$F^{\gamma}_{u,v} = \gamma(\bar{F}^q_{-u,-v}). \tag{10.8c}$$

Then, the partial spectra are obtained by

$$4F^{ee}_{u,v} = \bar{F}^q_{u,v} + F^{\alpha}_{u,v} + F^{\beta}_{u,v} + F^{\gamma}_{u,v} \tag{10.9a}$$
$$4iF^{oe}_{u,v} = \bar{F}^q_{u,v} + F^{\alpha}_{u,v} - F^{\beta}_{u,v} - F^{\gamma}_{u,v} \tag{10.9b}$$
$$4jF^{eo}_{u,v} = \bar{F}^q_{u,v} - F^{\alpha}_{u,v} + F^{\beta}_{u,v} - F^{\gamma}_{u,v} \tag{10.9c}$$
$$4kF^{oo}_{u,v} = \bar{F}^q_{u,v} - F^{\alpha}_{u,v} - F^{\beta}_{u,v} + F^{\gamma}_{u,v}. \tag{10.9d}$$

Proof. Due to linearity of the Fourier transform we obtain the following table of symmetries (table 10.1) for the addends of the overlapped quaternionic spectrum.

Table 10.1. Symmetries of the addends of $\bar{F}^q$

Addend	real part	i-imag. part	j-imag. part	k-imag. part
$\mathcal{F}^D_q\{f^{ee}\}$	ee	oe	eo	oo
$i\mathcal{F}^D_q\{f^{oe}\}$	oe	ee	oo	eo
$j\mathcal{F}^D_q\{f^{eo}\}$	eo	oo	ee	oe
$k\mathcal{F}^D_q\{f^{oo}\}$	oo	eo	oe	ee

e: even
o: odd
f^{eo} is even wrt. the x-coordinate and odd wrt. the y-coordinate

Hence, each partial spectrum can be extracted from $\bar{F}^q$. Without loss of generality we will reconstruct the spectrum F^{ee}. We obtain the real part of F^{ee} by

$$4\,\mathcal{R}(F^{ee}_{u,v}) = \mathcal{R}(\bar{F}^q_{u,v} + \bar{F}^q_{-u,v} + \bar{F}^q_{u,-v} + \bar{F}^q_{-u,-v}),$$

since all odd parts yield zero. Analogously, we obtain the other parts, namely

$$4\,\mathcal{I}(F^{ee}_{u,v}) = \mathcal{I}(\bar{F}^q_{u,v} - \bar{F}^q_{-u,v} + \bar{F}^q_{u,-v} - \bar{F}^q_{-u,-v}),$$
$$4\,\mathcal{J}(F^{ee}_{u,v}) = \mathcal{J}(\bar{F}^q_{u,v} + \bar{F}^q_{-u,v} - \bar{F}^q_{u,-v} - \bar{F}^q_{-u,-v}),$$
$$4\,\mathcal{K}(F^{ee}_{u,v}) = \mathcal{K}(\bar{F}^q_{u,v} - \bar{F}^q_{-u,v} - \bar{F}^q_{u,-v} + \bar{F}^q_{-u,-v}).$$

Due to linearity we can exchange the sums and the selection of the components which yield formula (10.9a). The other three spectra can be reconstructed analogously. □

Now we consider the optimization b) which means direct application of the Hermite symmetry. If we have calculated a spectral value $F^q_{u,v}$ (this might be a spectral value of a sub-spectrum, too), we obtain three more spectral values by

$$F^q_{-u,v} = \beta(F^q_{u,v}) \tag{10.10a}$$
$$F^q_{u,-v} = \alpha(F^q_{u,v}) \tag{10.10b}$$
$$F^q_{-u,-v} = \gamma(F^q_{u,v}). \tag{10.10c}$$

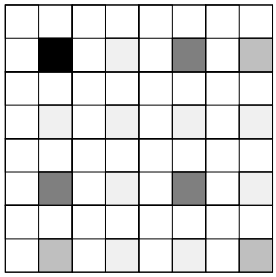

Fig. 10.3. Two effects speed up the calculation of the spectrum:
■ calculated value
■ changed signs
■ Hermite symmetry
□ signs and Hermite symmetry

Note, if u (v) is zero or $N/2$, we have $(-u, v) = (u, v)$ $((u, -v) = (u, v))$, since F^q is N-periodic.

Both effects (the π-phase and the Hermite symmetry) are illustrated in the following figure 10.3.

The black cell (calculated value) is the only cell which has to be *explicitly* calculated. The dark grey cells (changed signs) are obtained without additional multiplications due to the π-phase. The middle grey cells (Hermite symmetry) are obtained from the black cell without any additional arithmetic operations by applying the involutions. Finally, the light grey cells are obtained from the dark grey cells by the Hermite symmetry.

Both methods, a) and b), reduce the whole complexity by a factor of four. Though we only considered the forward transform, we can decrease the complexity of the inverse transform using the same methods. This is possible, since we formulated an inverse transform which reconstructs the Hermite-symmetric sub-spectra (a simple decimation of frequency does not yield such an algorithm).

The overlapping additionally reduces the memory complexity by a factor of four (except for the reconstruction procedure). This is quite important if we consider the execution of the algorithm on a computer.

The memory complexity and the localization of data have to be considered under two aspects: cache size and memory size. If the data exceeds the cache size, we have cache misses (in image processing we have this on most machines for e.g. images of 512×512 pixels). During the execution of the algorithm the data have to be moved between cache and main memory. It is advantageous if the algorithm acts locally, which means that the algorithm uses only a small part of the data (less than the cache size). The latency time for copying data from the main memory to the cache is similar to the latency time of floating point operations. Hence, if we would have a *branch first algorithm* (the recursion is executed layer by layer), the data must be moved in each incarnation of recursion. The resulting complexity of the data handling would be similar to that one of the calculation itself.

Fortunately, we have a *depth first algorithm* which means that the recursion is first finished completely for the first call, then for the second call and so on.Hence, we have two times a cache miss for each value if the cache size

is at least a quarter of the size of the spectrum.

If the data even exceeds the main memory, the computer uses disk space to extend the main memory (*swapping*). Since the access to the hard disk is one thousand times slower than the memory access, the time complexity of the algorithm itself becomes obsolete. The only important feature of the algorithm is to work as locally as possible. This problem might appear if 3-D data or image sequences are transformed.

In the following, we only consider the time complexity, since we have a 2-D transform and we assume that the cache is nearly as large as the data.

10.2.3 Complexities

In this section we consider the complexities of the presented algorithms. Since modern computers calculate floating point multiplications much faster than in former times, we do not restrict to the multiplicative complexities.

In literature, we have found some algorithms which reduce the multiplicative complexity of complex and quaternionic multiplications. Complex multiplications can be performed with three multiplications (instead of four) and quaternionic multiplications can be performed with nine multiplications (instead of 16) [208]. This decrease of the multiplicative complexity can only be achieved by increasing the additive complexity. Since multiplications are performed as fast as additions on today's processors (e.g. SUN UltraSPARC-II [231], SGI R10000 [217], DEC Alpha 21164PC [49]), the fastest algorithm is the one with the least number of operations – multiplications plus additions.

Consequently, we always consider the multiplicative complexity, the additive complexity, and the sum of both.

Although the FQFT is performed in the quaternion algebra, we need not calculate the general quaternionic multiplication. We know that in each multiplication of (10.4) one factor (the exponential term) is the element of a subalgebra which is isomorphic to the complex algebra. Hence, the complexity of such a multiplication can be reduced to eight multiplications and four additions.

Formula (10.4) yields four multiplications and three additions in the quaternion algebra or 32 multiplications and $16 + 12 = 28$ additions in the algebra of real numbers for each frequency. Hence, we have $32N^2$ multiplications and $28N^2$ additions for each recursive call. If the effect of the π-phase is used, we reduce the multiplications in the quaternion algebra by a factor of four, i.e. we need $8N^2$ multiplications and $16N^2$ additions in the algebra of real numbers for evaluating (10.4) for all frequencies. Since we have $\mathrm{ld}\, N$ recursive calls, we have a total complexity of $8N^2 \,\mathrm{ld}\, N$ multiplications and $16N^2 \,\mathrm{ld}\, N$ additions (total: $24N^2 \,\mathrm{ld}\, N$ floating point operations).

If we have a real signal, the complexity can be reduced to approximately one quarter. Firstly, we consider overlapping. Since the size of the new signal domain is reduced to $N^2/4$, the complexity is divided by four and $\mathrm{ld}\, N$

is substituted by $\mathrm{ld}\,\frac{N}{2} = -1 + \mathrm{ld}\,N$ (one recursive call less). The reconstruction of the overlapped spectrum (10.9a-d) increases the quadratic additive complexity by three quaternionic additions (or twelve real additions) and one real multiplication per frequency (the factor four can be eliminated component-wise in $\bar{F}^q$). The last recursive call of (10.4) increases the complexities according to the case without Hermite symmetry (e.g. eight real multiplications and 16 real additions per frequency). Finally, the overall complexity reads $\frac{8}{4}N^2(-1 + \mathrm{ld}\,N) + (1 + 8)N^2 = (7 + 2\,\mathrm{ld}\,N)N^2$ multiplications and $\frac{16}{4}N^2(-1 + \mathrm{ld}\,N) + (12 + 16)N^2 = (24 + 4\,\mathrm{ld}\,N)N^2$ additions (total: $(31 + 6\,\mathrm{ld}\,N)N^2$ operations).

If the Hermite symmetry is directly used, we only need calculate one quarter of each sub-spectrum. The other three quarters are evaluated by the automorphisms (8.5). Since there is no symmetric frequency if u and v are zero or $N/2$ and there is only one instead of three symmetric frequencies if either u or v are zero or $N/2$, we obtain a high additional quadratic complexity. Furthermore, this algorithm has a four times higher memory complexity than the overlapping algorithm (except for the last step). Hence, it is not advantageous to realize this fast algorithm.

Nevertheless, we can decrease the complexity of the overlapping algorithm if we reconstruct only one quarter of the spectrum by formulae (10.9a-d) and (10.4) and copy the other three quarters. This reduces the complexity of the reconstruction to $1 + \frac{8}{4} = 3$ real multiplications and $\frac{28}{4} = 7$ real additions per frequency. Due to the zero-frequencies (and the $N/2$-frequencies), we have an additional *linear* complexity, which is neglected in these considerations. The total complexity of this algorithm is $(1 + 2\,\mathrm{ld}\,N)N^2$ multiplications and $(3 + 4\,\mathrm{ld}\,N)N^2$ additions (total complexity: $(4 + 6\,\mathrm{ld}\,N)N^2$ operations).

The same complexity as for the last algorithm is obtained if four real signals are transformed at the same time. The four signals are mapped to the four parts of a quaternionic signal which is transformed with a complexity of $24N^2\,\mathrm{ld}\,N$ operations (i.e. $6N^2\,\mathrm{ld}\,N$ per spectrum). Afterwards, we reconstruct the four spectra similarly to overlapping (using the Hermite-symmetry), which yields the same additional complexity (N^2 multiplications and $\frac{12}{4}N^2$ additions per spectrum). All results are summarized in table 10.2.

Table 10.2. Complexities of the considered algorithms

Algorithm	multiplications	additions	operations
FQFT (π-phase)	$8N^2\,\mathrm{ld}\,N$	$16N^2\,\mathrm{ld}\,N$	$24N^2\,\mathrm{ld}\,N$
FQFT/overlapping	$N^2(7 + 2\,\mathrm{ld}\,N)$	$N^2(24 + 4\,\mathrm{ld}\,N)$	$N^2(31 + 6\,\mathrm{ld}\,N)$
FQFT/overl.+sym.	$N^2(1 + 2\,\mathrm{ld}\,N)$	$N^2(3 + 4\,\mathrm{ld}\,N)$	$N^2(4 + 6\,\mathrm{ld}\,N)$
FQFT/four spectra	$N^2(1 + 2\,\mathrm{ld}\,N)$	$N^2(3 + 4\,\mathrm{ld}\,N)$	$N^2(4 + 6\,\mathrm{ld}\,N)$

Note, that the algorithm using overlapping with Hermite symmetry in the last step is very complicated to implement. The enormous code length of the implementation could possibly slow down the algorithm more than it is sped up by the Hermite symmetry. Additionally, there is an alternative way to calculate the DQFT which is even faster than all algorithms above (see 10.4.2).

10.3 Discrete and Fast n-Dimensional Transforms

In this section we extend the definition of the discrete QFT for the n-dimensional commutative hypercomplex Fourier transform (HFT). We state the inverse discrete HFT and a fast algorithm (FHFT). Further on, we generalize the π-phase and the overlapping for n dimensions and consider the complexities.

10.3.1 Discrete Commutative Hypercomplex Fourier Transform and Fast Commutative Hypercomplex Fourier Transform

Recall the correspondence of the QFT and the HFT2 (9.3.1): both transforms yield the same coefficient in the case of real signals. We have extended that concept to n dimensions in Sec. 9.4.2, i.e. the coefficients of the HFTn are the same as those of the n-D CFT of a real signal. For the discrete n-D CFT and the discrete HFTn we have the same correspondence as for the continuous transforms, since we have not used the properties of the domain (i.e. if it is infinite and continuous or finite and discrete) in the proof. Therefore, we can calculate the Clifford spectrum via the HFTn and it is sufficient to give the definition of the discrete HFTn in this section.

Additionally, it is not possible to develop directly a fast algorithm for the CFT in the same way as we did for the QFT. We can apply the decimation method in a straightforward way only for commutative algebras because we have to exchange some factors. The QFT is an exception since we have *two* ways of multiplying and therefore we can extract one factor of the QFT kernel to the left (the i-term) and one to the right (the j-term). If $n \geq 3$ we have not enough ways of multiplying (e.g. from the top), so we must really permute some factors. Since the hypercomplex algebra introduced in section 9.4.2 is commutative, we use the commutative hypercomplex Fourier transform for developing a fast algorithm. In this respect, we want to annotate that we have originally introduced the commutative hypercomplex algebra for the purpose of developing fast algorithms. Therefore, the whole theory presented in Cha. 9 has been motivated by the mathematical properties which we needed for the algorithm.

We have shown in The. 9.3.1 that the coefficients of both transforms are the same if the signal is real-valued and due to linearity of the HFT, we can

even calculate the Clifford spectrum of a Clifford-valued signal (see 10.4.2). Thus, by developing the discrete and fast HFT, we indirectly obtain a discrete and fast CFT, respectively.

To start with, we now define the discrete HFT in analogy to the DQFT by sampling the continuous transform. The considerations we made for the QFT in section 10.2.1 (about the formal construction of discrete signals) are the same for the HFT, of course.

Definition 10.3.1 (Discrete HFT). *Let f be an n-dimensional discrete signal of the size N^n and let I_n be defined as*

$$I_n = \begin{bmatrix} \mathbf{e}_1 \wedge \mathbf{e}_{1+n} & 0 & \dots & \dots \\ 0 & \mathbf{e}_2 \wedge \mathbf{e}_{2+n} & 0 & \dots \\ \dots & 0 & \ddots & 0 \\ \dots & \dots & 0 & \mathbf{e}_n \wedge \mathbf{e}_{2n} \end{bmatrix}.$$

Then, the discrete HFT of f is defined by

$$F^h_{\boldsymbol{u}} = \sum_{\boldsymbol{x} \in \{0,\dots,N-1\}^n} f_{\boldsymbol{x}} e^{-2\pi \boldsymbol{x} I_n \boldsymbol{u}^T N^{-1}}. \tag{10.11}$$

Note that the discrete signal can have different lengths in each coordinate, i.e. a 3-D signal can have the size $M \times N \times K$ with arbitrary (true positive) M, N, K. We formulated the DHFT for signals with equal length in each dimension, since the formula is more compact and the FHFT (see (10.13)) can only be applied to such signals. Nevertheless, every signal can be embedded in a larger signal of that form (with a changed period, of course).

In analogy to the iDQFT we state a theorem for the inverse transform in the following. The transform and its proof are extensions of the two-dimensional case (10.2). Note that there is one important difference between the 2-D DHFT and the DQFT. The latter is formulated using a special order of multiplications (i-term from the left and j-term from the right). This order implies that the iDQFT must have the same sequence of factors, since the two i-terms and the two j-terms must compensate *directly* each other, respectively.

In contrast to this, the sequence of factors is irrelevant in the case of the DHFT and the iDHFT, since they are commutative. Hence, they can be formulated putting all exponential functions in one exponential function at the end. Additionally, we need not take care of the order of the multiplications in the proof of the theorem.

Theorem 10.3.1 (Inverse discrete HFT). *Let f be an n-dimensional discrete signal of the size N^n and F^h its DHFT. Then, the following equation holds true*

$$f_{\boldsymbol{x}} = \frac{1}{N^n} \sum_{\boldsymbol{u} \in \{0,\dots,N-1\}^n} F^h_{\boldsymbol{u}} e^{2\pi \boldsymbol{x} I_n \boldsymbol{u}^T N^{-1}}. \tag{10.12}$$

Proof. Applying (10.12) to the discrete HFT of f yields:

$$\begin{aligned}
&\frac{1}{N^n} \sum_{\boldsymbol{u}\in\{0,\dots,N-1\}^n} F^h_{\boldsymbol{u}} e^{2\pi \boldsymbol{x}' I_n \boldsymbol{u}^T N^{-1}} \\
&= \frac{1}{N^n} \sum_{\boldsymbol{u}\in\{0,\dots,N-1\}^n} \sum_{\boldsymbol{x}\in\{0,\dots,N-1\}^n} f_{\boldsymbol{x}} e^{-2\pi \boldsymbol{x} I_n \boldsymbol{u}^T N^{-1}} e^{2\pi \boldsymbol{x}' I_n \boldsymbol{u}^T N^{-1}} \\
&= \frac{1}{N^n} \sum_{\boldsymbol{x}\in\{0,\dots,N-1\}^n} \sum_{\boldsymbol{u}\in\{0,\dots,N-1\}^n} e^{2\pi (\boldsymbol{x}'-\boldsymbol{x}) I_n \boldsymbol{u}^T N^{-1}} f_{\boldsymbol{x}} \\
&= \frac{1}{N^n} \sum_{\boldsymbol{x}\in\{0,\dots,N-1\}^n} f_{\boldsymbol{x}} N^n \delta_{\boldsymbol{x}'-\boldsymbol{x}} \\
&= f_{\boldsymbol{x}'}.
\end{aligned}$$

Hence, (10.12) is the inverse DHFT. □

Now, having a discrete transform in a commutative algebra, we are able to state a fast algorithm for the HFT, since we can apply the decimation method. The following theorem is the extension of the FQFT (10.4). The notes we made for the DQFT concerning the order of multiplications are valid for the FHFT, too. Therefore, the recursive formula for the 2-D FHFT differs from that one of the FQFT. Nevertheless, both formulae yield the same coefficients for a real signal. Besides, the proof is simplified by the commutativity.

Theorem 10.3.2 (Fast HFT). *Let $f_{\boldsymbol{x}}$ be a discrete n-dimensional signal of the size N^n with $N = 2^k$. Then we have*

$$F^h_{\boldsymbol{u}} = \sum_{\substack{x_{l0}=0 \\ 1\le l\le n}}^{1} F_{\boldsymbol{x}_0 \boldsymbol{u}} e^{-2\pi \boldsymbol{x}_0 I_n \boldsymbol{u}/N} \tag{10.13}$$

where

$$F_{\boldsymbol{x}_0 \boldsymbol{u}} = \sum_{\substack{x_{l1}=0 \\ 1\le l\le n}}^{N/2-1} f_{2\boldsymbol{x}_1+\boldsymbol{x}_0} e^{-2\pi 2 \boldsymbol{x}_1 I_n \boldsymbol{u}/N}. \tag{10.14}$$

Proof. Since the proof of equation (10.13) is the same as the one for the FQFT (10.4) except for the dimension and the notation of even / odd signal-components, it is omitted. Instead of using "e" for even and "o" for odd signal-components, they are denoted by "0" and "1", respectively. Accumulating the indices yields an index-collection, which is an n-dimensional vector. This vector is identical to $\boldsymbol{x}_0 = \boldsymbol{x} \bmod 2$. Again, the order of multiplications is irrelevant. □

The notes which have been made for the FQFT concerning the signal length are valid for the n-D FHFT as well. Obviously, the restriction to the signal size becomes a drawback for higher dimensions, since the area filled up by zeros might be a multiple of the original signal size. For those cases the row-column algorithm in section 10.4.1 evaluates the spectrum with less complexity.

The inverse transform can be calculated by the same algorithm as the FHFT using positive exponential terms and a norming factor. Alternatively, an iDHFT can be developed by reconstructing the sub-spectra as it has been done for the iFQFT. Since such an iFHFT does not include any new ideas which have not been mentioned so far, we omit the explicit formulation of the algorithm.

10.3.2 Optimizations and Complexities

For the FHFT we have the same optimizations as for the FQFT. Essentially, we can apply the π-phase method and overlapping. We will consider only these two methods in the following.

Analogously to the FQFT, we exclusively have multiplications by complex factors. Note that for an implementation of the FQFT or FHFT, we reduce the complex exponential functions to sine and cosine. Therefore, we have to decompose the compact exponential function in (10.13) into n products of the sums of sines and cosines.

The multiplication of a general element U of the 2^n dimensional commutative algebra by a complex factor yields 2^{n+1} real multiplications and 2^n additions. The addition of two general elements U and V yields 2^n real additions.

One application of (10.13) yields

$$\sum_{i=0}^{n} \binom{n}{i} i = n \sum_{i=0}^{n-1} \binom{n-1}{i} = n2^{n-1}$$

of these special multiplications and $2^n - 1$ additions per frequency. Hence, we obtain $n2^{2n}$ real multiplications and

$$n2^{2n-1} + 2^{2n} - 2^n = \left(\frac{n}{2} + 1 - 2^{-n}\right) 2^{2n}$$

real additions. Therefore, we have $n2^{2n}N^n$ multiplications and $(\frac{n}{2} + 1 - 2^{-n})2^{2n}N^n$ additions for the whole spectrum. Since the algorithm is performed $\operatorname{ld} N$ times, we must multiply these complexities by $\operatorname{ld} N$ for obtaining the complexity of the whole algorithm.

Using the effect of the π-phase, we reduce the number of multiplications by 2^n. Consequently we need $n2^nN^n \operatorname{ld} N$ real multiplications and $(\frac{n}{2} + 2^n - 1)2^nN^n \operatorname{ld} N$ real additions for the whole algorithm.

Now, we describe the overlapping for the n-dimensional case: firstly, the domain of the signal $f_{\boldsymbol{x}}$ of the size N^n is divided into 2^n parts by the substitution $\boldsymbol{x} = 2\boldsymbol{x}_1 + \boldsymbol{x}_0$ where $x_{0j} = x_j \bmod 2$ and $x_{1j} = \lfloor x_j/2 \rfloor$. Each signal value $f_{2\boldsymbol{x}_1+\boldsymbol{x}_0}$ is mapped onto one component of a new hypercomplex-valued signal $\bar{f}_{\boldsymbol{x}_1}$. Which value is mapped onto which component is determined by $\boldsymbol{x}_0$.
We define a coding function

$$\boldsymbol{C} : \mathcal{P}(\{1,\ldots,n\}) \longrightarrow \{0,1\}^n$$

with

$$C_k(j) = \begin{cases} 1 & \text{if} \quad k \in j \\ 0 & \text{else} \end{cases} \tag{10.15}$$

and use the following mapping:

$$\mathcal{I}_j\{\bar{f}_{\boldsymbol{x}_1}\} = f_{2\boldsymbol{x}_1+\boldsymbol{C}(j)} \tag{10.16}$$

for all $j \in \mathcal{P}(\{1,\ldots,n\}) \setminus \emptyset$ and

$$\mathcal{R}\{\bar{f}_{\boldsymbol{x}_1}\} = f_{2\boldsymbol{x}_1}. \tag{10.16'}$$

If the set j is empty, the real part is taken, otherwise $\mathcal{I}_j\{\bar{f}_{\boldsymbol{x}_1}\}$. Consequently, a value at a position which is odd with respect to coordinate k is mapped onto an imaginary part which contains i_k.

Calculating the spectrum of $\bar{f}_{\boldsymbol{x}_1}$ yields

$$\bar{F}_{\boldsymbol{u}_0} = \sum_{j\in\mathcal{P}(\{1,\ldots,n\})} i_j F_{\boldsymbol{C}(j)\boldsymbol{u}_0} \tag{10.17}$$

where $F_{\boldsymbol{x}_0\boldsymbol{u}_0}$ are the sub-spectra from (10.13).

The sub-spectra can be extracted from (10.17) by inverting the involutions (8.5):

$$2^n i_j F_{\boldsymbol{C}(j)\boldsymbol{u}_0} = \sum_{k\in\mathcal{P}(\{1,\ldots,n\})} \alpha_k(\bar{F}_{\boldsymbol{u}_0^k})(-1)^{\mathrm{card}(j\cap k)} \tag{10.18}$$

where $u_{0i}^k = \begin{cases} -u_{0i} & \text{if } i \in k \\ u_{0i} & \text{else} \end{cases}$ and $\mathrm{card}(M)$ is the cardinality of M. Finally, the spectrum $F_{\boldsymbol{u}}$ is calculated by use of formula (10.13).

Roughly speaking, overlapping reduces the complexity by a factor of four. The total complexity is a little bit worse, since we have an additional N^n-complexity for the reconstruction. This additional complexity can be calculated analogously to that one in section 10.2.3. Finally, we obtain the following table 10.3:

We want to justify the neglect of the exact evaluation of the N^n-complexities by the fact that we will present a faster approach to calculate the DHFT in section 10.4.

Table 10.3. Complexities of the considered algorithms

Complexity	FHFT (π-phase)	FHFT/overlapping
multiplications	$n2^n N^n \operatorname{ld} N$	$nN^n \operatorname{ld} N + \mathcal{O}(N^n)$
additions	$(\frac{n}{2} + 2^n - 1)2^n N^n \operatorname{ld} N$	$(\frac{n}{2} + 2^n - 1)N^n \operatorname{ld} N + \mathcal{O}(N^n)$
operations	$(\frac{3n}{2} + 2^n - 1)2^n N^n \operatorname{ld} N$	$(\frac{3n}{2} + 2^n - 1)N^n \operatorname{ld} N + \mathcal{O}(N^n)$

The memory complexity is very crucial in the n-dimensional case, since the signal size increases exponentially with the dimension. The presented algorithm gets very slow, if the signal size is greater than the main memory. The reason for this is the global data access of the algorithm in the first recursive steps (the whole domain, 2^{-n}th of the domain, ...). In the section 10.4.1 we present an algorithm which only acts on 1-D sub-signals. For very big signal sizes this algorithm needs less swapping of the data.

10.4 Fast Algorithms by FFT

In this section we describe two methods of evaluating the DHFT and the DCFT by applying complex-valued FFT algorithms. The first method which is called *row-column method* cascades 1-D FFTs in order to calculate the spectrum. The second method uses the isomorphism between the commutative hypercomplex algebra and the indirect product of complex algebras. This isomorphism maps the HFT of a signal onto two complex spectra of the signal and vice versa. We obtain a method to calculate the hypercomplex spectrum from the complex spectrum. The last paragraph deals with the complexities of the presented algorithms.

10.4.1 Cascading 1-D FFTs

There is one simple approach to calculate the DCFT (or the DHFT) by the 1-D FFT algorithm. The method is called *row-column algorithm* and we will firstly introduce it for the DQFT. Of course, it can be generalized for arbitrary dimensions.

The idea is as follows. We pick up one coordinate, without loss of generality we will take the first one (i.e. the x-coordinate). Then, we calculate the 1-D FFT of each row and put the results in two domains of the same size, one for the real part and one for the imaginary part, denoted by f^R and f^I, respectively.

Next, each column of these two signals f^R and f^I is transformed by the 1-D FFT. Finally, each part of both spectra is mapped to one part of the quaternionic spectrum. The real spectrum of f^R is the real part of F^q, the real spectrum of f^I is the i-imaginary part of F^q, the imaginary spectrum of f^R

is the j-imaginary part of F^q and the remaining spectrum is the k-imaginary part of F^q. The method is illustrated in the following figure 10.4.

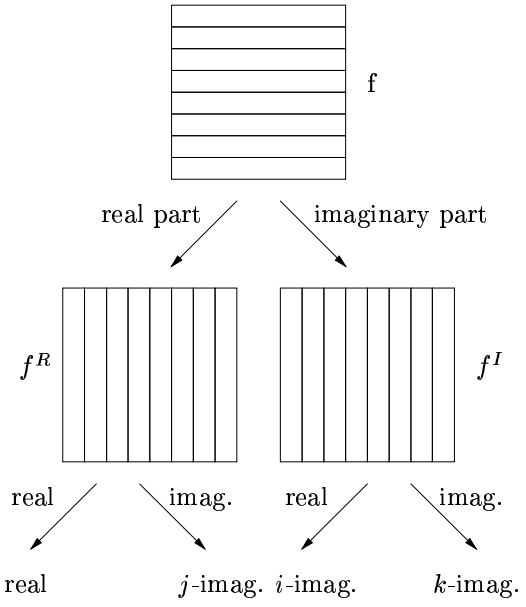

Fig. 10.4. The row-column algorithm for a 2-D signal (the arrow means application of the 1-D FFT)

Theorem 10.4.1 (Row-Column algorithm). *The application of the row-column algorithm to a signal yields its DQFT in the 2-D case.*

Proof. We start this proof with the DQFT of a signal f.

$$F^q_{u,v} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} e^{-i2\pi uxM^{-1}} f_{x,y} e^{-j2\pi vyN^{-1}}$$

Now, we will set some parenthesis to define N new 1-D functions $f^y_x = f_{x,y}$. The 1-D Fourier transform of f^y_x is denoted by F^y_u (complex-valued):

$$\begin{aligned} F^q_{u,v} &= \sum_{y=0}^{N-1} \left(\sum_{x=0}^{M-1} e^{-i2\pi uxM^{-1}} f^y_x \right) e^{-j2\pi vyN^{-1}} \\ &= \sum_{y=0}^{N-1} F^y_u e^{-j2\pi vyN^{-1}} \\ &= \sum_{y=0}^{N-1} \mathcal{R}\{F^y_u\} e^{-j2\pi vyN^{-1}} + i \sum_{y=0}^{N-1} \mathcal{I}\{F^y_u\} e^{-j2\pi vyN^{-1}} \end{aligned}$$

The 1-D Fourier transform of $\mathcal{R}\{F^y_u\}$ and $\mathcal{I}\{F^y_u\}$ are denoted by $F^R_{u,v}$ and

$F^I_{u,v}$, respectively. Note that these transforms use y as the spatial coordinate and v as the frequency.

$$\begin{aligned} F^q_{u,v} &= \sum_{y=0}^{N-1} \mathcal{R}\{F^y_u\}(\mathcal{R}\{e^{-i2\pi vyN^{-1}}\} + j\mathcal{I}\{e^{-i2\pi vyN^{-1}}\}) \\ &\quad + i\sum_{y=0}^{N-1} \mathcal{I}\{F^y_u\}(\mathcal{R}\{e^{-i2\pi vyN^{-1}}\} + j\mathcal{I}\{e^{-i2\pi vyN^{-1}}\}) \\ &= \mathcal{R}\{F^R_{u,v}\} + j\mathcal{I}\{F^R_{u,v}\} + i\mathcal{R}\{F^I_{u,v}\} + k\mathcal{I}\{F^I_{u,v}\} \end{aligned}$$

Hence, we obtain exactly the coefficients as described in the row-column algorithm and due to the fact that two quaternions are equal if and only if all coefficients are equal, the theorem is proved. □

For the DHFT we obtain an analogous algorithm. In each step the number of transforms is doubled, since the real and imaginary parts are transformed separately. Every exponential factor $e^{-i_j 2\pi x_j u_j N_j^{-1}}$ is rewritten as $\mathcal{R}\{e^{-i2\pi x_j u_j N_j^{-1}}\} + i_j\mathcal{I}\{e^{-i2\pi x_j u_j N_j^{-1}}\}$. Hence, we double the number of Fourier transforms $n-1$ times and consequently we obtain $2 \cdot 2^{n-1} = 2^n$ coefficients which we need for the DHFT. Furthermore, we have all imaginary units: $\Pi_{j=1}^n(1+i_j)$.

Since the order of the imaginary units in the product is ascending, we can perform the multiplications as well in the Clifford algebra as in the commutative hypercomplex algebra. This is an improvement, compared to the FHFT (see below). Besides, the drawback of all radix-2 methods (i.e. that the signal size must be a power of two) is less serious for the row-column algorithm. For the FHFT the signal length in each coordinate must be filled up to the same (i.e. the greatest) power of two. The row-column algorithm can handle different signal lengths in each coordinate.

The algorithm must be modified, if the spatial signal f is not real-valued. Consider the two-dimensional case. If the QFT is defined with one exponential factor to the left and one to the right, we can split the signal f into one complex signal $f^1 = \mathcal{R}\{f\} + i\mathcal{I}\{f\}$ and one complex signal $f^2 = \mathcal{J}\{f\} + i\mathcal{K}\{f\}$. Obviously $f = f^1 + f^2 j$. The 1-D Fourier transform (row-wise) of f^1 and f^2 are denoted F^1 and F^2, respectively. Now, we calculate the 1-D Fourier transform (column wise) of $\mathcal{R}\{F^1\} + i\mathcal{R}\{F^2\}$ and $\mathcal{I}\{F^1\} + i\mathcal{I}\{F^2\}$, denoted by F^{RJ} and F^{IK}, respectively. They are identical to $\mathcal{R}\{F^q\}+i\mathcal{J}\{F^q\}$ and $\mathcal{I}\{F^q\} + i\mathcal{K}\{F^q\}$. Hence, we obtain

$$F^q = \mathcal{R}\{F^{RJ}\} + j\mathcal{I}\{F^{RJ}\} + i(\mathcal{R}\{F^{IK}\} + j\mathcal{I}\{F^{IK}\}). \tag{10.19}$$

Considering the CFT, a new problem occurs. As we can see in the case of the 2-D CFT, the spectrum is different depending on the i-exponential term standing on the left or on the right of the signal. If we have more than two dimensions, we must use the CFT. Therefore, in order to compensate

the exchanged imaginary units, we have to alter some signs. These signs can be evaluated by formally splitting the CFT into 1-D FTs. The formulation of the specific rules for each dimension is omitted here, since the rules can be derivated from the eigenvalues in Lem. 9.4.3 and by the different signs in the multiplication tables of the algebras $\mathbb{R}_{0,n}$ and $\mathcal{H}_n$. Furthermore, we are mostly interested in image analysis and for that purpose the 2-D algorithm is sufficient.

10.4.2 HFT by Complex Fourier Transform

We have proved in section 9.4.2 that the commutative hypercomplex algebra is isomorphic to the Cartesian product of complex algebras. Let us consider this isomorphism more closely with respect to the Fourier transform. Firstly, we take the two-dimensional case.

We obtain the two complex coefficients η, ξ from the hypercomplex value F by the formulae

$$\eta = \mathcal{R}\{F\} - \mathcal{K}\{F\} + i(\mathcal{I}\{F\} + \mathcal{J}\{F\}) \tag{10.20a}$$

$$\xi = \mathcal{R}\{F\} + \mathcal{K}\{F\} + i(\mathcal{I}\{F\} - \mathcal{J}\{F\}) \tag{10.20b}$$

Assume now F being a hypercomplex spectrum. If we consider η, we can see that it is equal to the complex spectrum of the same signal. It is an amazing fact, that two totally different mappings (one between two algebras and one between two signal-theoretic concepts) are equal.

Even more amazing is the correspondence between the second complex component ξ and the complex spectrum. If F is the spectrum of a real signal, i.e. it is Hermite symmetric, we obtain ξ from η by inverting the v-coordinate (the reversion in v-direction yields a changed sign in the j- and k-components of the spectrum).

Using this knowledge, we can develop another fast algorithm for the HFT2 based on a complex 2-D FFT. Assume that the signal is real-valued. Now, calculate its complex spectrum, using an ordinary 2-D FFT algorithm. This spectrum is equal to η. Next, invert the v-axis in order to obtain the ξ component. Last, use the inverse of (10.20a and b) in order to reconstruct the hypercomplex spectrum.

The whole procedure can be shortened by writing

$$F^h = \mathcal{R}\{F^e\} + i\mathcal{I}\{F^e\} + j\mathcal{I}\{F^o\} - k\mathcal{R}\{F^o\} \tag{10.21}$$

where F is the complex spectrum, $F^e_{u,v} = 1/2(F_{u,v} + F_{u,-v})$ and $F^o_{u,v} = 1/2(F_{u,v} - F_{u,-v})$.

Up to now, we have exclusively considered spectra of real signals. In section 10.3.1 we have already mentioned that the DCFT of a Clifford-valued signal can be calculated using the DHFT. In order to develop such an algorithm we use the following derivation starting with the 2-D DCFT (not the DQFT):

$$
\begin{aligned}
F^q &= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}(\mathcal{R}\{f\}+i\mathcal{I}\{f\}+j\mathcal{J}\{f\}+k\mathcal{K}\{f\})e^{-i2\pi ux/M}e^{-j2\pi vy/N} \\
&= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\mathcal{R}\{f\}e^{-i2\pi ux/M}e^{-j2\pi vy/N} \\
&\quad +i\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\mathcal{I}\{f\}e^{-i2\pi ux/M}e^{-j2\pi vy/N} \\
&\quad +j\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\mathcal{J}\{f\}e^{-i2\pi ux/M}e^{-j2\pi vy/N} \\
&\quad +k\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\mathcal{K}\{f\}e^{-i2\pi ux/M}e^{-j2\pi vy/N} \\
&= F^R + iF^I + jF^J + kF^K
\end{aligned}
$$

where F^R, F^I, F^J, and F^K are spectra of the real signals $\mathcal{R}\{f\}, \mathcal{I}\{f\}, \mathcal{J}\{f\}$, and $\mathcal{K}\{f\}$, respectively. Hence, they can be evaluated either in the Clifford algebra or in the commutative hypercomplex algebra. Therefore, they can be calculated via the complex spectra. Only the last step (the multiplication of the partial spectra by the imaginary units) must be calculated in the Clifford algebra.

Since the isomorphism is proved for any dimension, every hypercomplex spectrum can be calculated using the complex Fourier transform. If the signal is real, the method is straightforward. The first complex coefficient is obtained by the complex Fourier transform. The other coefficient are calculated by inverting each coordinate axis except for the first. If the signal is Clifford-valued, each component must be transformed separately and afterwards the imaginary units are multiplied to the partial spectra using the Clifford algebra multiplication.

10.4.3 Complexities

In this section we consider some complexities of the presented algorithms. Though the row-column algorithm is most advantageous in the case where the signal length varies widely with respect to the different coordinates, we consider the case where all signal lengths are the same, in order to compare the row-column algorithm to the other ones.

We start with the complexities for the two-dimensional case. We already said that for real signals the row-column algorithm doubles the number of 1-D transforms for each dimension. Hence, we need three transforms in the 2-D case. Each transform has to pass N rows (columns) and its complexity is given according to table 10.3 by $N \operatorname{ld} N$ multiplications and $3/2 N \operatorname{ld} N$

additions ($5/2N \operatorname{ld} N$ operations). Hence, we obtain $3N^2 \operatorname{ld} N$ multiplications and $9/2N^2 \operatorname{ld} N$ additions for the whole spectrum ($15/2N^2 \operatorname{ld} N$ operations).

For quaternionic signals the row-column algorithm performs two times two 1-D FFTs. Since the signals are not real, the 1-D FFT itself needs twice the number of operations. That yields a complexity of $8N^2 \operatorname{ld} N$ multiplications and $12N^2 \operatorname{ld} N$ additions ($20N^2 \operatorname{ld} N$ operations).

If we use the isomorphism to calculate the Clifford spectrum of a signal, the complexity depends on the 2-D FFT algorithm (up to a quadratic additive complexity). Assuming that a 2-D FFT algorithm can be performed with $3/2N^2 \operatorname{ld} N$ multiplications and $(3/4+2)N^2 \operatorname{ld} N$ additions ($17/4N^2 \operatorname{ld} N$ operations)[125], an FHFT algorithm using the isomorphism has the same complexity if the signal is real.

If the signal is Clifford-valued, the complexity is four times the complexity of the algorithm for a real signal (up to a quadratic additive complexity for the combination of the four spectra). All the complexities calculated above are summarized in table 10.4.

Table 10.4. Complexities of the considered algorithms

algorithm	multiplications	additions	operations
row-column (real)	$3N^2 \operatorname{ld} N$	$9/2N^2 \operatorname{ld} N$	$15/2N^2 \operatorname{ld} N$
row-column (quat.)	$8N^2 \operatorname{ld} N$	$12N^2 \operatorname{ld} N$	$20N^2 \operatorname{ld} N$
isomorphism (real)	$\frac{3}{2}N^2 \operatorname{ld} N$	$\frac{11}{4}N^2 \operatorname{ld} N$	$\frac{17}{4}N^2 \operatorname{ld} N$
isomorphism (quat.)	$6N^2 \operatorname{ld} N$	$11N^2 \operatorname{ld} N$	$17N^2 \operatorname{ld} N$

Finally, we will roughly consider some complexities for the n-dimensional case. The row-column algorithm for the real case uses $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ 1-D FFTs N^{n-1} times. That yields a complexity of $(2^n - 1)N^n \operatorname{ld} N$ multiplications and $(2^n - 1)3/2N^n \operatorname{ld} N$ additions ($(2^n - 1)5/2N^n \operatorname{ld} N$ operations).

The row-column algorithm for Clifford-valued signals needs $n2^{n-1}$ 1-D FFTs (with double complexity) N^{n-1} times. Hence, we have a complexity of $n2^n N^n \operatorname{ld} N$ multiplications and $n2^n 3/2N^n \operatorname{ld} N$ additions ($n2^n 5/2N^n \operatorname{ld} N$ operations).

The algorithm using the isomorphism still has the same complexity as the complex nD FFT in the case of real signals and it has a 2^n times higher complexity in the case of Clifford-valued signals.

If we consider the memory complexity, it becomes obvious that the row-column algorithm applies FFTs only on 1-D sub-signals. Therefore, it works nearly independently on the dimension. If the signal does not fit into the main memory, the data is swapped as often as the coordinate for the 1-D FFT is changed (i.e. n times). Hence, the number of swapped data is nN^n. Note that

we provided implicitly, that one row always fits in the main memory, which is quite realistic.

For the n-D FFT (or FHFT) we have one crucial point in the recursion where the data begins to swap for each higher level. The number of the swapping levels s can be calculated from the size of the main memory M and the signal size N^n by the formula

$$s = \lceil \frac{1}{n} \operatorname{ld} \frac{N^n}{M} \rceil = \lceil \operatorname{ld} N - \frac{1}{n} \operatorname{ld} M \rceil = \operatorname{ld} N - \lfloor \operatorname{ld} \sqrt[n]{M} \rfloor. \tag{10.22}$$

The algorithm must swap $(\operatorname{ld} N - \lfloor \operatorname{ld} \sqrt[n]{M} \rfloor) N^n$ values. Since this formula is not as easy to understand as the one for the row-column algorithm, we will give an example.

Firstly, consider that we want to calculate the DQFT ($n = 2$). We know that each quaternion uses 32 bytes (four double floats). Assuming that the main memory consists of 64 megabyte, we obtain $M = 2^{21}$. Hence $\lfloor \operatorname{ld} \sqrt[2]{M} \rfloor = 10$. That means, if the image is greater than 1024×1024 the FHFT begins to swap. The number of swappings is linear with the power of two. The row-column algorithm swaps two times, if the image is greater that 1024×1024.

Now we have $n = 3$. We obtain $M = 2^{20}$ and $\lfloor \operatorname{ld} \sqrt[2]{M} \rfloor = 6$. Again, both algorithms begin to swap if the signal is greater than $64 \times 64 \times 64$. The row-column algorithm swaps three times, the FHFT algorithm needs a number of swappings linear with the power of two. Both examples are illustrated in figure 10.5. The x-axis indicates the exponent of the signal size and the y-axis indicates, how often the whole data is swapped.

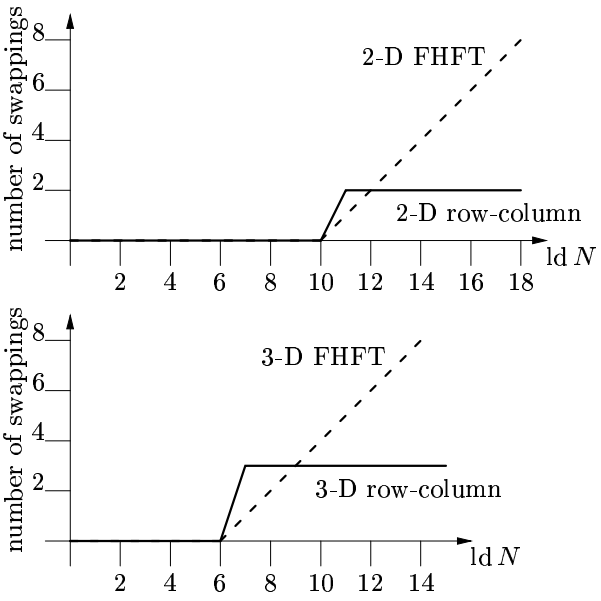

Fig. 10.5. The number of swapping operations for the 2-D and 3-D FHFT and row-column algorithm

Obviously, it depends on the size of the data which algorithm is the best choice.

10.5 Conclusion and Summary

We have considered several fast algorithms for multidimensional hypercomplex Fourier transforms. We can divide these algorithms into three classes: n-D decimation algorithms, n-D algorithms which use the complex FFTn, and row-column algorithms which apply 1-D FFT1s to each coordinate (separability). The last two algorithms use standard transforms and a simple mapping of the data is all to be done. Therefore, these algorithms are to be preferred if one wants to make only some experiments or if the dimension of the signals often changes because they can be implemented fast and easily.

Although the asymptotic complexity of the n-D decimation algorithms are the least and therefore these algorithms seem to be superior to the algorithms using the isomorphism or the row-column method, we cannot recommend using them, because their implementation is slower due to all the array accesses and the exhaustive length of the code. We ourselves recommend using the row-column method since the complex FFTn algorithms are mostly implemented in this way and one can merge the steps from 1-D FFTs to n-D FFT and from n-D FFT to n-D FHFT in one step. Additionally, it is possible to adapt the algorithm to signal sizes which differ for different coordinates.

For very large signals (i.e. greater than the actual memory size) the row-column method is superior as well, since the swapping is reduced to a minimum. Furthermore, it is easier to optimize the 1-D FFT algorithm than to speed up the n-D algorithms. These optimizations of the FFT1 automatically lead to an optimized n-D algorithm.

The most interesting result which we have presented is the fact that a theoretical algebraic result yields a practically optimal algorithm. The isomorphism of the n-fold tensor product and the 2^{n-1}-fold Cartesian product of the complex algebra leads to a decomposition technic for an algorithm. This result emphasizes the importance of deep mathematical knowledge for signal processing. Thus, geometric algebra has been shown to be a powerful embedding for multidimensional signal analysis.