Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Sparse fast Clifford Fourier transform[*]

Rui WANG[1], Yi-xuan ZHOU[1], Yan-liang JIN[1], Wen-ming CAO[†‡2,3]

([1]*School of Communication and Information Engineering, Shanghai University, Shanghai 200444, China*)

([2]*College of Information Engineering, Shenzhen University, Shenzhen 518060, China*)

([3]*Department of Electrical and Computer Engineering, University of Missouri, Columbia 65211, USA*)

[†]E-mail: wmcao@szu.edu.cn

Received Dec. 9, 2015; Revision accepted Aug. 23, 2016; Crosschecked Aug. 24, 2017

Abstract: The Clifford Fourier transform (CFT) can be applied to both vector and scalar fields. However, due to problems with big data, CFT is not efficient, because the algorithm is calculated in each semaphore. The sparse fast Fourier transform (sFFT) theory deals with the big data problem by using input data selectively. This has inspired us to create a new algorithm called sparse fast CFT (SFCFT), which can greatly improve the computing performance in scalar and vector fields. The experiments are implemented using the scalar field and grayscale and color images, and the results are compared with those using FFT, CFT, and sFFT. The results demonstrate that SFCFT can effectively improve the performance of multivector signal processing.

Key words: Sparse fast Fourier transform (sFFT); Clifford Fourier transform (CFT); Sparse fast Clifford Fourier transform (SFCFT); Clifford algebra

http://dx.doi.org/10.1631/FITEE.1500452 **CLC number:** TP391

1 Introduction

Methods have been used to analyze and visualize unstructured vector field data. There are basically two approaches: extracting features to determine the dataset and visualizing an entire dataset. As dataset sizes increase, feature extraction becomes increasingly important. A method called 'Clifford Fourier transform (CFT)' was proposed to apply the Fourier transform to vector fields (Ebling and Scheuermann, 2005; Schlemmer *et al.*, 2005; Hitzer and Mawardi, 2008). Hitzer and Sangwine (2013) provided an overview of the modern development of CFT and wavelet transformations. In recent years, a detailed algebraic characterization of the continuous manifolds of square roots of −1 has been established in all real Clifford algebras (Hitzer *et al.*, 2013). This is why CFT uses multivector square roots of –1 instead of the complex imaginary unit. Based on this, a general CFT was introduced (Hitzer, 2013). CFT has already become one of the most significant steps in image processing and analysis (Cao and Feng, 2010; Xu *et al.*, 2011; Cao *et al.*, 2013; Wang *et al.*, 2013), network analysis (Li, 2005; Xie *et al.*, 2008; Wang *et al.*, 2015), color edge detection (Evans *et al.*, 2000; Mishra *et al.*, 2015), and image registration (Batard *et al.*, 2010). CFT allows for frequency analysis of vector fields and the behavior of vector-valued filters. In the frequency space, CFT can transform vectors into general Clifford algebra multivectors. Many basic vector-valued patterns (such as source, sink, saddle points, and potential vortices) can be described by a certain number of multivectors in the frequency space (Ebling and Scheuermann, 2005).

On the other hand, explosive growth in data makes big data problems a cause for concern. Some solutions have already been proposed, such as using fast approximate correlation (Mueen *et al.*, 2010), sparse matrices (Gilbert and Indyk, 2010), and the sparse recovery system (Porat and Strauss, 2012).

[‡] Corresponding author

[*] Project supported by the National Natural Science Foundation of China (Nos. 61301027, 61375015, and 11274226)

ORCID: Yi-xuan ZHOU, http://orcid.org/0000-0002-5008-5802

Iwen (2010) presented a method to estimate the Fourier representation for sparse signals. However, Hassanieh *et al.* (2012b) developed a new approach called 'sparse fast Fourier transform (sFFT)' to process big data, especially sparse data. sFFT deals with big data problems by using only a small subset of the input data to compute a compressed Fourier transform. As a result, sFFT is much faster than the fast Fourier transform (FFT). sFFT has already been applied to areas such as GHz-wide sensing and decoding (Hassanieh *et al.*, 2014), light field reconstruction (Shi *et al.*, 2014), and the Global Positioning System (GPS) (Hassanieh *et al.*, 2012a). Our work is to combine sFFT with CFT to operate on multivector signals. sFFT can choose 'large' coefficients for calculation. Combining it with CFT reduces useless or remote data, and thus the Fourier transform of multivector signals can be computed more efficiently (Schumacher and Puschel, 2014).

2 Related work

2.1 Clifford algebra

Clifford algebra provides a powerful computing framework without using coordinate information. Hestenes and Sobczyk (1984) used it as an efficient and versatile computational tool. It integrates vector algebra, matrix algebra, and complex numbers algebra into a coherent mathematical language which not only reduces the computation complexity, but also improves the computation efficiency (Hestenes, 1999).

Let $\mathbb{R}^d$ be a d-dimensional Euclidean vector space and $\boldsymbol{G}_d$ a 2^d-dimensional real Clifford algebra over $\mathbb{R}$. One obtains $\boldsymbol{G}_3$ by using the rules of 3D Clifford algebra with the given basis $\{\mathbf{1}, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3, \boldsymbol{e}_1\boldsymbol{e}_2, \boldsymbol{e}_1\boldsymbol{e}_3, \boldsymbol{e}_2\boldsymbol{e}_3, \boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3\}$:

$$\mathbf{1}\boldsymbol{e}_j = \boldsymbol{e}_j,\ j=1,2,3, \tag{1}$$

$$\boldsymbol{e}_j\boldsymbol{e}_j = \mathbf{1},\ j=1,2,3, \tag{2}$$

$$\boldsymbol{e}_j\boldsymbol{e}_k = -\boldsymbol{e}_k\boldsymbol{e}_j j,\ k=1,2,3,\ j\neq k. \tag{3}$$

Note that the multiplication of Clifford algebra is not commutative.

We define an arbitrary multivector $\boldsymbol{P}$ as

$$\boldsymbol{P} = \boldsymbol{\alpha} + \boldsymbol{a} + \mathrm{i}_3(\boldsymbol{b}+\boldsymbol{\beta}), \tag{4}$$

with $\boldsymbol{\alpha}, \boldsymbol{\beta}\in\mathbb{R}$, $\boldsymbol{a}, \boldsymbol{b}\in\mathbb{R}^3$, $\mathrm{i}_3=\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3$, and $(\mathrm{i}_3)^2=-1$. We call such components, $\boldsymbol{\alpha}$, $\boldsymbol{a}$, $\mathrm{i}_3\boldsymbol{b}$, and $\mathrm{i}_3\boldsymbol{\beta}$, as blades of Clifford algebra that are generalizations of the concept of scalars and vectors.

We define the grade projector $\langle\cdot\rangle_j$: $\boldsymbol{G}_3\rightarrow\boldsymbol{G}_3$ as

$$\langle\boldsymbol{P}\rangle_0 = \boldsymbol{\alpha},\ \langle\boldsymbol{P}\rangle_1 = \boldsymbol{a}, \tag{5}$$

$$\langle\boldsymbol{P}\rangle_2 = \mathrm{i}_3\boldsymbol{b},\ \langle\boldsymbol{P}\rangle_3 = \mathrm{i}_3\boldsymbol{\beta}. \tag{6}$$

The Clifford product of two vectors $\boldsymbol{a}$, $\boldsymbol{b}\in\mathbb{R}^3$ results in

$$\boldsymbol{ab} = \langle\boldsymbol{a},\boldsymbol{b}\rangle + \boldsymbol{a}\wedge\boldsymbol{b}, \tag{7}$$

where $\langle\cdot,\cdot\rangle$ is the inner product and $\cdot\wedge\cdot$ is the outer product.

As shown in Eq. (4), a multivector is a linear combination of components in different grades. Thus, by using linearity, the Clifford product of multivectors involves Clifford products of component blades. The appendix in Ebling and Scheuermann (2003) contains an example of multiplying multivectors in 2D, and Table 2 on page 344 in Hitzer (2012) shows the multiplication table for $\boldsymbol{G}_3$.

Further, we have

$$\langle\boldsymbol{ab}\rangle_0 = \langle\boldsymbol{a},\boldsymbol{b}\rangle = \|\boldsymbol{a}\|\|\boldsymbol{b}\|\cos\omega, \tag{8}$$

$$\|\langle\boldsymbol{ab}\rangle_2\| = \|\boldsymbol{a}\wedge\boldsymbol{b}\| = \|\boldsymbol{a}\|\|\boldsymbol{b}\|\sin\omega, \tag{9}$$

where the usual norm for vectors, $\|\langle\boldsymbol{ab}\rangle_2\|=\sqrt{-(\langle\boldsymbol{ab}\rangle_2)^2}$, and ω is the angle between $\boldsymbol{a}$ and $\boldsymbol{b}$.

2.2 Clifford convolution

The convolution theorem states that a convolution in the time domain is equal to a multiplication in the frequency space. Therefore, convolution becomes an important tool in signal processing. Bujack *et al.* (2015) presented a convolution theorem for general CFTs with separable mappings being orthogonal, separable, and linear. It can be interpreted as a mutual commutation or anticommutation among the functions.

Let $\boldsymbol{U}$, $\boldsymbol{V}$: $\mathbb{R}^d\to\boldsymbol{G}_d$ (d=2, 3) be two multivector fields. Then the Clifford convolution can be defined as

$$(\boldsymbol{V}*\boldsymbol{U})(\boldsymbol{x})=\int_{\mathbb{R}^d}\boldsymbol{V}(\boldsymbol{\xi})\boldsymbol{U}(\boldsymbol{x}-\boldsymbol{\xi})\,|\,\mathrm{d}\boldsymbol{\xi}\,|, \tag{10}$$

where $\boldsymbol{x}\in\boldsymbol{V}$ and $\boldsymbol{\zeta}\in\boldsymbol{U}$.

It can be seen that the Clifford convolution is a conventional convolution when both fields are scalar. Further, let $\boldsymbol{U}$ be a scalar field and $\boldsymbol{V}$ a vector field. The Clifford convolution then allows a scalar multiplication and vector smoothing model.

If both $\boldsymbol{V}$ and $\boldsymbol{U}$ are vector fields, we obtain the following relationship:

$$\boldsymbol{V}(\boldsymbol{\xi})\boldsymbol{U}(\boldsymbol{x}-\boldsymbol{\xi})=\left\langle\boldsymbol{V}(\xi),(\boldsymbol{x}-\boldsymbol{\xi})\right\rangle+\boldsymbol{V}(\boldsymbol{\xi})\wedge\boldsymbol{U}(\boldsymbol{x}-\boldsymbol{\xi}). \tag{11}$$

It means that the convolution in the multivector field contains additional information, as the scalar part of the convolution result is a conventional convolution. Thus, Clifford convolution can be used to better analyze the vector field data.

2.3 Clifford Fourier transform

CFT allows a transformation from the position space to the frequency space, allowing signals to be analyzed in the frequency space. This makes it feasible to use such features as phases and frequencies of the signals. Further, the convolution theorem allows better filter response when a signal is analyzed in the frequency space (Ebling and Scheuermann, 2005). Similarly, Clifford convolution can be transformed into the frequency space. Thus, CFT can be developed as an extension of the usual Fourier transform for vector fields (Schlemmer *et al.*, 2005; Reich and Scheuermann, 2010).

2.3.1 Clifford Fourier transform in the 2D space

Ebling and Scheuermann (2005) proposed the CFT on both scalar and multi-dimensional fields, initiating a novel approach to analyzing vector fields. Subsequently, the generalization of the Fourier transform in Clifford geometric algebra has been presented (Hitzer and Mawardi, 2008; Hitzer and Sangwine, 2013).

CFT for multivector-valued functions F: $\mathbb{R}^2\to\boldsymbol{G}_2$ can be defined as

$$\mathscr{F}\{F\}(\boldsymbol{u})=\int_{\mathbb{R}^2}F(\boldsymbol{x})\exp(-2\pi\mathrm{i}_2\langle\boldsymbol{x},\boldsymbol{u}\rangle)\mathrm{d}^2\boldsymbol{x}, \tag{12}$$

where vectors $\boldsymbol{x}$, $\boldsymbol{u}\in\mathbb{R}^2$.

Based on Clifford algebra, a 4D multivector field can be written as

$$\begin{aligned}F(\boldsymbol{x})&=F_0(\boldsymbol{x})+F_1(\boldsymbol{x})\boldsymbol{e}_1+F_2(\boldsymbol{x})\boldsymbol{e}_2+F_{12}(\boldsymbol{x})\boldsymbol{e}_{12}\\&=F_0(\boldsymbol{x})+F_1(\boldsymbol{x})\boldsymbol{e}_1+F_2(\boldsymbol{x})\boldsymbol{e}_1\mathrm{i}_2+F_{12}(\boldsymbol{x})\mathrm{i}_2\\&=\mathbf{1}(F_0(\boldsymbol{x})+F_{12}(\boldsymbol{x})\mathrm{i}_2)+\boldsymbol{e}_1(F_1(\boldsymbol{x})+F_2(\boldsymbol{x})\mathrm{i}_2).\end{aligned} \tag{13}$$

Then, considering the linearity of CFT, one obtains

$$\begin{aligned}\mathscr{F}\{F\}(\boldsymbol{u})=\mathbf{1}(\mathscr{F}\{F_0(\boldsymbol{x})+F_{12}(\boldsymbol{x})\mathrm{i}_2\}(\boldsymbol{u}))\\+\boldsymbol{e}_1(\mathscr{F}\{F_1(\boldsymbol{x})+F_2(\boldsymbol{x})\mathrm{i}_2\}(\boldsymbol{u})).\end{aligned} \tag{14}$$

It can be seen that 2D CFT is the linear combination of two traditional complex Fourier transforms.

2.3.2 Clifford Fourier transform in the 3D space

CFT for multivector-valued functions F: $\mathbb{R}^3\to\boldsymbol{G}_3$ can be defined as

$$\mathscr{F}\{F\}(\boldsymbol{u})=\int_{\mathbb{R}^3}F(\boldsymbol{x})\exp(-2\pi\mathrm{i}_3\langle\boldsymbol{x},\boldsymbol{u}\rangle)\mathrm{d}^3\boldsymbol{x}, \tag{15}$$

where vectors $\boldsymbol{x}$, $\boldsymbol{u}\in\mathbb{R}^3$.

Based on Clifford algebra, an 8D multivector field can be written as

$$\begin{aligned}F&=F_0+F_1\boldsymbol{e}_1+F_2\boldsymbol{e}_2+F_3\boldsymbol{e}_3+F_{23}\boldsymbol{e}_{23}\\&\quad+F_{31}\boldsymbol{e}_{31}+F_{12}\boldsymbol{e}_{12}+F_{123}\boldsymbol{e}_{123}\\&=F_0+F_1\boldsymbol{e}_1+F_2\boldsymbol{e}_2+F_3\boldsymbol{e}_3+F_{23}\mathrm{i}_3\boldsymbol{e}_1\\&\quad+F_{31}\mathrm{i}_3\boldsymbol{e}_2+F_{12}\mathrm{i}_3\boldsymbol{e}_3+F_{123}\mathrm{i}_3,\end{aligned} \tag{16}$$

which can be seen as the decomposition of an 8D multivector field into four complex signals. Each complex signal is transformed separately by the standard Fourier transform. Then, considering the linearity of CFT, one obtains

$$\begin{aligned}\mathscr{F}\{F\}(\boldsymbol{u}) = &[\mathscr{F}\{F_0(\boldsymbol{x}) + F_{123}(\boldsymbol{x})\mathrm{i}_3\}(\boldsymbol{u})]\mathbf{1}\\ &+ [\mathscr{F}\{F_1(\boldsymbol{x}) + F_{23}(\boldsymbol{x})\mathrm{i}_3\}(\boldsymbol{u})]\boldsymbol{e}_1\\ &+ [\mathscr{F}\{F_2(\boldsymbol{x}) + F_{31}(\boldsymbol{x})\mathrm{i}_3\}(\boldsymbol{u})]\boldsymbol{e}_2\\ &+ [\mathscr{F}\{F_3(\boldsymbol{x}) + F_{12}(\boldsymbol{x})\mathrm{i}_3\}(\boldsymbol{u})]\boldsymbol{e}_3.\end{aligned} \quad (17)$$

This is useful as one can divide multivector algebra into four complex components. Also, the usual Fourier transform can be calculated separately for each direction $\{\mathbf{1}, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\}$.

3 Sparse fast Clifford Fourier transform

3.1 Sparse fast Fourier transform

Algorithms for computing the Fourier transform are inefficient because they take time proportional to the output size, and most of the Fourier signal coefficients are small or equal to zero in many applications. For instance, a normal 16×16 block in a video frame is sparse, since it has 228 negligible coefficients on average. It means that nearly 90% of the data are useless. Audio and image data are sparse as well. Compression schemes, such as JPEG and MPEG, have been provided to use this sparsity. The sFFT algorithm uses the sparsity of the signal spectrum and computes an approximated or compressed version of the Fourier transform. The algorithm uses only a small subset of the input data and runs in time proportional to the sparsity or desirable compression. It can be faster than that in time proportional to the signal length (Hassanieh *et al.*, 2012c).

There are three stages of sFFT: identification of frequencies with large Fourier coefficients, accurate estimation of the Fourier coefficients of the frequencies identified in the first stage, and subtraction of the contribution of the partial Fourier representation computed in the first two stages (Gilbert *et al.*, 2014). Generally, repetitions of the three stages are guaranteed to concentrate a substantial portion of the energy with a high probability. In the first stage, frequencies are sampled randomly to obtain permuted Fourier coefficients. Then filters are used to divide the permuted coefficients into various frequency bands. In the final stage, the energy is estimated in each frequency band.

For the complex affine N-space $\mathbb{C}^N$, the Fourier component $\hat{\boldsymbol{f}} \in \mathbb{C}^N$ can be defined as

$$\hat{f}_\omega = \frac{1}{N}\sum_{j=0}^{N-1} f_j \exp(-2\pi\mathrm{i}\omega j / N), \quad (18)$$

where $0 \le \omega$ and $j < N$.

Equivalently, one can simply obtain

$$f_j = \sum_{\omega=0}^{N-1} \hat{f}_\omega \exp(2\pi\mathrm{i}\omega j / N). \quad (19)$$

Two basic Fourier properties are used for randomly permuting $\boldsymbol{f} \in \mathbb{C}^N$ and then $\hat{\boldsymbol{f}}$. One is the scaling property, stating that for $a_j = f_{cj}$, we have $\hat{a}_j = \hat{f}_{c^{-1}j}$ (where c^{-1} is the inverse of (c mod N)); the other is the modulation property, stating that for $a_j = \exp(2\pi\mathrm{i}bj/N) \cdot f_j$, we have $\hat{a}_j = \hat{f}_{j-b}$. Then we randomly select two integers b, $c \in [0, N]$ and define $\boldsymbol{a} \in \mathbb{C}^N$ as

$$a_j = \exp(2\pi\mathrm{i}bj / N) \cdot f_{cj}. \quad (20)$$

Note that for $j \in [0, N-1]$, $\hat{\boldsymbol{a}}$ is a permutation of $\hat{\boldsymbol{f}}$, as entry $\hat{f}_\omega$ appears in $\hat{\boldsymbol{a}}$ as $(\omega c + b)$ mod N.

In the simplest case, the Fourier coefficients $\hat{f}_\omega$ can be computed for each ω identified in the first stage by using L independent and uniformly distributed random samples f_l (L should be far smaller than N) and the estimator:

$$\hat{f}'_\omega = \frac{1}{L} \cdot \sum_{l=1}^{L} f_l \exp(-2\pi\mathrm{i}\omega l / N), \quad (21)$$

It can be seen that $\hat{f}_\omega{}'$ is an unbiased estimator for $\hat{f}_\omega$. Therefore, high precision is guaranteed, as the estimator can approximate $\hat{f}_\omega$ with a high probability.

Assume that

$$\left\{\hat{f}'_{\omega_m} \middle| m = 1, 2, \cdots, N\right\} \subset \mathbb{C}, \quad (22)$$

which is the approximated sparse Fourier transform computed in the previous stage of the current repetition. Here, ω_1, ω_2, …, ω_N are the frequencies that have been identified in the first stage, while $\hat{f}'_{\omega_1}, \hat{f}'_{\omega_2}, ..., \hat{f}'_{\omega_N}$ are the estimators whose Fourier coefficients are discovered in the second stage. In future iterations of all the stages, each sampled entry of $\boldsymbol{f}$ and f_j can be replaced with

$$f_j - \sum_{m=1}^{N} \hat{f}'_{\omega_m} \exp(2\pi \mathrm{i} \omega_m j / N). \tag{23}$$

In most cases, the entries of $\boldsymbol{f}$ used in each iteration can be predetermined, and then be used to update themselves all. These 'updated samples' are used in subsequent repetitions of the three stages (Gilbert *et al.*, 2014).

3.2 Sparse fast Clifford Fourier transform

Assume that a signal $\boldsymbol{f}$ has k nonzero coefficients. We can regard the signal as k-sparse. More often, we calculate a signal's k largest coefficients; thus, we call it 'approximately k-sparse'. If its CFT is $\hat{\boldsymbol{f}}$, then an approximation $\hat{\boldsymbol{f}}$ to $\hat{\boldsymbol{f}}'$, of an algorithm is required to be output to meet the following ℓ_2/ℓ_2 guarantee:

$$\left\|\hat{\boldsymbol{f}} - \hat{\boldsymbol{f}}'\right\|_2 \le C \min_{k\text{-sparse } \boldsymbol{y}} \left\|\hat{\boldsymbol{f}} - \boldsymbol{y}\right\|_2, \tag{24}$$

where $\boldsymbol{y}$ is a k-sparse standard reference. Note that the minimization is over k-sparse signals, and C is an approximation factor. Further, if k is small enough, the output of the sparse fast Clifford Fourier transform (SFCFT) can be expressed succinctly. The key to SFCFT is to filter useless and negligible coefficients, and then implement CFT.

Fig. 1 shows an example of normalizing a k-sparse vector field. Assume that a vector field (Fig. 1b) is sampled over a given cube (Fig. 1a). For each face of the cube, there is a vector $\boldsymbol{a}_i$ (i=0, 1, …) on each sample point. Place the tail of each sample vector at the origin. Then normalize these vectors to form trivectors $a_i / 6 \wedge a_{i+1} \wedge a_{i+2}$. Sum the trivectors over all six faces of the cube (Fig. 1c). Note that such a trivector is not a triple of vectors, but a volume in space. The magnitude of the resulting trivector will be the volume of the sphere approximately. Fig. 1d shows the estimated k-sparse transformed vector field with an equal result. The normalization is carried out using the following steps:

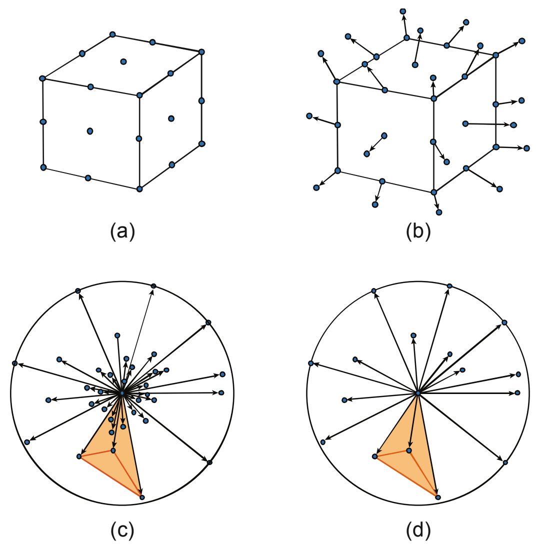

Fig. 1 Example of normalizing a *k*-sparse vector field: (a) the given cube; (b) normalized vector field; (c) trivectors over all six faces of the cube; (d) estimated *k*-sparse transformed vector field with an equal result

1. Window function

Definition 1 (Window function) For an n-dimensional signal, a constant ϵ>0, and a parameter δ>0, we define the window function $w(\epsilon, \delta, \omega)$ to be a symmetric vector $\boldsymbol{F}\in\mathbb{R}^n$ with $\omega\in[-\pi, \pi]$. The set of nonzero coordinates of vector $\boldsymbol{F}$ is nzc($\boldsymbol{F}$)∈[$-\omega$/2, ω/2], and $\hat{F}_0 = 1$, and $\hat{F}_i > 0$, for $i\in[-\epsilon n, \epsilon n]$.

For each ϵ and δ, a standard window function (ϵ, δ, $O(\log(1/\delta)/\epsilon)$) exists (Smith, 2011), since one can obtain a standard window function through taking a Gaussian with standard deviation $\Theta\left(\sqrt{\log(1/\delta)}/\epsilon\right)$ and truncating it at $\omega=O(\log(1/\delta)/\epsilon)$.

Definition 1 means that the window function w works like a filter, allowing us to concentrate on a subset of the Clifford Fourier coefficients.

2. Spectral permutation: This step allows us to permute the Clifford Fourier spectrum by permuting the time domain as follows:

Lemma 1 Given a constant vector $\boldsymbol{a}\in\mathbb{R}^n$, a random integer σ, and an integer $\tau\in[n]$, we define a transform $T_{\sigma,\tau}$: $(T_{\sigma,\tau}\boldsymbol{a})_i=\boldsymbol{a}_{\sigma i+\tau}$, which refers to the permutation of the components of $\boldsymbol{a}$. Then, $\widehat{(T_{\sigma,\tau}\boldsymbol{a})}_{\sigma i} = \hat{a}_{i,j}\omega^{-\tau i}(T_{\sigma,\tau}\boldsymbol{a})$.

Proof $\forall m$, we have

$$\widehat{(T_{\sigma,\tau}\boldsymbol{a})}_{\sigma i \sigma i} = \sum_{j=1}^{n} a_{\sigma j+\tau}\omega^{mj} = \sum_{j=1}^{n} a_j \omega^{m(j-\tau)\sigma^{-1}}$$
$$= \hat{a}_{m\sigma^{-1}}\omega^{-\tau m\sigma^{-1}}.$$

Varying the permutation can vary the set of coefficients binned to a bucket. Note that for simplicity, our algorithm is analyzed on the condition that n is a power of two.

3. Subsampled CFT: suppose a vector $\boldsymbol{a}\in\mathbb{C}^n$ and a parameter B dividing n, one obtains

$$\hat{y}_i = \hat{a}_{i(n/B)}. \tag{25}$$

Lemma 2 $\hat{\boldsymbol{y}}$ is the B-dimensional CFT of $y_i = \sum_{i=0}^{n/B-1} a_i$.

Proof $\hat{a}_{i(n/B)} = \sum_{j=0}^{n-1} a_j \omega^{ij(n/B)} = \sum_{t=0}^{B-1}\sum_{j=0}^{n/B-1} a_{Bj+m}\omega^{i(Bj+m)n/B}$

$$= \sum_{m=0}^{B-1}\sum_{j=0}^{n/B-1} a_{Bj+m}\omega^{imn/B} = \sum_{m=0}^{B-1} y_{\boldsymbol{a}}\omega^{imn/B} = \hat{y}_i.$$

4. Location loops

Definition 2 (Hash function**)** Given a parameter B that divides n, we define a hash function h_σ: $[n]\rightarrow[B]$ by $h_\sigma(i)$=round($\sigma iB/n$) and an 'offset' o_σ: $[n]\rightarrow[-n/(2B), n/(2B)]$ by $o_\sigma(i)=\sigma i-h_\sigma(i)(n/B)$, where $[n]=\{0, 1, \ldots, n-1\}$.

The location loops begin with a given parameter d and then output a parameter J that controls to find the coordinate of each large coefficient, and a set $\boldsymbol{I}\subset[n]$ of coordinates that contains each large coefficient.

5. Estimation loops: given a set $\boldsymbol{I}\subset[n]$, we define G as a window function: $w(1/B, \delta, \omega)$ and $\boldsymbol{y}=G\cdot(T_{\sigma,\tau}\boldsymbol{a})$; thus, for $i\in\boldsymbol{I}$, estimate $\hat{\boldsymbol{a}}$ as $\hat{a}'_i = \hat{z}_{h_{\sigma(i)}}\omega^{\tau i}\big/\hat{G}_{o_{\sigma(i)}}$, where $\hat{z}_i = \hat{y}_{i(n/B)}$. This estimates each coordinate with a good probability.

Fig. 2 shows an example of loops of the algorithm when the input is sparse. Note that for simplicity, the process is analyzed only in the scalar field. The algorithm outputs $T_{\sigma,\tau}\boldsymbol{a}$ and obtains the spectrum permutation by permuting the Clifford algebra samples in the time domain (Fig. 2a). The algorithm computes signal $\boldsymbol{f}=G\cdot T_{\sigma,\tau}\,\boldsymbol{a}$ in the time domain

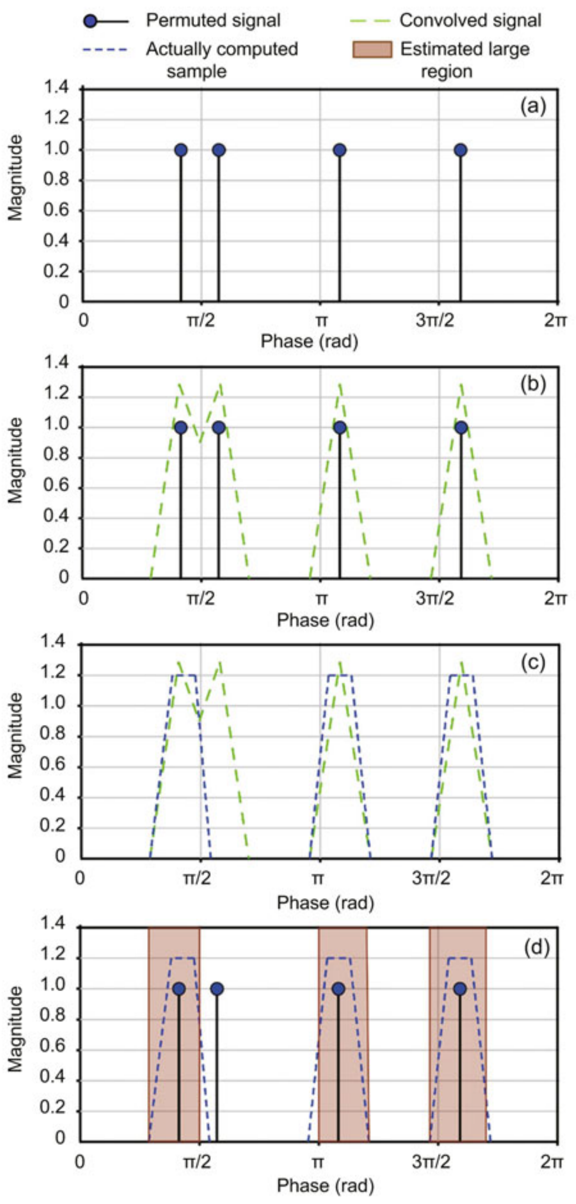

Fig. 2 Results of estimating large regions in the scalar field: (a) permuted signal; (b) convolved signal; (c) actually computed samples; (d) estimated large regions

(Fig. 2b). It can be seen that the spectrum of $\boldsymbol{f}$ is large around the large coordinates. The algorithm computes the subsampling of $\hat{\boldsymbol{f}}$ (Fig. 2c). Finally, the algorithm outputs the coordinates of each large coefficient (Fig. 2d). Note that there is a 'hash collision' when two coordinates are too close in the permutation. It results in missing the second coordinate from the left one.

3.3 The proposed algorithm

We use Clifford algebra to divide a multivector into a Clifford basis $\{\mathbf{1}, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3, \ldots\}$ and transform a Clifford basis into complex signals, as in Eqs. (13) and (16). Then we calculate the Fourier transform of each complex signal separately. The Fourier transform is usually dominated by a small number of 'peaks', meaning that it is sparse. The primary algorithm works by first 'locating' a set of loops that contain most of the peaks and then 'estimating' $\hat{\boldsymbol{f}}$ to obtain $\hat{z}$. Thus, we can calculate the k-sparse Fourier transform as mentioned before. With the property of linearity, we add up each transform to obtain the final transform. The proposed method is provided in Algorithm 1.

Algorithm 1 Sparse fast Clifford Fourier transform

```
Input: a multivector signal
Output: result of SFCFT
1. Extract the Clifford basis components of the multivector
2. Reconstruct the components to obtain vectors
   $\begin{pmatrix} [F_0(x)+F_{123}(x)i_3]\mathbf{1} \\ [F_1(x)+F_{23}(x)i_3]\boldsymbol{e}_1 \\ [F_2(x)+F_{31}(x)i_3]\boldsymbol{e}_2 \\ [F_3(x)+F_{12}(x)i_3]\boldsymbol{e}_3 \end{pmatrix}$
   // For each vector, it is appropriate to use sFFT,
   // as it is sparse
3. Run a number of location loops, returning $L$ sets of coordinates $\boldsymbol{I}_1, \boldsymbol{I}_2, \ldots, \boldsymbol{I}_L$
4. Randomly choose a $\sigma$ invertible mod $n$ and $\tau\in[n]$, and permute input vector $\boldsymbol{f}$ with permutation $T_{\sigma,\tau}$: $(T_{\sigma,\tau}\boldsymbol{f})_j=\boldsymbol{f}_{\sigma j+\tau}$
5. Use a flat window function $G$, and compute the filtered and permuted vector $\boldsymbol{y}=G\cdot(T_{\sigma,\tau}\boldsymbol{f})$
6. With $B$ dividing $n$, compute $\hat{z}=\hat{y}_{j(n/B)}$ for $j\in[B]$
7. Keep only the $d\cdot k$ coordinates of the maximum magnitude in $\hat{z}$
8. Reverse steps 5–7 as a hash function $h_\sigma$: $[n]\rightarrow[B]$
9. Count the number of occurrences of each coordinate $j$ found, i.e., $s_j=|\{r|j\in\boldsymbol{I}_r\}|$
10. Keep only the coordinates that occur in at least half of the location loops:
    $\boldsymbol{I}'=\{j\in\boldsymbol{I}_1\cup\boldsymbol{I}_2\cup\ldots\cup\boldsymbol{I}_L|s_j>L/2\}$
11. Run a number of estimation loops on $\boldsymbol{I}'$, and return sets of frequency coefficients $\hat{f}_j^r$
    // These loops work similarly to location loops in
    // steps 4–6 and the next step is: given a set of coordinates
    // $\boldsymbol{I}$, estimate $\hat{f}_j$ as $\hat{f}_j'=\hat{z}_{h\sigma(j)}\omega^{\tau j}/\hat{G}_{\sigma(j)}$
12. Take median real and imaginary components separately and estimate each frequency coefficient $\hat{f}_j$ as
    $\hat{f}_j'=\text{median}\{f_j^r \mid r\in\{1,2,\cdots,L\}\}$
13. Based on the linearity of CFT, add up $\hat{f}_j$ to obtain the output
```

4 Experiment analysis

In this section, we compare the SFCFT algorithm to several existing FFT implementations and demonstrate how SFCFT provides a better computational performance when processing multivector signals. We evaluate SFCFT performance in comparison to the fastest Fourier transform in the West (FFTW) (Frigo and Johnson, 2005), sFFT1.0, sFFT2.0 (Hassanieh *et al.*, 2012c), and FCFT (Schlemmer *et al.*, 2005) in the scalar field. FFTW is the fastest public implementation for computing FFT. sFFT is the implementation for computing sFFT. Compared with sFFT1.0, sFFT2.0 adds a heuristic to improve the runtime. FCFT is also a fast algorithm for computing CFT. We fix the signal size and signal sparsity, and present the runtime and robustness of the compared algorithms in Figs. 3–5. In higher-dimensional fields, we implement the algorithms on processing multispectral images and present the runtime and spectrum map in Figs. 6 and 7. For accuracy, each experiment is run more than 5000 times. Each experiment is implemented on a 2.40 GHz Core™ i5 processor running Ubuntu 15.01 with a 4 GB RAM.

4.1 Performance in the scalar field

4.1.1 Runtime vs. signal size

Here, the signal sparsity parameter is fixed to k=50, and we compute the runtime of each algorithm for five different signal sizes: 2^{16}, 2^{17}, 2^{18}, 2^{19}, and 2^{20}. Table 1 summarizes the runtimes for processing various signal sizes using SFCFT, sFFT1.0, sFFT2.0, FCFT, and FFTW. Fig. 3 shows comparisons among these algorithms.

As expected, the runtime of each algorithm is approximately linear (Fig. 3). However, Fig. 3 demonstrates that for signal size n>2^{16}, SFCFT is faster than FFTW and FCFT when they have to recover exactly 50 nonzero coefficients. Thus, we can see that the slopes of the lines for FFTW and FCFT are larger than those for sFFT1.0, sFFT2.0, and SFCFT. It means that FFTW and FCFT do not efficiently process signals with large sizes. SFCFT is slower than sFFT1.0 and sFFT2.0, because SFCFT spends a certain amount of time for Clifford algebra operations.

Table 1 Runtimes of each algorithm at different signal sizes

Signal size	Runtime (ms)				
	SFCFT	sFFT2.0	sFFT1.0	FFTW	FCFT
2^{16}	0.4031	0.3471	0.3615	0.2028	0.8804
2^{17}	0.6234	0.5218	0.5804	0.4405	1.0615
2^{18}	0.8098	0.7048	0.7595	1.1626	1.6095
2^{19}	1.5343	0.9578	1.2825	2.1543	2.8825
2^{20}	3.3743	1.8775	2.5088	4.5130	5.5088

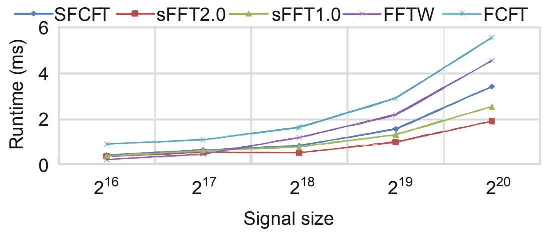

Fig. 3 Runtimes of the compared algorithms for different signal sizes

4.1.2 Runtime vs. sparsity (k)

In this experiment, the signal size is fixed to $n=2^{20}$, and the runtime of each algorithm is calculated. We choose sparsity parameter k from 50 to 500. The experiment is repeated 50 times for each value of k. Table 2 summarizes the runtimes for processing various signal sparsities using SFCFT, sFFT1.0, sFFT2.0, FCFT, and FFTW. Fig. 4 illustrates the comparison among these algorithms.

Fig. 4 shows that the runtime of each algorithm is approximately linear. Note that FFTW and FCFT are independent of sparsity k. It can be seen that when a sparse approximation of the Fourier transform is available, SFCFT, sFFT1.0, and sFFT2.0 greatly improve the runtime performances and extend the range of applications. Further, SFCFT, sFFT1.0, and sFFT2.0 are faster than FFTW and FCFT when sparsity parameter k is small enough (Fig. 4).

Table 2 Runtimes of each algorithm with different sparsity parameters

k	Runtime (ms)				
	SFCFT	sFFT2.0	sFFT1.0	FFTW	FCFT
50	3.3743	1.8875	2.1088	4.5130	5.7505
100	3.7175	2.9301	3.3965	4.5130	5.7505
200	4.6654	3.8355	4.2505	4.5130	5.7505
500	6.5449	5.3826	5.8371	4.5130	5.7505

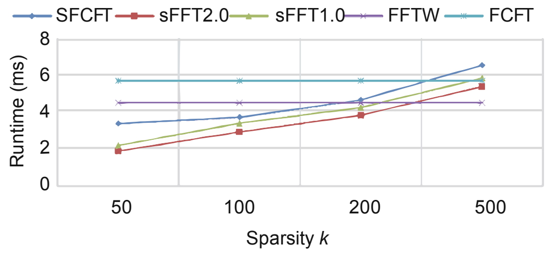

Fig. 4 Runtimes of the compared algorithms for different sparsity parameters

4.1.3 Robustness to noise

Here, we choose signal size $n=2^{20}$ and sparsity parameter $k=50$. For robustness, we add Gaussian noise to signals and run each experiment with different signal-to-noise ratios (SNRs). SNR is changed by changing the Gaussian noise. For each variance of the Gaussian noise, we regenerate new cases of noises and signals to run each experiment 5000 times. For each run, we compute the error metric as the average error between the best k-sparse approximation of $\hat{x}$ and the output approximation $\hat{x}'$ which is restricted to its k largest entries. This allows us to check whether our SFCFT can improve its performance in terms of runtime without reducing its robustness to noise.

The average errors per entry for SFCFT, sFFT1.0, sFFT 2.0, FCFT, and FFTW are shown in Fig. 5. It can be seen that the performances of all these algorithms are stable under noise, and SFCFT is more robust to noise than FFTW and FCFT.

4.2 Performance using 2D grayscale images

In this experiment, we apply our algorithm in 2D grayscale image Fourier transform and compare it with FFT and CFT. We choose a 'Lena' grayscale image (512×512) to evaluate the performance. Results show that our SFCFT is the fastest, since SFCFT computes data directly in the spatial domain first, and then computes the effective coefficients.

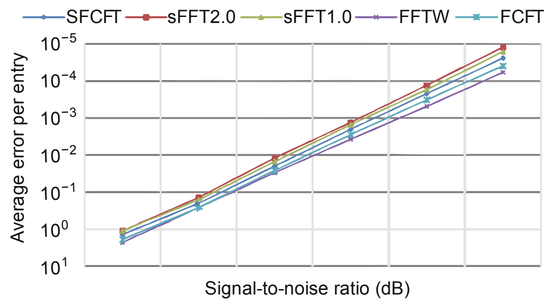

Fig. 5 Robustness to noise of the compared algorithms

Further, Figs. 6b–6d show that SFCFT outputs a clearer spectrum map than FFT and CFT. It means that SFCFT presents more image data information. As mentioned before, compared with the traditional Fourier transform algorithm, our SFCFT algorithm has lower error rates. SFCFT transforms the original field into a field where nonzero Fourier coefficients exist. This improves SFCFT's robustness to noise.

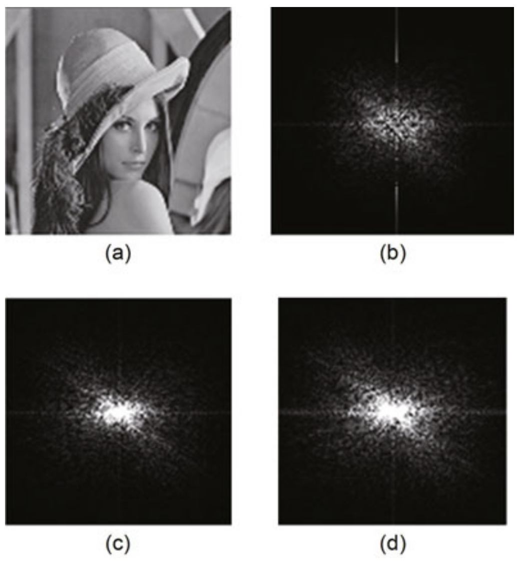

Fig. 6 The original image (a) and the frequency spectrum maps with FFT (b), CFT (c), and SFCFT (d)
The runtimes of FFT, CFT, and SFCFT are 0.029599, 0.054489, and 0.006536 s, respectively

4.3 Performance using color images

In this experiment, we use SFCFT in processing higher-dimensional signals. We apply our algorithm in color-image processing and compare the results with those using FFT and CFT. We choose a 'Lena' RGB image (512×512) to evaluate the performance of the algorithm. An RGB image has three color-channels (i.e., red, green, and blue channels); thus, FFT is calculated for each color-channel separately. It means that three grayscale Fourier transforms are calculated for each channel.

However, SFCFT can extract the multivectors from the color-image data and transform them into complex signals by implementing the Clifford algebra and then calculating their Fourier transformations in a numerical way. A 3D vector field is transformed into a multivector field with only bivector and vector parts that are unequal to zero, as bivector and vector form three complex signals. Thus, we can see from Fig. 7 that the SFCFT algorithm is much faster than FFT and CFT, and its frequency spectrum map is clearer. It means that SFCFT presents more image data information. SFCFT can discover both geometric and spectral information of the multispectral image. The wider the bands of the multispectral image, the more outstanding the SFCFT.

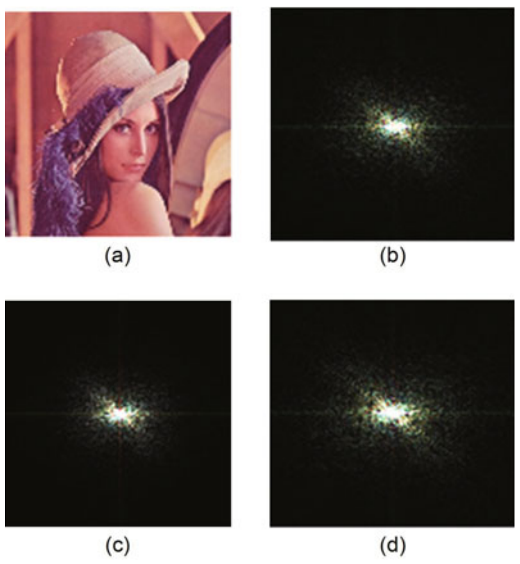

Fig. 7 The original RGB image (a) and the frequency spectrum maps with FFT (b), CFT (c), and SFCFT (d)
The runtimes of FFT, CFT, and SFCFT are 0.195382, 0.774559, and 0.016852 s, respectively. References to color refer to the online version of this figure

5 Conclusions

In this paper, we have proposed a novel algorithm called the 'sparse fast Clifford Fourier transform', which combines the sparse Fourier transform with CFT to process multivector signals. We have discussed the application of SFCFT to the scalar field and grayscale and color image processing. The results demonstrate that SFCFT can effectively improve the performance of multivector signal processing. Our method is available in most vector fields; however, it may not be sparse for a large number of vectors concentrated in specific subareas. As next steps, we will measure the uncertainties to indicate the importance of matches, and offer some improvements to the preliminary segmentation and transformation computed on each segment.

References

Batard, T., Berthier, M., Saint-Jean, C., 2010. Clifford–Fourier transform for color image processing. *In*: Bayro-Corrochano, E., Scheuermann, G. (Eds.), Geometric Algebra Computing. Springer London, London, UK, p.135-162. https://doi.org/10.1007/978-1-84996-108-0_8

Bujack, R., Scheuermann, G., Hitzer, E., 2015. Demystification of the geometric Fourier transforms and resulting convolution theorems. *Math. Meth. Appl. Sci.*, **39**(7): 1877-1890. https://doi.org/10.1002/mma.3607

Cao, W.M., Feng, H., 2010. Geometric Algebra in Biomimetic Pattern Recognition and Signal Processing. Science Press, Beijing, China (in Chinese).

Cao, W.M., Liu, H., Xu, C., *et al.*, 2013. 3D medical image registration based on conformal geometric algebra. *Sci. China Inform. Sci.*, **43**(2):254-274. https://doi.org/10.1360/112012-592

Ebling, J., Scheuermann, G., 2003. Clifford convolution and pattern matching on vector fields. IEEE Visualization, p.193-200. https://doi.org/10.1109/VISUAL.2003.1250372

Ebling, J., Scheuermann, G., 2005. Clifford Fourier transform on vector fields. *IEEE Trans. Visual. Comput. Graph.*, **11**(4):469-479. https://doi.org/10.1109/TVCG.2005.54

Evans, C.J., Sangwine, S.J., Ell, T.A., 2000. Colour-sensitive edge detection using hypercomplex filters. 10th European Signal Processing Conf., p.1-4.

Frigo, M., Johnson, S.G., 2005. The design and implementation of FFTW3. *Proc. IEEE*, **93**(2):216-231. https://doi.org/10.1109/JPROC.2004.840301

Gilbert, A., Indyk, P., 2010. Sparse recovery using sparse matrices. *Proc. IEEE*, **98**(6):937-947. https://doi.org/10.1109/JPROC.2010.2045092

Gilbert, A.C., Indyk, P., Iwen, M., *et al.*, 2014. Recent developments in the sparse Fourier transform: a compressed Fourier transform for big data. *IEEE Signal Process. Mag.*, **31**(5):91-100. https://doi.org/10.1109/MSP.2014.2329131

Hassanieh, H., Adib, F., Katabi, D., *et al.*, 2012a. Faster GPS via the sparse Fourier transform. Proc. 18th Annual Int. Conf. on Mobile Computing and Networking, p.353-364. https://doi.org/10.1145/2348543.2348587

Hassanieh, H., Indyk, P., Katabi, D., *et al.*, 2012b. Nearly optimal sparse Fourier transform. Proc. 44th Annual ACM Symp. on Theory of Computing, p.563-578. https://doi.org/10.1145/2213977.2214029

Hassanieh, H., Indyk, P., Katabi, D., *et al.*, 2012c. Simple and practical algorithm for sparse Fourier transform. Proc. 23rd Annual ACM-SIAM Symp. on Discrete Algorithms, p.1183-1194. https://doi.org/10.1137/1.9781611973099.93

Hassanieh, H., Shi, L., Abari, O., *et al.*, 2014. GHz-wide sensing and decoding using the sparse Fourier transform. Proc. IEEE INFOCOM, p.2256-2264. https://doi.org/10.1109/INFOCOM.2014.6848169

Hestenes, D., 1999. New Foundations for Classical Mechanics. Springer, New York, USA.

Hestenes, D., Sobczyk, G., 1984. Clifford Algebra to Geometric Calculus. Springer, New York, USA.

Hitzer, E., 2012. Introduction to Clifford's geometric algebra. *SICE J. Contr. Meas. Syst. Integr.*, **4**(1):1-11.

Hitzer, E., 2013. The Clifford Fourier transform in real Clifford algebras. Int. Conf. on the Applications of Computer Science and Mathematics in Architecture and Civil Engineering, p.227-240.

Hitzer, E., Mawardi, B., 2008. Clifford Fourier transform on multivector fields and uncertainty principles for dimensions *n*=2 (mod 4) and *n*=3 (mod 4). *Adv. Appl. Clifford Alg.*, **18**(3-4):715-736. https://doi.org/10.1007/s00006-008-0098-3

Hitzer, E., Sangwine, S.J., 2013. Quaternion and Clifford Fourier transforms and wavelets. *In*: Hitzer, E., Stephen, J., Sangwine, S.J. (Eds.), Trends in Mathematic. Springer Basel, Basel, Switzerland. https://doi.org/10.1007/978-3-0348-0603-9

Hitzer, E., Helmstetter, J., Abłamowicz, R., 2013. Square roots of −1 in real Clifford algebras. *In*: Hitzer, E., Stephen, J., Sangwine, S.J. (Eds.), Trends in Mathematic. Springer Basel, Basel, Switzerland. https://doi.org/10.1007/978-3-0348-0603-9_7

Iwen, M.A., 2010. Combinatorial sublinear-time Fourier algorithms. *Found. Comput. Math.*, **10**(3):303-338. https://doi.org/10.1007/s10208-009-9057-1

Li, H.B., 2005. Conformal geometric algebra—a new framework for computational geometry. *J. Comput. Aid. Des. Comput. Graph.*, **17**(11):2383-2393.

Mishra, B., Wilson, P., Wilcock, R., 2015. A geometric algebra co-processor for color edge detection. *Electronics*, **4**(1): 94-117. https://doi.org/10.3390/electronics4010094

Mueen, A., Nath, S., Liu, J., 2010. Fast approximate correlation for massive time-series data. Proc. ACM SIGMOD Int. Conf. on Management of Data, p.171-182. https://doi.org/10.1145/1807167.1807188

Porat, E., Strauss, M.J., 2012. Sublinear time, measurement-optimal, sparse recovery for all. Proc. 23rd Annual ACM-SIAM Symp. on Discrete Algorithms, p.1215-1227. https://doi.org/10.1137/1.9781611973099.96

Reich, W., Scheuermann, G., 2010. Analyzing real vector fields with Clifford convolution and Clifford–Fourier transform. *In*: Bayro-Corrochano, E., Scheuermann, G. (Eds.), Geometric Algebra Computing. Springer, London, UK, p.121-133.
https://doi.org/10.1007/978-1-84996-108-0_7

Schlemmer, M., Hotz, I., Natarajan, V., *et al*., 2005. Fast Clifford Fourier transformation for unstructured vector field data. Proc. Int. Conf. on Numerical Grid Generation in Computational Field Simulations, p.101-110.

Schumacher, J., Puschel, M., 2014. High-performance sparse fast Fourier transforms. IEEE Workshop on Signal Processing Systems, p.1-6.
https://doi.org/10.1109/SiPS.2014.6986055

Shi, L., Hassanieh, H., Davis, A., *et al*., 2014. Light field reconstruction using sparsity in the continuous Fourier domain. *ACM Trans. Graph.*, **34**(1), Article 12.
https://doi.org/10.1145/2682631

Smith, J.O., 2011. Spectral Audio Signal Processing. W3K Publishing, London, UK.

Wang, R., Jing, L.B., Tao, L., *et al*., 2013. Digital watermarking algorithm for 3D point cloud model based on Clifford algebra. *J. Shanghai Jiao Tong Univ.*, **47**(12):1863-1869.

Wang, R., Zhang, X., Cao, W.M., 2015. Clifford fuzzy support vector machines for classification. *Adv. Appl. Clifford Alg.*, **26**(2):1-22.
https://doi.org/10.1007/s00006-015-0616-z

Xie, W., Cao, W.M., Meng, S., 2008. Coverage analysis for sensor networks based on Clifford algebra. *Sci. China Inform. Sci.*, **51**(5):460-475.
https://doi.org/10.1007/s11432-008-0048-7

Xu, C., Liu, H., Ouyang, C.J., *et al*., 2011. Theory and application of Clifford pseudo-differential operator on multispectral image. *Sci. Sin. Inform.*, **41**(12):1423-1435.
https://doi.org/10.1360/zf2011-41-12-1423