J Fourier Anal Appl (2008) 14: 145–179
DOI 10.1007/s00041-008-9013-5

FFTs on the Rotation Group

Peter J. Kostelec · Daniel N. Rockmore

Received: 2 March 2005 / Revised: 27 September 2007 / Published online: 22 February 2008

Abstract We discuss an implementation of an efficient algorithm for the numerical computation of Fourier transforms of bandlimited functions defined on the rotation group $SO(3)$. The implementation is freely available on the web. The algorithm described herein uses $O(B^4)$ operations to compute the Fourier coefficients of a function whose Fourier expansion uses only (the $O(B^3)$) spherical harmonics of degree at most B. This compares very favorably with the direct $O(B^6)$ algorithm derived from a basic quadrature rule on $O(B^3)$ sample points. The efficient Fourier transform also makes possible the efficient calculation of convolution over $SO(3)$ which has been used as the analytic engine for some new approaches to searching 3D databases (Funkhouser et al., ACM Trans. Graph. 83–105, 2003; Kazhdan et al., Eurographics Symposium in Geometry Processing, pp. 167–175, 2003). Our implementation is based on the "Separation of Variables" technique (see, e.g., Maslen and Rockmore, Proceedings of the DIMACS Workshop on Groups and Computation, pp. 183–237, 1997). In conjunction with techniques developed for the efficient computation of orthogonal polynomial expansions (Driscoll et al., SIAM J. Comput. 26(4):1066–1099, 1997), our fast $SO(3)$ algorithm can be improved to give an algorithm of complexity $O(B^3 \log^2 B)$, but at a cost in numerical reliability. Numerical and empirical results are presented establishing the empirical stability of the basic algorithm. Examples of applications are presented as well.

Communicated by Thomas Strohmer.

First author was supported by NSF ITR award; second author was supported by NSF Grant 0219717 and the Santa Fe Institute.

P.J. Kostelec
MIT Lincoln Laboratory, 244 Wood Street, Lexington, MA 02420, USA

D.N. Rockmore (✉)
Department of Mathematics, Dartmouth College, Hanover, NH 03755, USA
e-mail: dnrockmore@gmail.com

Keywords Fast Fourier transform · Rotation group · Spherical harmonics · Wigner D-function · Discrete polynomial transform · Pattern matching

Mathematics Subject Classification (2000) Primary 43-04 · Secondary 43A75

1 Introduction

As usual, let $SO(3)$ denote the rotation group in three dimensions, represented as the three-by-three matrices of determinant one. Functions defined on $SO(3)$ have an expansion analogous to the Fourier series representation of a function defined on the circle, also recognized as the group of one-by-one matrices of determinant one. In the latter case there is a well-known notion of bandlimit, and in this case the classic Fast Fourier Transform (FFT) is a family of algorithms that computes the Fourier expansion of a function of bandlimit B in $O(B \log B)$ arithmetic operations, as opposed to $O(B^2)$ required for direct computation. Moreover, the FFT is also highly stable numerically (see, e.g., [13] and [27]).

A natural notion of bandlimit for $SO(3)$ is given in terms of the highest degree spherical harmonic used in the Fourier expansion of a function defined on the group. In this case direct calculation of the corresponding Fourier transform of a function of bandlimit B (using a concomitant quadrature formula) would require $O(B^6)$ operations. In this article we present and discuss an implementation of an $O(B^4)$ algorithm for this calculation. The associated software package, which we named "The *SOFT* Package," is freely available on the web [32]. Note that for $SO(3)$, bandlimit B implies $O(B^3)$ Fourier coefficients obtained (according to our scheme) by using $O(B^3)$ sample points. All results stated herein have these underlying assumptions.

Beyond theoretical interest, the existence and implementation of an FFT for $SO(3)$ has many potential applications [2]. Many of these use the FFT on $SO(3)$ as a key step in a fast convolution algorithms on $SO(3)$ or the 2-sphere, S^2 (realized as the quotient $SO(3)/SO(2)$). In short, in the same way that convolution or correlation on the cyclic group is used as a form of matched filter for time series data, we can try to use convolution over $SO(3)$ to find a pattern on the sphere by convolving a template against an arbitrary function over $SO(3)$. Of particular interest is some recent work by Funkhouser et al. [11, 19] using these FFTs for the efficient searching of 3D databases — volumetric databases. Objects are stored as discretized concentric spheres each of whose spherical Fourier transforms are then computed and stored. The norms of the sets of Fourier coefficients (grouped together according to degree of the associated harmonic) give an analogue to the usual power spectrum descriptor of a one-dimensional function, that can be used for a search algorithm. Other promising possibilities for applications have been suggested in molecular biology [21], industrial manufacturing [15, 16], cosmology [35] and even spherical near-field antenna measurements [14].

1.1 Related Work

In addition to the approach detailed herein, there are other possible techniques for computing the Fourier transform on $SO(3)$. In particular, we would like to point out that an implementation based the work of Risbo [28] would also give rise to an $O(B^4)$ implementations. The critical difference between such an algorithm and our own is that our separation of variables approach preserves the possibility of finding an improvement through the use of a fast algorithm for the computation of a *discrete Wigner transform*. Indeed, there is such an algorithm, based on the fact that these special functions (like so many others) satisfy a three-term recurrence [7] (see the remark at the end of Section 3). Gluing this fast algorithm into the middle of our algorithm results in a $O(B^3 \log^2 B)$ algorithm. However, the improved complexity estimate comes with a cost of some practical difficulties (see, e.g., [17, 18] for the problems that occur in the case of implementing this for efficient spherical harmonic expansions) that would need to be addressed in the $SO(3)$ setting. We will discuss these difficulties later. Both Risbo's approach and ours are algebraic and exact in exact arithmetic.

A different method of attack might come from so-called "approximate" techniques whose complexity increases with an increase in accuracy. In particular, we point to the recent interesting work of Rokhlin and Tygert [29], which gives an $O(N \log N \log(1/\epsilon))$ algorithm for spherical harmonic expansions for functions on S^2 where N is the number of nodes used in the associated discretization and ϵ is the precision. Thus, the running time is tied to accuracy in both sampling and precision. In [29] experiments are performed showing the stability and efficiency of this approach, implemented in Fortran, for individual associated Legendre transforms. It would be of interest to apply directly these ideas to the problem of finding a fast Wigner transform.

1.2 Organization

This article is organized as follows. We begin with a brief introduction to $SO(3)$, including some discussion of the necessary harmonic analysis. This is then followed by introducing a sampling theorem for functions defined on $SO(3)$, and outlining exactly what we mean by the "separation of variables" approach. The discrete Wigner transform is also defined. Due to the plethora of conventions that exist, in Section 4 we provide explicit definitions of the Wigner-*d* functions as implemented in our software. Section 5 gives error and timing results for *SOFT*. This is then followed by discussion of an application of Fourier transforms defined on $SO(3)$: Correlating two functions defined on S^2. We conclude with a brief recap and discussion.

2 Life in $SO(3)$

We are interested in the Fourier analysis or harmonic analysis of functions defined on $SO(3)$. This is one of the most well-studied groups in all of mathematics. The books [34, 37, 38] are standard places to look for this material. We collect here as much as we need to keep things relatively self-contained.

2.1 Euler Angle Coordinates

We may express any element $g \in SO(3)$ in terms of rotations about the z and y axes. Let

$$u(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad a(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \tag{2.1}$$

where $0 \leq \alpha < 2\pi$ and $0 \leq \beta \leq \pi$. Geometrically speaking, $u(\alpha)$ corresponds to rotation by α radians about the z-axis, and $a(\beta)$ to rotation by β radians about the y-axis. Using these two matrices, any rotation $g \in SO(3)$ has an associated *Euler angle decomposition*:

$$g = g(\alpha, \beta, \gamma) = u(\alpha)\, a(\beta)\, u(\gamma)\ . \tag{2.2}$$

The Euler angle decomposition provides a natural coordinate system for working with functions on $SO(3)$, allowing us to write a function $f(g)$ for $g \in SO(3)$ as $f(\alpha, \beta, \gamma)$, where $0 \leq \alpha, \gamma < 2\pi$ and $0 \leq \beta \leq \pi$.

2.2 Fourier Analysis in $SO(3)$

Let $L^2(SO(3))$ denote the space of square integrable functions on $SO(3)$. In coordinates the inner product of two functions f and h on $SO(3)$ is given as

$$< f, h > \;=\; \int_0^{2\pi} d\alpha \int_0^{\pi} d\beta \sin\beta \int_0^{2\pi} d\gamma\;\; f(\alpha, \beta, \gamma)\; h^*(\alpha, \beta, \gamma) \tag{2.3}$$

where $h^*(\alpha, \beta, \gamma)$ denotes the complex conjugate of $h(\alpha, \beta, \gamma)$.

To each $g \in SO(3)$, we can associate a linear operator $\Lambda(g)$ which acts on f in $L^2(S^2)$:

$$\Lambda(g) f(\omega) = f\left(g^{-1}\omega\right)\ . \tag{2.4}$$

This is the *left regular representation of* $SO(3)$ *on* $L^2(S^2)$. The invariant (and pairwise orthogonal) subspaces of this action, denoted V_l, are indexed by the nonnegative integers with $\dim V_l = 2l + 1$. The familiar *spherical harmonics*, $\{Y_l^m | l \geq 0;\ |m| \leq l\}$, span these invariant subspaces. To be precise, for a given integer $l \geq 0$, V_l is spanned by the spherical harmonics of degree l and orders $|m| \leq l$, denoted $\{Y_l^m(\omega) \mid |m| \leq l\}$. We therefore have

$$\Lambda(g) Y_l^m(\omega) = \sum_{|k| \leq l} Y_l^k(\omega) D_{km}^l(g) \tag{2.5}$$

where $D_{km}^l(g)$ is a **Wigner-*D* function**.

Using the Euler angle decomposition, we can write the Wigner-D function as

$$D_{MM'}^J(\alpha, \beta, \gamma) = e^{-iM\alpha}\, d_{MM'}^J(\beta)\, e^{-iM'\gamma}, \tag{2.6}$$

where J ranges over the nonnegative integers, and $0 \le |M|, |M'| \le J$, and $d^J_{MM'}(\beta)$ denotes what we will refer to as the **Wigner *d*-function**. The Wigner-d functions are related to the Jacobi polynomials, and satisfy a three-term recurrence relation. Wigner-D functions are related to the more familiar Y_l^m functions by

$$Y_l^m(\theta,\phi) = (-1)^m \sqrt{\frac{2l+1}{4\pi}} D^{l*}_{0m}(\chi,\theta,\phi)$$

where χ is an arbitrary angle. Explicit expressions for $d^J_{MM'}(\beta)$ will be given in Section 4.

By the celebrated Peter-Weyl Theorem, the collection of functions $\{D^J_{MM'}(\alpha, \beta,\gamma)\}$ form a complete set of orthogonal functions with respect to integration over $SO(3)$:

$$\left\langle D^{J_1}_{M_1 M'_1}, D^{J_2}_{M_2 M'_2} \right\rangle = \frac{8\pi^2}{2J_1+1} \delta_{J_1 J_2}\, \delta_{M_1 M_2}\, \delta_{M'_1 M'_2} \,. \tag{2.7}$$

Hence, any $f \in L^2(SO(3))$ may be written as a sum of Wigner D-functions:

$$f(\alpha,\beta,\gamma) = \sum_{J\ge 0} \sum_{M=-J}^{J} \sum_{M'=-J}^{J} \hat{f}^J_{MM'} D^J_{MM'}(\alpha,\beta,\gamma) \tag{2.8}$$

where

$$\begin{aligned}\hat{f}^J_{MM'} &= \frac{2J+1}{8\pi^2} \left\langle f, D^J_{MM'} \right\rangle \\ &= \frac{2J+1}{8\pi^2} \int_0^{2\pi} d\alpha \int_0^{\pi} d\beta \sin\beta \int_0^{2\pi} d\gamma \;\; f(\alpha,\beta,\gamma) D^{J*}_{MM'}(\alpha,\beta,\gamma) \,. \end{aligned} \tag{2.9}$$

We will refer to the set of values $\{\hat{f}^J_{MM'}\}$ as the **Fourier coefficients** of f, and the map going from $f(\alpha,\beta,\gamma)$ to the set of coefficients $\{\hat{f}^J_{MM'}\}$ as the ***SO*(3) Fourier transform of *f***. As a notational convenience, we will denote the $SO(3)$ Fourier transform of f as $SOFT(f)$.

Let us now take a second look at the linear operator, $\Lambda(g)$, and see its effect on a function $f \in L^2(S^2)$. Such a function can be expanded in spherical harmonics:

$$f(\omega) = \sum_{l\ge 0} \sum_{|m|\le l} \hat{f}_l^m Y_l^m(\omega)$$

where

$$\hat{f}_l^m = \int_{S^2} f(\omega)\, \overline{Y_l^m}\, d\omega \,.$$

Applying the operator to a function $f \in L^2(S^2)$ can be thought of as rotating its graph. And how do the Fourier coefficients of f relate to its rotated cousin? Using (2.5), we can elegantly obtain the Fourier expansion of the rotated f by means of a matrix multiplication. The matrix in question is a semi-infinite block diagonal matrix, where the l^{th} block is of size $(2l+1)\times(2l+1)$, and each block is $D^l(g) = \left(D^l_{mn}\right)(g)$. Using all this, the Fourier expansion of the rotated f is the matrix-vector product

$$\left(\begin{array}{c|c|c|c} D^0(g) & & & \\ \hline & D^1(g) & & \\ \hline & & D^2(g) & \\ \hline & & & \ddots \end{array}\right) \cdot \left(\begin{array}{c} \hat{f}_0^0 \\ \hdashline \hat{f}_1^{-1} \\ \hat{f}_1^0 \\ \hat{f}_1^1 \\ \hdashline \hat{f}_2^{-2} \\ \hat{f}_2^{-1} \\ \hat{f}_2^0 \\ \vdots \end{array}\right) .$$

2.3 Convolution on $L^2(SO(3))$

Convolution of two functions $f_1, f_2 \in L^2(SO(3))$ is defined to be

$$f_1 * f_2(h) = \int_{SO(3)} f_1\left(hg^{-1}\right) f_2(g)\, dg$$

where $h \in SO(3)$. Given the Fourier coefficients in the expansions of f_1 and f_2:

$$f_1(g) = \sum_{J\geq 0} \sum_{M=-J}^{J} \sum_{M'=-J}^{J} a^J_{MM'} D^J_{MM'}(g)$$

$$f_2(g) = \sum_{J\geq 0} \sum_{M=-J}^{J} \sum_{M'=-J}^{J} b^J_{MM'} D^J_{MM'}(g) ,$$

we find the Fourier coefficients of $f_1 * f_2$ to be

$$(f_1 * f_2)\hat{}^{\,J}_{MM'} = c^J_{MM'} = \sum_{k=-J}^{J} a^J_{Mk} b^J_{kM'} . \tag{2.10}$$

Details may be found in Vilenkin [34].

Note that there is an elegant algebraic description of the Fourier coefficients of $f_1 * f_2$. Corresponding to each integer J is an irreducible representation of $SO(3)$. As described above the coefficients of f_1 and f_2 each can be arranged as a block diagonal matrix, where the J-th block, of size $(2J+1)\times(2J+1)$, contains the (suitably ordered) Fourier coefficients at that degree J. The set of Fourier coefficients of the convolution is just the product of the two matrices. So all that is happening is that the J-th block in f_1's matrix is multiplied with the J-th block in f_2's matrix. This is precisely the noncommutative analogue of the more widely known result describing the Fourier transform of the convolution of two functions on the circle as the pointwise product of the individual Fourier transforms.

2.4 Sampling Theorem

Much of the discussion which follows is based on [22, 23].

To compute $SOFT(f)$ means to evaluate the inner products (2.9). In practice, to even attempt to do this on the computer means that we must discretize the integral, i.e., replace it with a finite sum. This compels us to sample f on some finite grid. Ergo, the sum we need to evaluate is of the general form:

$$\sum_{x \in X} w(x) f(x) D^{l*}_{MM'}(x) \tag{2.11}$$

where X denotes some finite sampling grid and $w(x)$ is some weighting function. Furthermore, to ensure that a collection of sums of the form (2.11) will accurately compute the Fourier transform of f this way, certain conditions must be placed on f.

Definition 1 (Bandlimited functions on $SO(3)$). A continuous function f on $SO(3)$ is **bandlimited with bandlimit (or bandwidth)** B if $\hat{f}^l_{MM'} = 0$ for all $l \geq B$.

The Euler angle coordinates (2.2) make possible an easily described sampling grid for the exact recovery of bandlimited functions by a sum of the type (2.11). This is better known as a *quadrature rule*. In particular, let X_B denote the set of points:

$$X_B = \{u(\alpha_{j_1}) a(\beta_k) u(\gamma_{j_2}) | 0 \leq j_1, j_2, k < 2B\} \tag{2.12}$$

where $\alpha_j = \gamma_j = \frac{2\pi j}{2B}$ and $\beta_k = \frac{\pi(2k+1)}{4B}$. Furthermore, let $w_B(k)$, $0 \leq k < 2B$, be the unique solution to the system of linear equations

$$\sum_{k=0}^{2B-1} w_B(K)\ P_m(\cos \beta_k) = \delta_{0m} \quad \text{for } 0 \leq m < B \tag{2.13}$$

where P_m denotes the Legendre polynomial of degree m. We can now state the following result, whose proof may be found in [23].

Theorem 1 *Suppose $f \in L^2(SO(3))$ is a bandlimited function with bandlimit B. Then for all $l < B$, we have*

$$\hat{f}^l_{MM'} = \frac{1}{(2B)^2} \sum_{j_1=0}^{2B-1} \sum_{j_2=0}^{2B-1} \sum_{k=0}^{2B-1} w_B(k) f(\alpha_{j_1}, \beta_k, \gamma_{j_2}) D^{l*}_{MM'}(\alpha_{j_1}, \beta_k, \gamma_{j_2}) \ , \tag{2.14}$$

where the sample points are those defined in (2.12) *and weights $w_B(k)$ are the solutions to the linear system* (2.13).

The method of computing the Fourier coefficients of f by summing (2.14) will be called the **Discrete $SO(3)$ Fourier Transform** of f, $DSOFT(f)$.

The weights have a closed form expression.

Lemma 1 (cf. [6]) *Let $\beta_k = \pi(2k+1)/4B$. The solutions to the linear system of Equations* (2.13) *have the closed form expression*

$$w_B(k) = \frac{2}{B} \sin\left(\frac{\pi(2j+1)}{4B}\right) \sum_{k=0}^{B-1} \frac{1}{2k+1} \sin\Big((2j+1)(2k+1)\frac{\pi}{4B}\Big) \tag{2.15}$$

where $0 \le k < 2B$.

3 Our Algorithm

Theorem 1 tells us that we can compute exactly the Fourier coefficients of a properly sampled bandlimited function f of bandlimit B. However, if we were to evaluate directly (2.14) as it is written, each Fourier coefficient would require $O(B^3)$ operations. There being $O(B^3)$ coefficients $\hat{f}^l_{MM'}$ to compute, the cost of computing the complete Fourier transform of f is $O(B^6)$ operations. Is it possible to reorganize the computation and hence reduce the total complexity? Yes, by application of the *separation of variables* technique (see, e.g., [23, 24]).

3.1 Separation of Variables

The basic strategy is to make a prudent choice of factorization of the group elements of $SO(3)$ and use of a set of *adapted* representations, also called Gel'fand-Tsetlin bases, to re-index the sum (2.11) as a multiple sum. (Because we are using the Euler angle decomposition, (2.14) is already in the proper form). Next, terms that do not depend on particular indices are factored through the individual sums, and then the sums are evaluated coordinate by coordinate.

By using the fact that $D^l_{MM'}(\alpha,\beta,\gamma) = e^{-iM\alpha}\, d^l_{MM'}(\beta)\, e^{-iM'\gamma}$, we may rewrite (2.14) as

$$\hat{f}^l_{MM'} = \frac{1}{(2B)^2} \sum_{k=0}^{2B-1} w_B(k) d^l_{MM'}(\beta_k) \sum_{j_2=0}^{2B-1} e^{iM'\gamma_{j_2}} \sum_{j_1=0}^{2B-1} e^{iM\alpha_{j_1}} f(\alpha_{j_1},\beta_k,\gamma_{j_2})\ . \tag{3.1}$$

This form lends itself to the efficient computation of $\hat{f}^l_{MM'}$: First sum over j_1, then j_2 and conclude by summing over k. To be slightly more explicit, first sum the following quantities (*note the indices they depend on*) for $0 \le j_2$, $k < 2B$ and $|M|, |M'| \le l < B$:

$$S_1(k, M, j_2) = \frac{1}{(2B)} \sum_{j_1=0}^{2B-1} e^{iM\alpha_{j_1}} f(\alpha_{j_1},\beta_k,\gamma_{j_2}) \tag{3.2}$$

$$S_2\big(k, M, M'\big) = \frac{1}{(2B)} \sum_{j_2=0}^{2B-1} e^{iM'\gamma_{j_2}} S_1(k, M, j_2) \tag{3.3}$$

$$\hat{f}^l_{MM'} = \sum_{k=0}^{2B-1} w_B(k) d^l_{MM'}(\beta_k) S_2\big(k, M, M'\big) . \tag{3.4}$$

For a given pair k, j_2, the sum $S_1(k, M, j_2)$ for all M can be evaluated in $O(B \log B)$ operations using a standard, Cooley-Tukey-esque FFT. Therefore, to compute $S_1(k, M, j_2)$ for all k, M, j_2 takes $O(B^3 \log B)$ operations. For identical reasons, the same big-O is applicable for computing $S_2(k, M, M')$. All that remains is evaluating that last sum (3.4). For given M, M', $O(B^2)$ operations are needed to compute $\hat{f}^l_{MM'}$ for all l. Therefore, putting all the pieces together yields an $O(B^4)$ algorithm for computing the Fourier transform of f, substantially smaller than the original, "naive," $O(B^6)$ computation. From now on, we will refer to this method of computing all the Fourier coefficients of f as the **$SO(3)$ Fast Fourier transform of f, $SOFFT(f)$.**

Given the well-known numerical reliability of the Cooley-Tukey FFT (see, e.g., [27]), there is very little reason to investigate the quality of the sums (3.2)–(3.3). The place where errors would be introduced (and exacerbated) is in the third sum (3.4). Given its import, we are motivated to make the following definition.

Definition 2 For given integers (M, M'), define the **Discrete Wigner Transform (DWT) of a data vector s** to be the collection of sums of the form

$$\hat{\mathbf{s}}\big(l, M, M'\big) = \sum_{k=0}^{2B-1} w_B(k)\; d^l_{M,M'}(\beta_k)[\mathbf{s}]_k \qquad \max\big(|M|, \big|M'\big|\big) \le l < B \tag{3.5}$$

where $d^l_{M,M'}$ is a Wigner d-function of degree l and orders M, M', and $\beta_k = \dfrac{\pi(2k+1)}{4B}$.

In our case, the data vector **s** is simply the original function f sampled at the $2B$-many locations β_k defined above. In Section 5 we report the results of our experiments, testing the numerical validity of the DWT for a variety of bandlimits, as well as the $SOFFT$ algorithm.

Since, as we will soon see, the Wigner-d functions satisfy a three-term recurrence, it is possible to use the efficient algorithms given in [7] to compute the DWTs, thereby reducing the $O(B^2)$ operation to $O(B \log^2 B)$. This would then yield an $O(B^3 \log^2 B)$ "really" $SOFFT$ algorithm. However, this was not done in the C code. We will explain our reasoning in Section 5.2.

Finally, we mention that DWT may be cast in a linear algebraic light. Let **s** = data vector, **ŝ** = coefficient vector, **w** = diagonal matrix whose entries are the weights, **d** = sampled Wigner-d functions, $d_{ij} = d^i_{MM'}(\beta_j)$. Then

$$\mathbf{d} * \mathbf{w} * \mathbf{s} = \hat{\mathbf{s}} \tag{3.6}$$

is the forward DWT, and

$$\mathbf{d}^T * \hat{\mathbf{s}} = \mathbf{s} \tag{3.7}$$

is the inverse DWT.

Table 1 Amount of memory, in megabytes, required to hold all the complex-valued function samples necessary for a discrete Fourier transform on $SO(n)$, using the C-type double.

Bandwidth	$SO(3)$	$SO(4)$	$SO(5)$	$SO(6)$	$SO(7)$
4	< 1	< 1	1/2	4	32
8	< 1	1	16	256	4096
16	1/2	16	512	16384	524288
32	4	256	16384	1.05×10^6	6.71×10^7
64	32	4096	524288	6.71×10^7	8.59×10^9
128	256	65536	1.68×10^7	4.29×10^9	1.10×10^{12}
256	2048	1.05×10^6	5.37×10^8	2.75×11^{11}	1.41×10^{14}

Generalizations to higher order rotation groups. Certainly the Separation of Variables technique can be applied to the higher order rotation groups, $SO(n)$. However, certain practical challenges are encountered which need to be overcome, not the least of which is having sufficient memory simply to hold the function samples. Table 1 illustrates this quite clearly. In it, we give the amount of RAM, in megabytes, required to hold all complex-valued function samples required to take the discrete Fourier transform of a function $f \in L^2(SO(n))$ at various bandlimits. Note how quickly memory requirements grow as bandwidth and n increase. If someone really wants to calculate the Fourier coefficients of a function on $SO(7)$, probably they will not get very far.

3.2 Alternative Approaches

Computing the inverse FFT of $f \in L^2(SO(3))$, i.e., to go from the Fourier coefficients $\{\hat{f}^l_{MM'}\}$ to the function samples $\{f(\alpha_{j_1}, \beta_k, \gamma_{j_2})\}$, can be done in one of two ways. One way, as suggested by (3.2)–(3.4), is to first to do all the necessary inverse DWTs, followed by all the necessary inverse Cooley-Tukey-esque FFTs. This can be loosely described as doing one set of inverse DWTs, followed by two sets of inverse "classical" FFTs.

Another way is to apply Risbo's technique [28], which uses the identity

$$d^l_{MM'}(\beta) = \imath^{M-M'} \sum_{M''=-j}^{j} d^j_{M''M}(\pi/2) e^{-\imath M''\beta} d^j_{M''M'}(\pi/2) \tag{3.8}$$

(here, we quote the version from [26]). The expression (3.8) is the Fourier expansion of $d^l_{MM'}(\beta)$. Edmonds [8] seems to have been the first to publish this observation, citing unpublished work of Wigner [36]. It all arises from considering the rewriting of a rotation about the y-axis in terms of a rotation about the z-axis:

$$g(0,\beta,0) = g(-\pi/2,0,0)\; g(0,-\pi/2,0)\; g(\beta,0,0)\; g(0,\pi/2,0)\; g(\pi/2,0,0)\ .$$

Note that the rotation β is now about the z-axis.

We will not go into details here, but Risbo gives efficient and stable recursive algorithms for evaluating $d^l_{MM'}(\beta)$. Unlike the more familiar three-term recurrences (which we will give in Section 4), these mix both the degree and orders when incrementing by *half-degree* steps. The basic recurrence can be realized as a clever convolution involving a certain 2×2 matrix whose entries are related to the Cayley-Klein rotation parameters

$$a = \cos\frac{\beta}{2} e^{-\imath\frac{\alpha+\gamma}{2}}$$
$$b = \sin\frac{\beta}{2} e^{\imath\frac{\alpha-\gamma}{2}} .$$

Risbo mentions that this algorithm may be gleaned from an appendix to Chapter 7 of Courant and Hilbert [3].

The point of all this is that it makes possible an algorithm in which we first efficiently calculate $d^l_{MM'}(\pi/2)$, then replace $d^l_{MM'}(\beta)$ with its Fourier expansion in (3.1), move summations around, and from there go on to computing the inverse $SO(3)$ FFT via three sets of classical inverse FFTs. No inverse DWTs need be done. This technique has seen application in a number of different areas, including cosmology [35], molecular biology [21], and elsewhere [12].

However, the algorithm is still $O(B^4)$, and its structure precludes the possibility of obtaining an asymptotically faster algorithm. That is not to imply that it is not fast, i.e., in terms of looking at the clock on the wall. It all depends on the application, and what is important to the user. Both Risbo's technique and the approach presented here get the job done.

4 Explicit Wigners

Everyone agrees that the Wigner D-function may be written as

$$D^J_{MM'}(\alpha,\beta,\gamma) = e^{-iM\alpha}\, d^J_{MM'}(\beta)\, e^{-iM'\gamma} ,$$

where, we remind the reader, $d^J_{MM'}(\beta)$ denotes the Wigner d-function. There are, however, at least a few common expressions for $d^J_{MM'}(\beta)$. One such (see [1]) is

$$d^J_{MM'}(\beta) = \sqrt{\frac{(J+M')!\,(J-M')!}{(J+M)!\,(J-M)!}} \left(\sin\frac{\beta}{2}\right)^{M'-M} \left(\cos\frac{\beta}{2}\right)^{M+M'} \times P^{(M'-M,M'+M)}_{J-M'}(\cos\beta) \tag{4.1}$$

where $P^{(m,n)}_l(x)$ denotes a Jacobi polynomial. Another common formulation is in terms of derivatives [26] and is given by:

$$d^J_{MM'}(\beta) = C_{JM'}\sqrt{\frac{(J+M)!}{(J-M)!(2J)!}}\,(1-t)^{-(M-M')/2}(1+t)^{-(M+M')/2}$$

$$\times \frac{d^{J-M}}{dt^{J-M}}\left[(1-t)^{J-M'}(1+t)^{J+M'}\right] \tag{4.2}$$

where $t=\cos\beta$ and $C_{JM'}=(-1)^{J-M'}2^{-J}\sqrt{\frac{(2J)!}{(J+M')!(J-M')!}}$.

In the interests of clarity and full disclosure we feel it appropriate to provide the reader with the precise definition of $d^J_{MM'}(\beta)$, as well as other relevant functions, upon which our C code for computing $SOFFT(f)$ for band-limited f is based. All these definitions may be found in [33].

Following (4.1) we too use Jacobi polynomials to express the Wigner d-function, though at first glance, the expressions may not seem equivalent:

$$d^J_{MM'}(\beta)=\zeta_{MM'}\sqrt{\frac{s!(s+\mu+\nu)!}{(s+\mu)!(s+\nu)!}}\left(\sin\frac{\beta}{2}\right)^{\mu}\left(\cos\frac{\beta}{2}\right)^{\nu} \times P_s^{(\mu,\nu)}(\cos\beta) \tag{4.3}$$

where

$$\mu=\left|M-M'\right| \; \nu=\left|M+M'\right| \; s=J-\frac{\mu+\nu}{2}$$

and

$$\zeta_{MM'}=\begin{cases}1 & \text{if} \quad M'\geq M\\ (-1)^{M'-M} & \text{if} \quad M'<M\,.\end{cases}$$

Note that unless $J\geq\max(|M|,|M'|)$, we have $d^J_{MM'}(\beta)=0$. The d-functions satisfy the orthogonality condition

$$\int_0^{\pi} d^J_{MM'}(\beta)d^{J'}_{MM'}(\beta)\;\sin\beta\;d\beta=\frac{2}{2J+1}\delta_{JJ'}\,, \tag{4.4}$$

in addition to the following three-term recurrence:

$$\begin{aligned}0 \;=\; &\frac{\sqrt{\left[(J+1)^2-M^2\right]\left[(J+1)^2-M'^2\right]}}{(J+1)(2J+1)}\;d^{J+1}_{MM'}(\beta)\\ &+\left(\frac{MM'}{J(J+1)}-\cos\beta\right)d^J_{MM'}(\beta)\\ &+\frac{\sqrt{\left(J^2-M^2\right)\left(J^2-M'^2\right)}}{J(2J+1)}\;d^{J-1}_{MM'}(\beta)\,.\end{aligned} \tag{4.5}$$

To properly initialize the above recurrence, we have found the following special cases, where $0\leq M\leq J$, especially useful:

$$d^J_{JM}(\beta)=\sqrt{\frac{(2J)!}{(J+M)!(J-M)!}}\left(\cos\frac{\beta}{2}\right)^{J+M}\left(-\sin\frac{\beta}{2}\right)^{J-M} \tag{4.6}$$

$$d^J_{-JM}(\beta) = \sqrt{\frac{(2J)!}{(J+M)!(J-M)!}} \left(\cos\frac{\beta}{2}\right)^{J-M} \left(\sin\frac{\beta}{2}\right)^{J+M} \tag{4.7}$$

$$d^J_{MJ}(\beta) = \sqrt{\frac{(2J)!}{(J+M)!(J-M)!}} \left(\cos\frac{\beta}{2}\right)^{J+M} \left(\sin\frac{\beta}{2}\right)^{J-M} \tag{4.8}$$

$$d^J_{M-J}(\beta) = \sqrt{\frac{(2J)!}{(J+M)!(J-M)!}} \left(\cos\frac{\beta}{2}\right)^{J-M} \left(-\sin\frac{\beta}{2}\right)^{J+M} . \tag{4.9}$$

Using the above, we were able to confirm correct generation of the d-functions by comparison with the tables found in [33].

For computational purposes we deal with the L^2-normalized cousins of the Wigner d-functions,

$$\tilde{d}^J_{MM'}(\beta) = \sqrt{\frac{2J+1}{2}}\, d^J_{MM'}(\beta) . \tag{4.10}$$

Therefore, for the convenience of the reader, here is the three-term recurrence relation satisfied by the functions $\tilde{d}^J_{MM'}$. This is the actual relation the C code uses to generate the normalized Wigner d-functions:

$$\begin{aligned}
&\tilde{d}^{J+1}_{MM'}(\beta) \\
&= \sqrt{\frac{2J+3}{2J+1}} \frac{(J+1)(2J+1)}{\sqrt{\left[(J+1)^2 - M^2\right]\left[(J+1)^2 - M'^2\right]}} \left(\cos\beta - \frac{MM'}{J(J+1)}\right) \tilde{d}^J_{MM'}(\beta) \\
&\quad - \sqrt{\frac{2J+3}{2J-1}} \frac{\sqrt{\left[J^2 - M^2\right]\left[J^2 - M'^2\right]}}{\sqrt{\left[(J+1)^2 - M^2\right]\left[(J+1)^2 - M'^2\right]}} \frac{J+1}{J} \tilde{d}^{J-1}_{MM'}(\beta) .
\end{aligned} \tag{4.11}$$

So how stable is the above recurrence? To try to get a handle on this question, let us consider the DWT from the perspective of linear algebra.

At a given bandwidth B and orders M, M', let $J = \max(|M|, |M'|)$, and let $\mathbf{w}$ denote the $2B \times 2B$ diagonal matrix containing the quadrature weights. Let $\mathbf{d}$ be the $(B - J + 1) \times 2B$ matrix containing the sampled Wigner d-functions:

$$\mathbf{d} = \begin{pmatrix} d^J_{MM'}(\cos\theta_0) & d^J_{MM'}(\cos\theta_1) & \dots & d^J_{MM'}(\cos\theta_{2B-1}) \\ d^{J+1}_{MM'}(\cos\theta_0) & d^{J+1}_{MM'}(\cos\theta_1) & \dots & d^{J+1}_{MM'}(\cos\theta_{2B-1}) \\ \vdots & & & \\ d^{B-1}_{MM'}(\cos\theta_0) & d^{B-1}_{MM'}(\cos\theta_1) & \dots & d^{B-1}_{MM'}(\cos\theta_{2B-1}) \end{pmatrix} . \tag{4.12}$$

Then the product $\mathbf{d} * \mathbf{w} * \mathbf{d}^T$ is the $(B - J + 1) \times (B - J + 1)$ identity matrix.

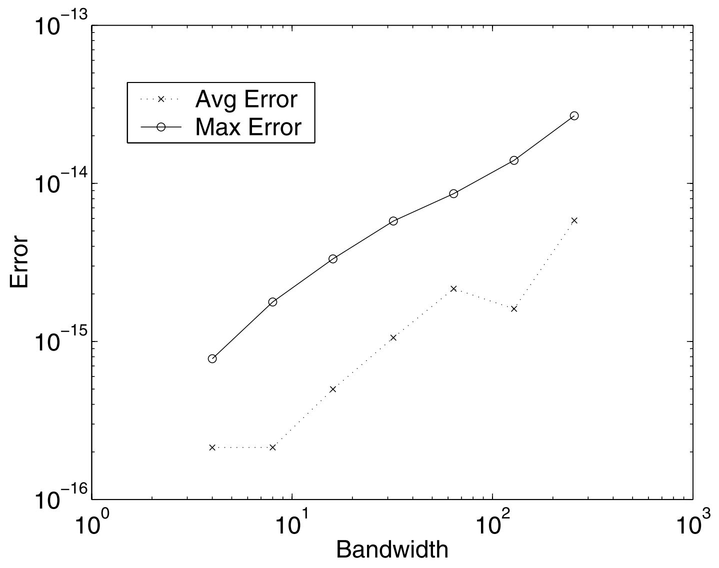


Fig. 1 The largest values of AveErr(M, M') and MaxErr(M, M'), as defined in (4.13)–(4.14), at different bandwidths.

So, how close to the identity matrix do we get in floating point arithmetic? In *Matlab,* at a given bandwidth B, and orders M, M', we calculated two quantities:

$$\text{AveErr}\big(M, M'\big) = 1/B^2 \sum \left|\mathbf{d} * \mathbf{w} * \mathbf{d}^T - I\right| \tag{4.13}$$

$$\text{MaxErr}\big(M, M'\big) = \max \left|\mathbf{d} * \mathbf{w} * \mathbf{d}^T - I\right| , \tag{4.14}$$

where I is the identity matrix, and the sum is done over the matrix elements. The first quantity, (4.13), may be thought of as the average "per-element" error, and (4.14) the maximum "per-element" error.

In Figure 1, at bandwidth B, we plot the largest AveErr(M, M') and MaxErr(M, M') values obtained as the order M ranged from 0 through $B - 1$, and M' ranged from M through $B - 1$. As can be seen, the errors are quite small: Agreement with the identity matrix is good.

5 Numerical Results

In this section, we present numerical evidence which indicates that the proposed algorithm can accurately compute the Fourier transform of band-limited functions $f \in L^2(SO(3))$ at realistic (to be explained later) problem sizes. We will also present some timing results, and discuss issues of computational efficiency.

All code was written in C, with some of the more basic routines borrowed from *SpharmonicKit* [31]. It is freely available on the web as "The *SOFT* Package" [32]. In certain instances, which will be made clear, we availed ourselves of the Fourier

Fig. 2 Outline of our experiment.

$$\begin{tikzcd}
\text{Generate random coefficients} \arrow[d, "\text{Inverse transform}"] \\
\text{Compute sample values} \arrow[d, "\text{Forward transform}"] \\
\text{Compute new coefficients}
\end{tikzcd}$$

transform routines provided by *FFTW* [10]. Experiments were performed on four platforms. Three were GNU/Linux platforms running a variety of processors: An Intel 800 MHz Pentium-III, an Intel 2.4 GHz Xeon, and an AMD 1.66 GHz Athlon MP 2000+. The flavour of GNU/Linux on these machines was either RedHat or Debian. The fourth platform was an HP/Compaq AlphaServer with a 833 MHz Alpha processor running OSF1 V5.1. While some of the hardware was multi-processor, the code was always run on a single processor. No parallelizing was done. The code was compiled with available optimizations. In the case of the HP/Compaq machine, the native compiler, and not gcc, was used.

Figure 2 provides a coarse description of a typical experiment.

The process illustrated in the figure would be iterated some number of times, at the end of which the average absolute and relative errors would be calculated.

5.1 The Discrete Wigner Transform

Before attempting to compute $SOFFT(f)$, for our first series of experiments we investigated the stability of the discrete Wigner d-function transform, with the sums being evaluated naively. We wanted to ensure that an actual implementation of this critical component of the Fourier transform over the full group was in fact stable for a large range of problem sizes. To flesh out the outline given in Figure 2, a precise description of this experiment now follows.

(1) Choose a bandwidth B, and orders M, M'.
(2) Generate random Wigner d-coefficients $\{\hat{f}^k_{MM'}\}$ where $\max(|M|, |M'|) \leq k \leq B-1$, uniformly distributed between -1 and 1.
(3) Take the inverse DWT of the above coefficients, resulting in $2B$-many function samples $\{f(\beta_j) \mid 0 \leq j \leq 2B-1\}$ where $\beta_j = \dfrac{\pi(2j+1)}{4B}$.
(4) Take the forward DWT of the above sample values, resulting in a new set of d-coefficients, $\hat{g}^k_{MM'}$.
(5) Compute the error

$$\max_k \left\| \hat{f}^k_{MM'} - \hat{g}^k_{MM'} \right\| .$$

(6) Repeat Steps (2) through (5) one hundred thousand times.
(7) Compute the average absolute and relative errors over all trials.

The above experiment was run for various bandwidths B, and combinations M and M'. The results obtained on the 2.4 Xeon are given in Table 2. Runs on other platforms displayed similar behavior.

Table 2 Discrete Wigner Transform average absolute (first row) and relative (second row) errors, as obtained on the Intel 2.4 GHz Xeon. The errors for the case $B = 1024$ are the average of 1,000, and not 100,000, iterations. As might be indicated by the symmetry of the d-functions, errors for the case $M = 0$, $M' = B/2$ were, for all practical purposes, similar to the case $M = B/2$, $M' = 0$.

Bandwidth B	$M = M' = 0$	$M = B/2,\ M' = 0$	$M = M' = B/2$
16	2.0990e-12	2.1644e-12	1.9806e-12
	9.8256e-11	3.7374e-11	2.6711e-11
32	2.3308e-12	3.3753e-12	2.3639e-12
	3.4070e-10	2.5885e-10	3.0303e-10
64	1.1389e-11	9.1509e-12	1.1076e-11
	3.7872e-09	1.5096e-09	1.6904e-09
128	3.5974e-11	2.8620e-11	3.6464e-11
	3.2129e-08	8.0481e-09	4.5913e-08
256	1.2473e-10	9.2109e-11	1.0939e-10
	2.0330e-07	4.1057e-08	8.3698e-08
512	5.5025e-10	2.9709e-10	4.6540e-10
	1.6429e-06	1.6119e-07	7.4623e-07
1024	2.1756e-08	9.3919e-10	1.5819e-08
	3.1479e-04	5.6260e-07	8.0374e-05

At the smaller bandwidths, the errors are all acceptably and gratifyingly small. However, a careful examination of Table 2 reveals that as the bandwidth increases, the relative error eventually approaches anxiety-inducing levels. By the time we reach bandwidth $B = 1024$, we are seeing relative errors roughly on the order of 10^{-5}. However, as will be detailed in Section 5.2, when results of the full transform are presented and discussed, other practical considerations render any causes of concern at these larger bandwidths unnecessary.

In Figure 3, we show the average CPU runtime of the forward DWT on four different platforms, at a number of different bandwidths B. The Wigner functions necessary for the transform were precomputed in advance. Runtimes for the inverse transform were comparable to those of the forward transform. Some numbers indicating this are shown in Table 3.

Precomputing itself does not take long. On the Xeon, for $B = 1024$, it takes roughly 0.04 seconds to compute the Wigners necessary for a forward transform. However, in our particular implementation, precomputing for the inverse transform takes about 4 times as long. The reason? To obtain the sampled Wigners necessary for the inverse transform, we are simply transposing the matrix of sampled Wigners needed for the forward transform. This takes time. We will see later how matrix transposition becomes a real issue in doing a Fourier transform over the full group.

We find it interesting, from a cultural standpoint, that even on the older 'slow' 800 MHz machine, the DWT is still quite fast. E.g., while the 800 MHz is more than twice as slow as the Xeon, at bandwidth = 64, it still takes the 800 Mhz only about

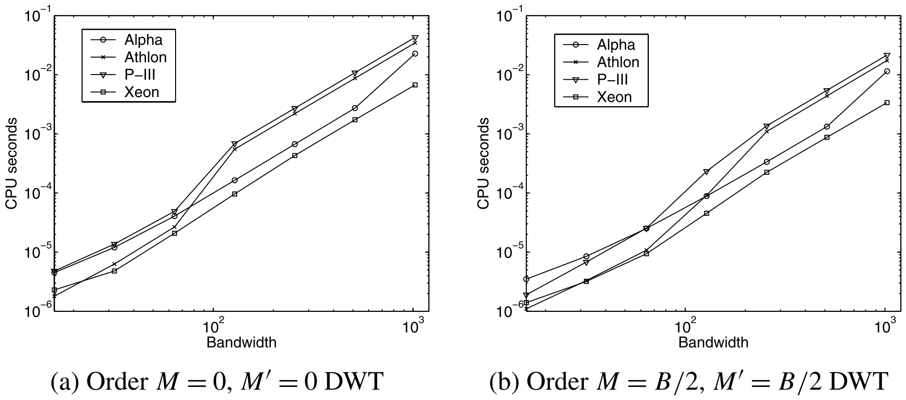


Fig. 3 Average CPU runtime for a Wigner Transform on four different platforms: The 833 MHz Alpha, the 1.66 GHz Athlon, the 800 MHz P-III, and the 2.4 GHz Xeon.

Table 3 Average CPU runtimes (in seconds) of the forward and inverse $M = 0$, $M' = 0$ DWT on the Xeon.

Bandwidth	16	32	64	128
Forward DWT	2.3000e-06	4.8000e-06	2.0900e-05	9.6200e-05
Inverse DWT	1.1000e-06	5.2000e-06	2.2500e-05	9.1200e-05
Bandwidth		256	512	1024
Forward DWT		4.3070e-04	1.7334e-03	6.7300e-03
Inverse DWT		4.2550e-04	1.6697e-03	6.7200e-03

10^{-5} seconds to do a DWT. Of course, this is just for a single DWT. Speed becomes a more pertinent issue when considering the full transform.

Now, the astute reader may ask the reasonable question, “When doing a bandwidth B DWT, why sample at the Chebyshev points, at twice the bandwidth? Why not sample at the B Gaussian points?” We looked into that question with *Matlab*, and our results are shown in Tables 4 and 5. In *Matlab*, we implemented our standard experiment (described at the beginning of this section), but this time iterated only 100 times. The tables show that the errors are comparable (and small) for both sets of points.

So why not stick with Gaussian quadrature for the DWT? Well, since the Wigner-d functions satisfy a three-term recurrence, we want the option of applying the techniques of Driscoll-Healy [6] and Driscoll-Healy-Rockmore [7] to yield a $O(B \log^2 B)$ DWT. Their techniques require sampling at twice the bandwidth — at the Chebyshev points. However, as noted previously [18], the “classic” Driscoll-Healy algorithm, in certain circumstances, exhibits numerical instabilities.

To investigate the stability of the discrete Wigner transform, as computed by application of the Driscoll-Healy method, we implemented in *Matlab* the following procedure outlined in Figure 4. When applying the “divide-and-conquer” DH algo-

Table 4 Average (first row) and relative (second row) errors of the discrete Wigner transform, as implemented in *Matlab*, at various bandwidths B and orders m_1, m_2 over 100 iterations. For each problem size B, a B-point Gaussian quadrature was used.

Orders	$B=64$	$B=128$	$B=256$	$B=512$	$B=1024$
$m_1=0, m_2=0$	3.0792e-13	8.7333e-13	1.3840e-12	5.7836e-12	2.5360e-11
	2.3122e-11	9.8438e-11	4.4669e-10	3.1174e-09	3.9491e-08
$m_1=B/2, m_2=0$	3.8430e-14	9.2385e-14	1.8135e-13	4.7824e-13	1.1782e-12
	4.2521e-12	5.8872e-12	8.0913e-11	7.9856e-11	4.6796e-10
$m_1=m_2=B/2$	1.3247e-13	5.9881e-13	7.6512e-13	3.5332e-12	5.5265e-10
	9.6054e-12	8.3802e-11	1.5318e-10	2.3103e-09	2.5862e-08

Table 5 Average (first row) and relative (second row) errors of the discrete Wigner transform, as implemented in *Matlab*, at various bandwidths B and orders m_1, m_2 over 100 iterations. For each problem size B, the functions were sampled at the $2B$-many Chebyshev points (i.e., the "usual" grid).

Orders	$B=64$	$B=128$	$B=256$	$B=512$	$B=1024$
$m_1=0, m_2=0$	4.7564e-14	4.4736e-14	2.3443e-13	3.9389e-13	1.4254e-12
	3.6413e-12	7.4549e-12	1.0121e-10	2.6971e-10	1.5577e-09
$m_1=B/2, m_2=0$	9.6901e-15	2.1207e-14	4.0782e-14	9.5098e-14	1.9725e-13
	5.1438e-13	1.1691e-12	1.3858e-11	2.3071e-11	7.4163e-11
$m_1=m_2=B/2$	9.5791e-15	3.9630e-14	1.0923e-13	2.0820e-13	4.1624e-10
	3.4753e-13	4.3671e-12	2.8238e-11	1.0873e-10	7.2033e-09

rithm, we "divided" the problem size twice. That is to say, we reduced a problem of size N to four problems each of size $N/4$. Our results are in Table 6.

The behavior reflected in the results in Table 6 is similar to that reported in [18]. In that work, shifted Legendre polynomials (generated by repeated application of the three-term recurrence) were seen to be the culprit. In the present case, we have "shifted Wigner polynomials" to blame for the instability.[1] One cannot shift "too far" with them, at least from certain locations. In Figure 5 we show a log-plot of the average error from a "per coefficient" perspective. For a DH-based DWT, at bandwidth $B=64$ and orders $m_1=m_2=32$, there are 32 coefficients to compute. After 100 iterations (i.e., the procedure outlined above), what was the average error, over those 100 iterations, of each coefficient? That is what Figure 5 shows.

It is possible that, as in [18], a hybrid-DH algorithm could be applied to stably compute a DWT at all orders. However, given the memory constraints on the relatively straightforward algorithm we present, the problem size where such a hybrid-

[1] Actually, we are not being entirely fair to the shifted Wigner polynomials. The problems are not with the polynomials themselves. Rather, things go awry when one takes their discrete cosine transform. When we instead tried the DH-strategy of "dividing" without "conquering" in the procedure we outlined above, we did not see any instabilities. Errors were on the order of 10^{-11} and smaller. This is the good news. The bad news is that one has an algorithm with the same complexity as the direct approach.

```
Fix bandwidth $B$, and orders $m_1$ and $m_2$
Let $m = \max(|m_1|, |m_2|)$
for $i = 1$ to 100 do
    1. Generate set of random Wigner $d$-coefficients $\hat{f}^k_{m_1,m_2}$, normally
    distributed with mean 0 and standard deviation 1, for
    $k = m, \ldots, B-1$
    2. Synthesize the function
        $$f(\cos\beta_l) = \sum_{k=m}^{B-1} \hat{f}^k_{m_1,m_2} d^k_{m_1,m_2}(\cos\beta_l)$$
    where $l = 0, \ldots, 2B-1$, and $\beta_l = \frac{\pi(2l+1)}{4B}$
    3. Apply the DH-based DWT algorithm to the synthesized function,
    obtaining coefficients Wigner $d$-coefficients $\hat{g}^k_{m_1,m_2}$.
    4. Compute the error as
        $$\max_k ||\hat{f}^k_{m_1,m_2} - \hat{g}^k_{m_1,m_2}||$$
end
Finally compute the average and relative errors over all trials.
```

Fig. 4 Outline of stability experiment for the DWT.

Table 6 Average (first row) and relative (second row) errors of a DH-based discrete Wigner transform, as implemented in *Matlab*, at various bandwidths B and orders m_1, m_2. Two recursive "splits" in the DH-implementation were done in all cases.

Orders	$B = 64$	$B = 128$	$B = 256$
$m_1 = 0, m_2 = 0$	7.2866e-14	1.1693e-13	2.8986e-13
	1.9497e-11	1.0186e-11	9.5298e-11
$m_1 = B/2, m_2 = 0$	2.5139e-05	2.4748e+08	6.0021e+34
	2.2400e-04	5.7008e+09	1.7985e+36
$m_1 = m_2 = B/2$	6.3073e+06	1.0381e+33	3.8828e+85
	3.6475e+07	4.9463e+33	3.3584e+86

DH algorithm would start to "win" over our algorithm is considerably beyond our current resources to investigate.

Finally, that same astute reader may ask, "Since the Wigner-d functions are trigonometric polynomials, couldn't the method of Dilts [5] (called the "semi-naive" algorithm in [18]) be used? Couldn't the sum (3.5) be evaluated in the cosine domain?" Yes, it could, but why we chose not to go that route will be given in the next section. In brief, at the point at which we can begin to achieve real-time gains in performance by working in the cosine domain, the problem becomes intractable due to hardware limitations.

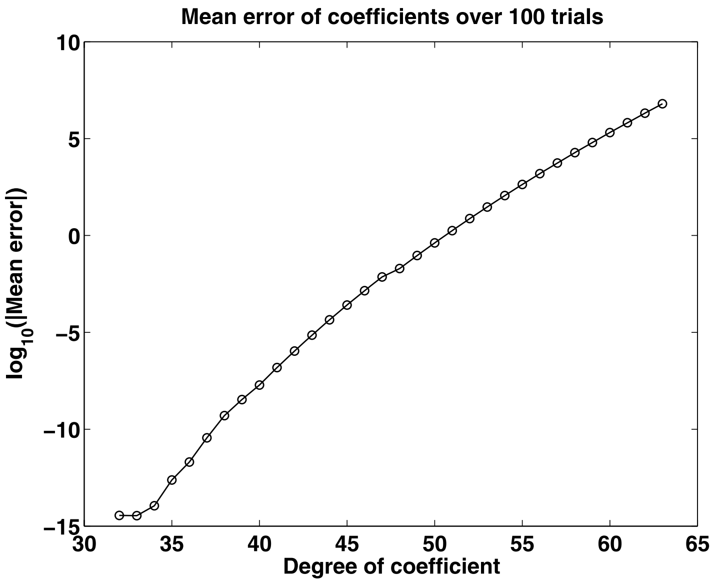


Fig. 5 DH-based DWT: Mean error of coefficients over 100 trials at bandwidth $B = 64$ and orders $= m_1 = m_2 = 32$.

5.2 Nice 'n $SOFFT$

The basic structure of the experiment described in Figure 2 was preserved when doing a Fourier transform over the entire group.

(1) Choose a bandwidth B.
(2) Generate random Fourier coefficients, $\{\hat{f}^l_{MM'}\}$ over the appropriate ranges of l, M and M'. There are $\frac{B}{3}\left(4B^2 - 1\right)$ many coefficients. Real and imaginary parts of the coefficients are values uniformly distributed between -1 and 1.
(3) Compute the inverse $SOFFT$ of the above coefficients, resulting in values of the function $f(\alpha, \beta, \gamma)$ sampled on the equiangular $2B \times 2B \times 2B$ grid.
(4) Compute the forward $SOFFT$ of the above sample values, resulting in a new set of Fourier coefficients, $\{\hat{g}^l_{MM'}\}$.
(5) Compute the error

$$\max_{l,M,M'} \left\| \hat{f}^l_{MM'} - \hat{g}^l_{MM'} \right\| .$$

(6) Repeat Steps (2) through (5) ten times.
(7) Compute the average absolute and relative errors, and their standard deviations, over the ten trials.

Note that by going from spectral to spatial to spectral representations, we ensure that we really are dealing with bandlimited functions $f \in L^2(SO(3))$. Furthermore, the Fourier coefficients in Step (2) are randomly generated without the constraint

Table 7 Average absolute and relative errors, over ten iterations, of **Sym**, when run on the Xeon.

Band-limit	8	16	32	64	128
Average Error	1.6147e-12	5.7296e-12	1.5481e-11	1.1007e-10	7.0047e-09
Std Dev	1.4289e-13	7.4973e-13	1.4024e-12	1.4696e-10	1.8530e-08
Relative Error	1.4330e-11	1.0247e-10	8.9718e-10	5.3790e-09	4.1743e-07
Std Dev	1.4069e-11	4.9962e-11	6.6493e-10	4.5457e-09	8.7350e-07

of ensuring that the resulting sample values should be strictly real, e.g., as is done in [20].

We implemented four fundamentally different versions of $SOFFT(f)$, the full $SO(3)$ Fourier transform, with a variant or two thrown in. The full transform may be treated as two components: The "regular" FFT portion [Equations (3.2)–(3.3)], and the DWT portion [Equation (3.4)]. With this in mind, here are the four versions:

- **Plain**: The FFT portion is computed via the home-grown FFT available as part of *SpharmonicKit* [31]. The DWT is evaluated for all legal M and M', computing the Wigner-d function values on the fly as needed.
- **Sym**: As in **Plain**, the FFT portion is computed via the home-grown FFT. However, we take advantage of the symmetries enjoyed by the Wigner-d functions and get as much mileage out of each set as possible, e.g., the Wigner-d functions needed for orders $M = 4$, $M' = 14$ may also be used for orders $M = 14$, $M' = 4$, and $M = 4$, $M' = -14$, and so on.
- **FFTW**: The DWT is evaluated as in **Sym**, but the FFT portion is computed via the routines in *FFTW*.
- **FFTW-PC**: Just like **FFTW**, except that *all* the Wigner-d functions are precomputed in advance.

These different implementations all exhibit the same stability behavior. Their real differences become apparent when the question of CPU time is addressed.

Stability. The results of the above experiment, when running **Sym** on the Xeon, are given in Table 7. Results on the other platforms are comparable. Quite clearly, at the bandwidths tested, the implementation is quite stable.

Now, the reader will note that the full transform is run only through bandwidth $B = 128$. The reason for this is quite simple: We ran out of memory! Or, to be more precise, the computer ran out of memory. For each doubling of the bandwidth, memory requirements increase eight-fold. Suppose the C code uses the data type **double**, which is 8 bytes long. If we make the reasonable assumption that there should be sufficient RAM to contain both the input data, all $(2B)^3$ sample values (which we assume are complex), and the Fourier coefficients (also complex), all $\frac{B}{3}\left(4B^2 - 1\right)$ of them, then for problem size $B = 128$, we would need nearly 300 megabytes of RAM. For $B = 256$, requirements would blossom to nearly 2.4 gigabytes! And we are not even considering the memory necessary for temporary storage, e.g., matrix transposing. So this is the downside. The upside is that we cannot even attempt to

Table 8 Average CPU runtime (in seconds) for the forward (first row) and inverse (second row) $SO(3)$ transform on the Xeon.

Band-limit	8	16	32	64	128
Plain	2.8800e-03	3.2120e-02	3.2610e-01	3.1180e+00	3.5070e+01
	2.9100e-03	3.1650e-02	2.9960e-01	3.1240e+00	3.5771e+01
Sym	1.3200e-03	1.5390e-02	1.6850e-01	1.4890e+00	1.6631e+01
	1.5300e-03	1.3230e-02	1.2510e-01	1.1970e+00	1.3544e+01
FFTW	5.2000e-04	6.7900e-03	7.2200e-02	7.2900e-01	8.2510e+00
	5.6000e-04	7.0700e-03	7.6300e-02	7.9700e-01	8.8970e+00
FFTW-PC	2.9000e-04	4.9600e-03	5.3900e-02	5.2200e-01	
	3.2000e-04	5.4400e-03	5.8500e-02	5.6000e-01	

compute a Fourier transform on the full group at a bandwidth where the numerical stability of the DWT we know (by the results of Section 5.1) to be suspect.

Running Times. The four variants of implementations were run on all four platforms. At the smaller bandwidths, $B = 8$, 16 and 32, in order to obtain as accurate a timing result as possible, the experiment described at the beginning of this section was slightly modified. First a random set of coefficients was generated. Then the stopwatch would be turned on, and the inverse transform would be performed (on that one set of coefficients) X-many times, then the stopwatch would be turned off. This would be repeated for the forward transform, using the random sample points just obtained. For $B = 8$, 16, $X = 1000$, and for $B = 32$, $X = 100$. The remaining bandwidths used $X = 10$.

Furthermore, the different machines had different amounts of RAM. We ran the tests for as large as possible on the particular machine. And it is always possible the amount of RAM affected performance. Still, we believe our results are fairly indicative of what might be expected on other platforms.

Finally, using *FFTW* involves first generating a "plan" before carrying out the actually Fourier transforms. In a plan, the user specifies parameters such as the size of the (multi-dimensional) transform, whether or not the transform should be in-place or not, and if the transform is multi-dimensional, how the algorithm should "skip" through the data. Prior to executing "for real," *FFTW* determines how to most efficiently perform the specified plan on that specific platform. The responsibility of determining how hard *FFTW* should try to find the optimal transform is up to the user. In our tests, the *FFTW* level we used was `FFTW_MEASURE`. In addition to this, we freely admit the possibility that more efficient plans than ours is possible.

In Figure 6 we plot the average CPU runtime (in seconds) of the forward $SO(3)$ transform. In order to provide a little more quantitative "meat," in Table 8 we give the average runtimes of the $SO(3)$ transform on the Xeon, both forward and inverse.

As is no surprise, **FFTW-PC** performed fastest on all four platforms, and **Plain** the slowest, although by how much might not be apparent in Figure 6. Therefore, in Figure 7 we plot the ratios of the runtimes for **Sym**, **FFTW**, and **FFTW-PC** versus **Plain**.

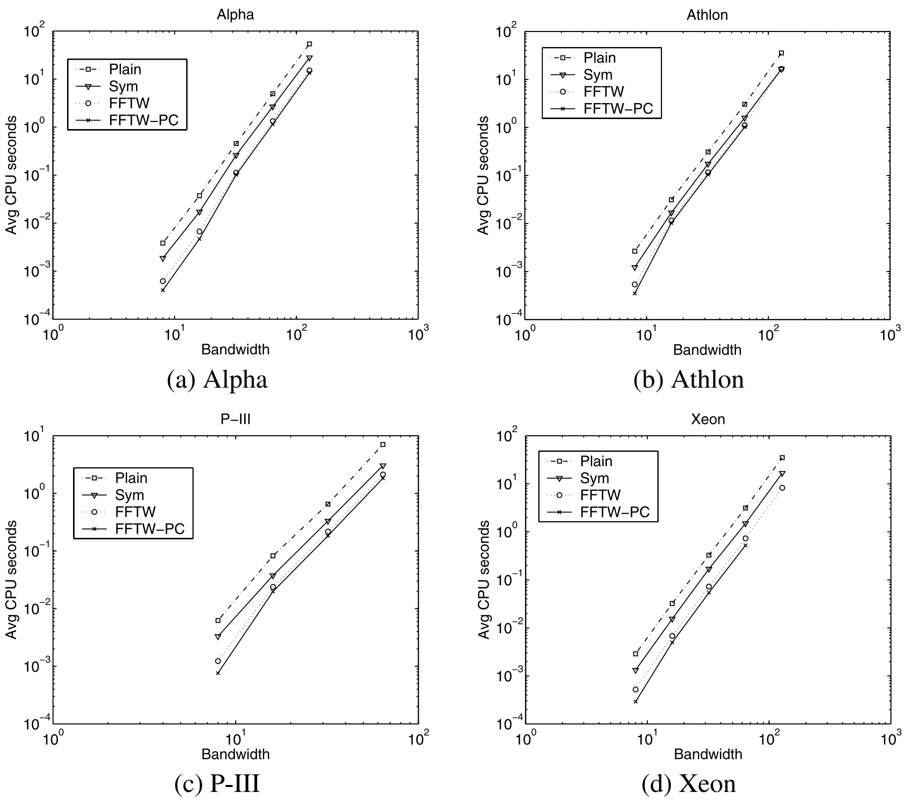
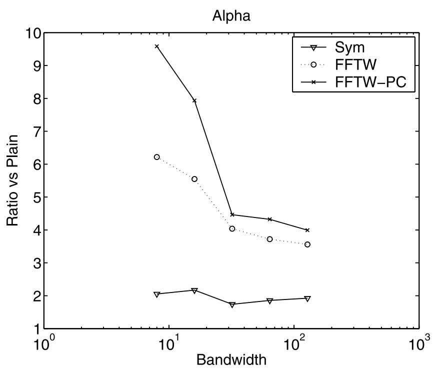


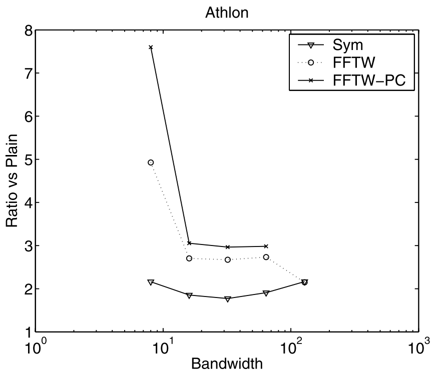
Fig. 6 Average CPU runtime for the different forward $SO(3)$ transforms.

Figure 7 appears to confirm the following rules of thumb: Taking advantage of the symmetries of the Wigner-ds speeds up the runtime by roughly a factor of 2 over **Plain**, and *FFTW* plus symmetries buys a factor of 4 speed-up. However, precomputing the Wigner-ds plus *FFTW* plus symmetries yields a noticeable, though not particularly dramatic, gain in performance only on the Xeon. A similar less-than-spectacular gain was observed when we assumed that the function was strictly real-valued. This assumption introduces an additional symmetry in the coefficients which can be used for computational advantage.

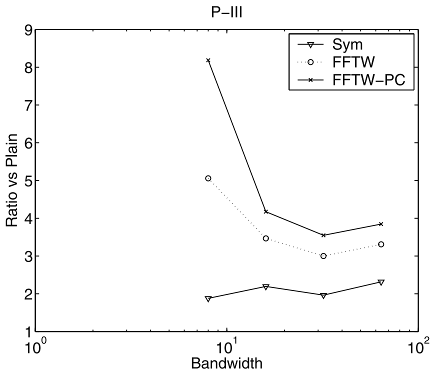
These results were somewhat disappointing, and they very much relate to why the semi-naive algorithm was not employed in performing the DWT. This algorithm depends in part on precomputed data, to wit the DCT (discrete cosine transform) coefficients of the Wigner-d functions. In fact, a semi-naive version of the DWT was coded up and tested in a full $SO(3)$ transform, but the runtimes were basically no different from those obtained with **FFTW-PC**, which themselves are not significantly different from **FFTW**. Indeed, this behavior made us skeptical that a Driscoll-Healy based algorithm, which also depends on precomputed data, would offer much more efficiency.



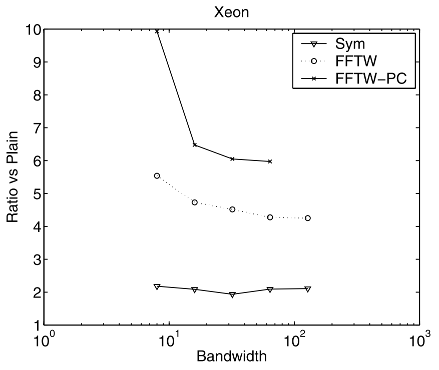
(a) Alpha



(b) Athlon



(c) P-III



(d) Xeon

Fig. 7 Ratios of average CPU runtime for the different forward $SO(3)$ transforms.

We are not certain why we are failing to see much improvement in the runtimes. However, we do believe we can identify some of the causes. One is memory, or lack thereof. The problem sizes we are able to run at may be too small to see the gains the asymptotic analyses predict. And this relates to the hidden constant in these analyses — the overhead. An integral part of the algorithm, as far as an actual implementation is concerned, is the matrix transpose. After all, not only are the arrays large, but they must be transposed. This is a significant portion of the runtime. On the Xeon, more than 40% of the time spent doing a bandwidth $B = 64$ forward $SO(3)$ transform is spent transposing matrices. For $B = 32$, this figure approaches 75%! This encourages us to believe that B would have to be pretty large before an algorithm like semi-naive would present a big win over, say, **FFTW-PC**. Indeed, it could be argued that precomputing does not yield such significant improvement, either.

We tried many variations of the code, to reduce the number of matrix transpositions, e.g., having a different ordering the data prior to transforming; letting FFTW do the matrix transposing by means of a cleverly chosen plan. For us, the performance gained in doing these things was barely perceptible.

Now, we do not mean to imply that this precludes the possibility of an optimal semi-naive or Driscoll-Healy like $SO(3)$ transform at bandwidths within our realm. Rather, given how our code is structured, we do not think it is possible. However, the reader should keep in mind that the "slow" algorithms we have tested are, in terms of absolute CPU time, still pretty fast, and hence may be of real use.

6 An Application: Correlation

Given two functions on the sphere, f and h, and the knowledge that h is a rotated version of f, i.e., $f = \Lambda(g)h$ for some $g \in SO(3)$, how does one find that g? Phrased another way, we have a pattern f, and we wish to identify its latitude, longitude, and orientation, on the sphere. This can be accomplished by correlating the two functions:

$$C(g) = \int_{S^2} f(\omega)\, \overline{\Lambda(g)h(\omega)}\, d\omega \tag{6.1}$$

and finding the g that maximizes the above integral. This has a number of useful applications, in such diverse areas as molecular biology [4, 21], and 3-D shape-matching [19].

Before we show how one may efficiently correlate two functions defined on the sphere, we will take a momentary and minor detour to discuss one way spherical harmonics are used in 3-D shape-matching. The particular application is in developing rotation invariant descriptors [19].

First of all, one of the well known properties of the (Euclidean) Fourier transform is that the *magnitudes* of the Fourier coefficients are invariant under translation, since a translation in the signal manifests itself as a phase shift in the coefficients. For Fourier transforms on the sphere, where the action is now rotation, we have the next best thing. Let f_l denote the the l^{th} frequency component of $f \in L^2(S^2)$:

$$f_l(\omega) = \sum_{|m| \le l} a_{lm} Y_l^m(\omega) .$$

The quantity $|f_l(\omega)|$ is invariant under rotation. Let $P(f)$ (for "power") denote the set

$$P(f) = \{|f_0|, |f_1|, |f_2|, \ldots\} .$$

Alright, then. How is this applied to 3-D shape-matching? We begin with the assumption that we have a 3-D model of an object. Consider concentric spheres, of increasing radius, centered at the object's center of mass. The intersection of a sphere of radius r with the object defines a function $f(r,\omega)$ on the sphere. For each object, then, we form the set

$$\{P(f(r_0,\omega)), P(f(r_1,\omega)), P(f(r_2,\omega)), \ldots\}$$

where $r_0 < r_1 < r_2 < \ldots$. This provides a rotation-invariant representation of the 3-D object. These representations can be searched over, and there you have 3-D shape matching. The Princeton Shape Retrieval and Analysis Group [30] is carrying out exciting work in this area.

And now, back to our story. How do we maximize (6.1)? How do we identify that rotation g which aligns h to f, i.e., $f = \Lambda(g)h$?

Instead of undertaking the time-consuming task of evaluating $C(g)$ for all possible rotations, we may efficiently determine the maximum g by means of the FFT on $SO(3)$. We develop this as follows. First of all, since it is the case that $f, h \in L^2(S^2)$, we have their Fourier expansions:

$$f(\omega) = \sum_l \sum_{|m|\le l} a_{lm} Y_l^m(\omega)$$

$$h(\omega) = \sum_l \sum_{|m|\le l} b_{lm} Y_l^m(\omega) \ .$$

We wish to find the $g \in SO(3)$ which maximizes

$$C(g) = \int_{S^2} f(\omega)\ \overline{\Lambda(g)h(\omega)}\ d\omega \ .$$

Let us write things out in gory detail. We begin with

$$\begin{aligned} C(g) &= \int_{S^2} f(\omega)\ \overline{\Lambda(g)h(\omega)}\ d\omega \\ &= \int_{S^2} \left[\sum_l \sum_{|m|\le l} a_{lm} Y_l^m(\omega) \right] \overline{\left[\Lambda(g) \sum_{l'} \sum_{|m'|\le l'} b_{l'm'} Y_{l'}^{m'}(\omega) \right]} d\omega \\ &= \sum_l \sum_{|m|\le l} \sum_{l'} \sum_{|m'|\le l'} a_{lm}\ \overline{b_{l'm'}} \int_{S^2} Y_l^m(\omega)\ \overline{\Lambda(g) Y_{l'}^{m'}(\omega)}\ d\omega \ . \end{aligned} \tag{6.2}$$

At this point, we can achieve some simplification. The integral in (6.2) equals 0, *unless* $l = l'$. Therefore, we can remove a summation:

$$C(g) = \sum_l \sum_{|m|\le l} \sum_{|m'|\le l} a_{lm}\ \overline{b_{lm'}} \int_{S^2} Y_l^m(\omega)\ \overline{\Lambda(g) Y_l^{m'}(\omega)}\ d\omega \ . \tag{6.3}$$

Recall that under rotation, a spherical harmonic of degree l is transformed into a linear combination of spherical harmonics of the same degree. In terms of the Wigner-Ds, we can express this as:

$$\Lambda(g) Y_{lm}(\omega) = \sum_{|k|\le l} Y_l^k(\omega)\ D_{km}^l(g) \ . \tag{6.4}$$

Therefore, we may write (6.3) as:

$$\begin{aligned} C(g) &= \sum_l \sum_{|m|\le l} \sum_{|m'|\le l} a_{lm}\ \overline{b_{lm'}} \int_{S^2} Y_l^m(\omega)\ \overline{\sum_{|k|\le l} Y_l^k(\omega)\ D_{km'}^l(g)}\ d\omega \\ &= \sum_l \sum_{|m|\le l} \sum_{|m'|\le l} \sum_{|k|\le l} a_{lm}\ \overline{b_{lm'}}\ \overline{D_{km'}^l(g)} \int_{S^2} Y_l^m(\omega)\ \overline{Y_l^k(\omega)}\ d\omega \ . \end{aligned} \tag{6.5}$$

We are nearly there. Note that the integral in (6.5) is equal to 0 *unless* $k = m$. This being the case, and using two of the many symmetries the Wigner-D functions satisfy, we can zap one of the summations:

$$\begin{aligned}
C(g) &= \sum_{l} \sum_{|m|\le l} \sum_{|m'|\le l} a_{lm}\, \overline{b_{lm'}}\, \overline{D^l_{mm'}(g)} \\
&= \sum_{l} \sum_{|m|\le l} \sum_{|m'|\le l} a_{lm}\, \overline{b_{lm'}}\, (-1)^{m'-m} D^l_{-m-m'}(g) \\
&= \sum_{l} \sum_{|m|\le l} \sum_{|m'|\le l} a_{l-m}\, \overline{b_{l-m'}}\, (-1)^{m-m'} D^l_{mm'}(g)\,. \qquad (6.6)
\end{aligned}$$

That's it. That's the recipe.

So, if f and h are band-limited, combining the Fourier coefficients of f and h thusly, as prescribed in (6.6), and then taking the inverse $SO(3)$-Fourier transform of the result, we can efficiently evaluate the correlation $C(g)$ of f and h, at a whole slew of g's. Then we can easily find which rotation g maximizes $C(g)$. In other words, if we knew in advance that h was some rotated copy of f, we would know which g satisfies $f = \Lambda(g)h$.

In the next section, we will show some examples of correlating via the technique just described.

7 An Example: Pattern Matching on S^2

In order to investigate the viability of pattern-matching on the sphere via FFTs on $SO(3)$, we applied the technique described in the previous section on a variety of images. Those results, which are presented in this section, are quite promising.

In Figure 8, we show the signal, pattern, and the difference between the signal and aligned pattern. The difference image is shown on the "unrolled" sphere, with the north pole being the top horizontal line, and the south pole the bottom. The bandwidth is $B = 128$. The total CPU running time, from computing the spherical coefficients of both signals through finding the optimal rotation, on the Xeon, was about 10 seconds. We used **FFTW** for the inverse $SO(3)$ Fourier transform, for this and all the examples which follow.

The original data was a 256×256 square image, i.e., sampled in the plane. We simply interpreted the samples as instead living on the equiangular 256×256 grid on the sphere. We then took the forward and inverse S^2 Fourier transform, resulting in samples of a truly bandlimited signal. To rotate, we "massaged" the S^2 Fourier coefficients with the Wigner-D functions, as prescribed by (2.5).

In Figure 8, we also plot the values of the parameters α, β, and γ which yield the maximum value of $C(g)$, as a function of the maximum degree of Y_l^ms used in the correlation. The plot is only through degree $l = 9$, because by that point, we know the three angles as well as we are going to. Indeed, in this particular example, correlating using only up to the degree 2 spherical harmonics is sufficient to yield the final answer. Higher degree harmonics do not improve the results. (Later examples

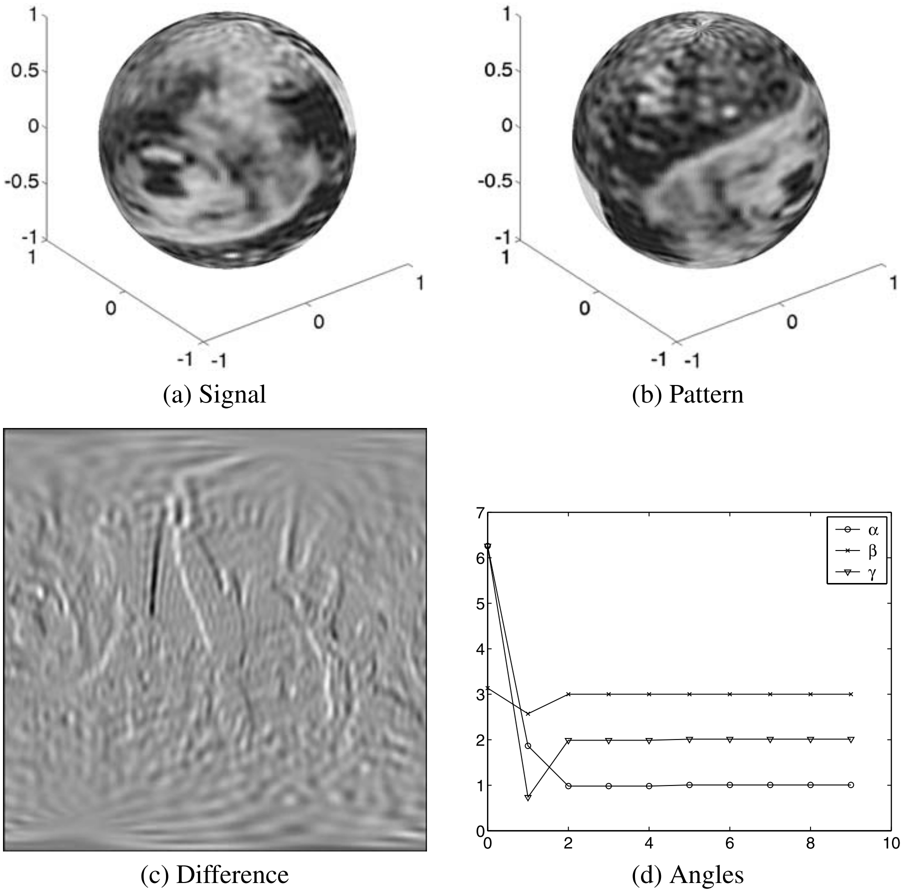


Fig. 8 Rotating the pattern to match the signal. The bandwidth is $B = 128$. In the lower left we show the difference between the signal and aligned pattern on the unrolled sphere (the South Pole is at the bottom, the North Pole at the top). The lower right panel shows the rotation parameters α, β, and γ which yield the maximum value of $C(g)$, as a function of the maximum degree of Y_l^ms used in the correlation.

will show this is not always the case. Higher degree harmonics are at times necessary.) Qualitatively, we can view this as indicating the resolution in the angular features needed to accurately find the pattern in the signal.

Before going further, we should clarify what we mean by "maximum degree of Y_l^ms." A band-limited function $f \in L^2(S^2)$ has the following expansion:

$$f(\omega) = \sum_{l=0}^{B-1} \sum_{|m|\le l} \hat{f}_l^m \; Y_l^m(\omega) \; .$$

If we say that the maximum degree of Y_l^m used was, say, 5, then we are setting equal to 0 all those coefficients $\hat{f}_l^m$ where $l \geq 6$. In the correlation routine, this zeroing is done *prior* to using the S^2 coefficients to construct, via (6.6), the Fourier expansion of $C(g)$, and hence prior to taking its inverse $SO(3)$ Fourier transform. Note that it is still a bandwidth B inverse transform being performed. Doing this can be viewed in one of two ways, as either correlating smoothed versions of the two spherical functions, or taking the inverse $SO(3)$ transform of a smoothed version of $C(g)$, e.g.,

$$C(g) = \sum_{l=0}^{5} \sum_{|m|\le l} \sum_{|m'|\le l} a_{l-m} \, \overline{b_{l-m'}} \, (-1)^{m-m'} D^{(l)}_{mm'}(g) \, ,$$

which amounts to the same thing.

Before going on to the next example, there is one more point we wish to make. The average relative error between the signal and aligned pattern is 0.0212. This may seem unacceptably large to some, but bear in mind what we are doing. We wish to find the rotation $g \in SO(3)$ which maximizes $C(g)$. However, the gs we are trying are points on a particular $2B \times 2B \times 2B$ grid. The "true" g, the rotation which satisfies $f(\omega) = \Lambda(g)h(\omega)$, may not lie on this grid. In this case, the g we find is, in some sense, the "next best g."

In another experiment we performed, using the same bandlimited image as in the first example, the true angles were very close to sample locations on the $2B \times 2B \times 2B$:

$$\begin{aligned} \alpha &= 1.006291 \\ \beta &= 3.000466 \\ \gamma &= 2.012582 \, . \end{aligned}$$

The angles the correlation routine found, which lie *on the grid*, were

$$\begin{aligned} \alpha &= 1.006291396852981 \\ \beta &= 3.000466421104314 \\ \gamma &= 2.012582793705961 \, . \end{aligned}$$

The average relative error in this case, between the aligned pattern and signal, was 2.5259×10^{-6}. So the error decreases, the closer the true g lies on the grid point.

We therefore have the following completely natural situation. Correlating a signal with itself, i.e., $f(\omega) = \Lambda(e)h(\omega)$, where e is the identity element of $SO(3)$, will **not** yield $\alpha = \beta = \gamma = 0$, but rather $\beta = \pi/(4B)$, where B is the bandwidth, and α and γ such that $\alpha + \gamma = 2\pi$. (At least, this is what our code gives us!) The behavior of α and γ might seem reasonable, but what about β? Recall where we are sampling: $\beta_j = \pi(2j+1)/(4B)$. When $j = 0$, we have $\beta = \pi/(4B)$. Bingo.

In our second example, we stereographically projected MR images onto the sphere, sampling on the $B = 128$ equiangular grid. Interpolation to the grid was done using *Matlab*. Results are shown in Figure 9. The reader should compare these results with Figure 8, noting the minimum degree Y_l^m required in order to achieve the optimal correlation. If nothing else, this example shows that sometimes one has to go higher than degree 2 spherical harmonics in order to accurately align the pattern to the signal.

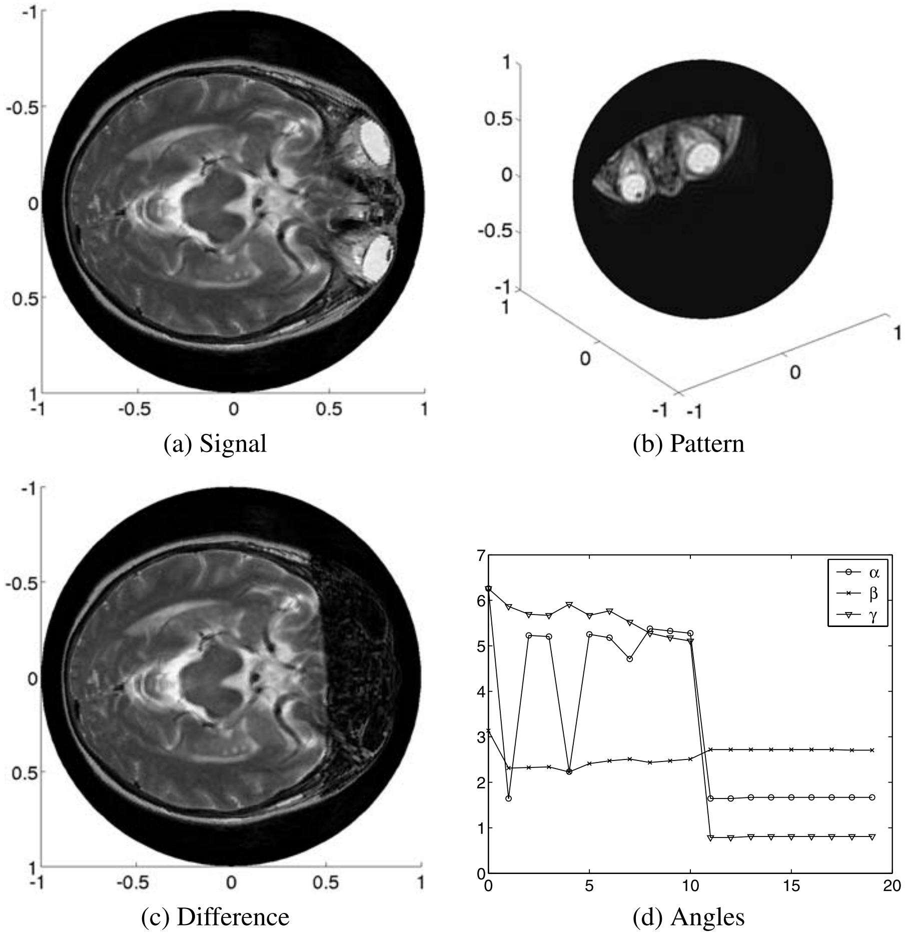


Fig. 9 Rotating the pattern to match the signal. The signal is shown from *beneath* the sphere, i.e., from our vantage point, the South Pole is at the center of the image. The bandwidth is $B = 128$. The pattern lives primarily in the Northern hemisphere. The difference image is rendered on the sphere where, as with signal, the South Pole at the center. The lower right panel shows the rotation parameters α, β, and γ which yield the maximum value of $C(g)$, as a function of the maximum degree of Y_l^ms used in the correlation.

The correlation examples we have so far seen are for problem sizes $B = 128$. However, wanting to do a bandwidth $B = 256$ correlation, we quickly run into the problem of insufficient memory (at least for the routines in our test suite). For $B = 256$, approximately 4.5 GB of RAM would be required, whereas a bandwidth $B = 128$ correlation requires “only” about 560 MB of RAM. This motivated us to do the next best thing. While the spherical functions we want to correlate may have bandwidth $B = 256$, there is no reason why the inverse $SO(3)$ Fourier transform needs to be run at that same bandwidth.

Figure 10 shows the results of our efforts. As with the previous experiment, the data was sterographically projected onto the sphere. The original spherical functions were of bandwidth $B = 256$, but the correlation function $C(g)$ was treated as having

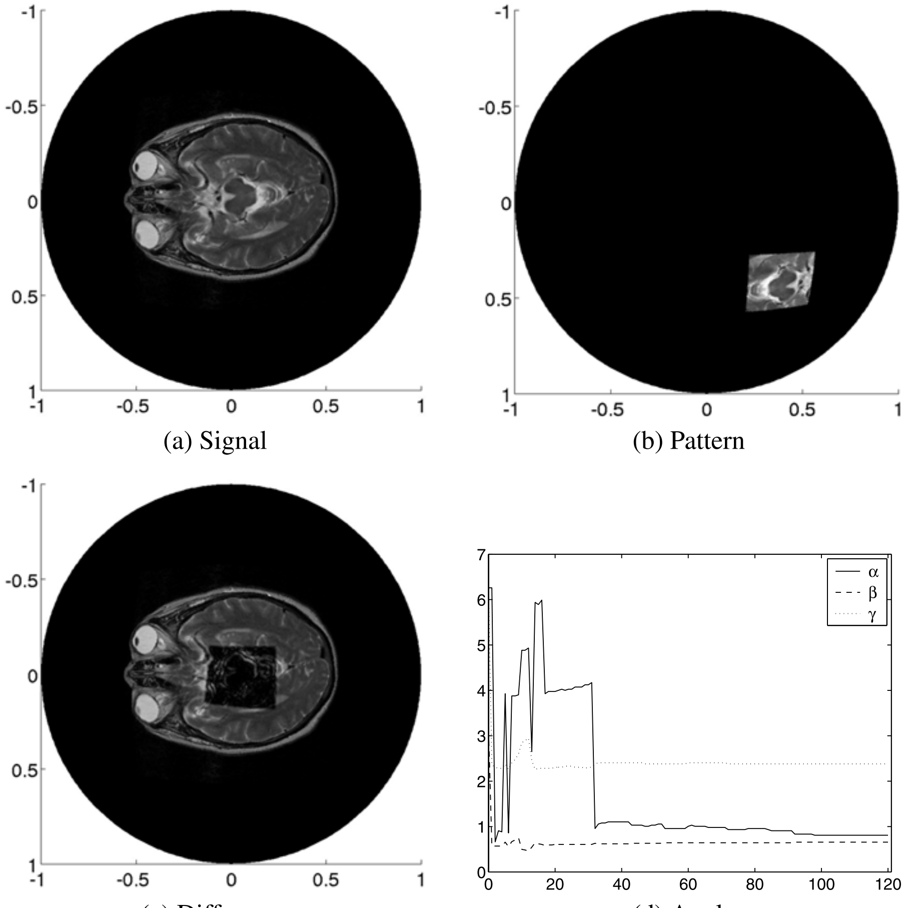


Fig. 10 Rotating the pattern to match the signal. The signal and pattern are shown from *beneath* the sphere, i.e., from our vantage point, the South Pole is at the center of the image. The difference image, the absolute value of the signal — the rotated pattern, is shown from the same vantage point. The bandwidth of the signal and pattern are both $B = 256$, but the bandwidth of the inverse $SO(3)$ transform was $B = 128$. The lower right panel shows the rotation parameters α, β, and γ which yield the maximum value of $C(g)$, as a function of the maximum degree of Y_l^ms used in the correlation.

bandwidth $B = 128$. Granted, we are effectively correlating smoothed versions of the original functions, but visually, the alignment we achieve is quite satisfactory.

The results of Figure 10 show some minor "jiggling," with regards to determining the rotation angles α, β, and γ, as functions of the maximum degree Y_l^m used. This is at least partially a result of the "true" rotation not being on the sample grid. The sharp boundary (a highly nonbandlimited feature) in the pattern is probably also contributing to the "jiggling."

We also tried out correlating geophysical data. The data we obtained was from the National Geophysical Data Center [25]. The particular dataset we used was the

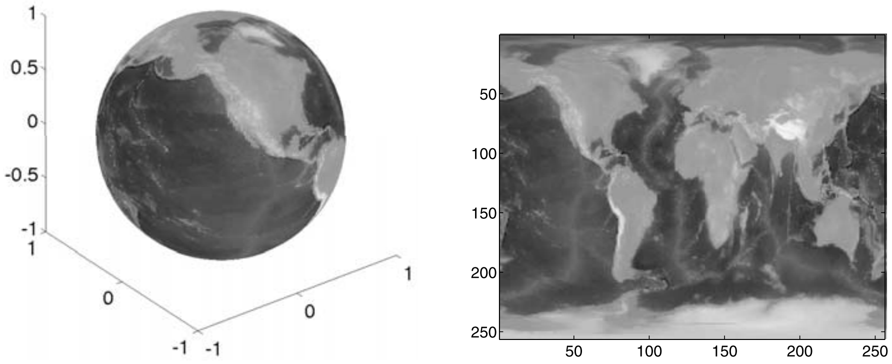


Fig. 11 The world: On S^2 and unrolled.

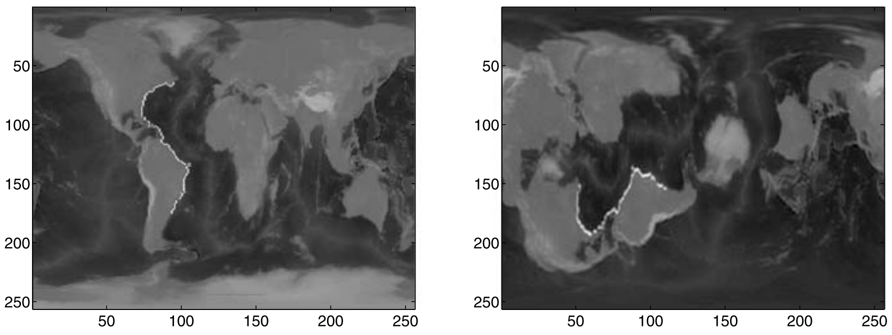


Fig. 12 The left panel shows the world and the correct location of the coastline (our pattern) as indicated by the highlighted portion of the coastline. The right panel shows the rotated world (our signal) and indicates (by highlighting) where correlation places the coastline.

ETOPO5 5-minute gridded elevation dataset [9]. Our goal was to see how well we could find a particular coastline.

To place the data on the sphere, we took the original, 5-minute gridded elevation data and, using *Matlab*, interpolated that data onto the $B = 128$ grid. Figure 11 shows the data both on the sphere and "unrolled."

To obtain the coastline we performed, admittedly very crudely, edge-detection on the "sent to the $B = 128$ grid" data, including manually adding and removing nonzero pixels until we obtained a visually satisfying coastline. In Figure 12 the panel on the left shows the world and where the coast belongs. The right panel shows the *rotated* world (this is our signal) and where correlation places the coastline. There is excellent agreement.

However, things do not work as well when we consider a smaller coastline. Consider Figure 13, which shows correlation placing the Atlantic coast of North America on the Pacific coast of South America. What might be happening? Any number of things. First of all, there is the fact that we were sloppy in defining the coast. Secondly, the coast is not bandlimited. (Neither is the original, nonrotated world, for that matter.) Indeed, given that the coast is basically a jagged line, we can even say that it

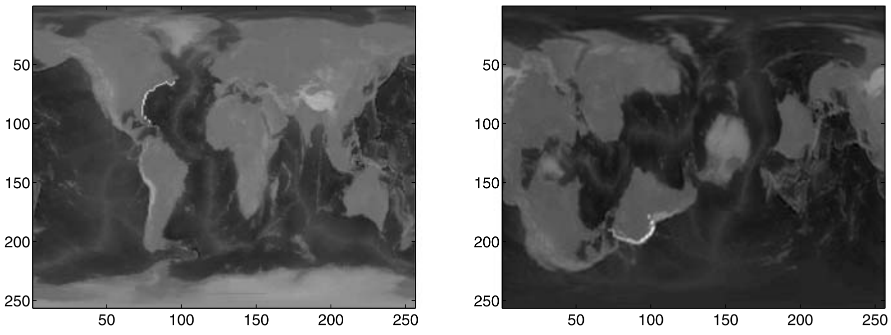


Fig. 13 The left panel shows the world and the correct location of a small (compare with Figure 12) piece of the coastline (our pattern) as indicated by the highlighted portion of the coastline. The right panel shows the rotated world (our signal) and indicates (by highlighting) where correlation places the coastline.

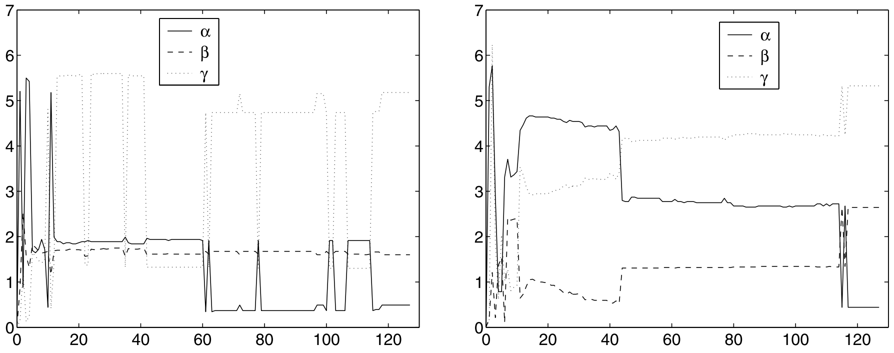


Fig. 14 The Euler angles as a function of the maximum degree of Y_l^m used in the correlation. The left panel corresponds to the successful alignment shown in Figure 12, and the right panel to the incorrect result shown in Figure 13.

is highly non-bandlimited. When taking its spherical Fourier transform, we are projecting the coast onto the space spanned by spherical harmonics of maximum degree $L = 127$. This degree might not be large enough to capture all of the coast's subtle features. As a result, correlation misplaces the coast.

We believe the reason why the results were good in Figure 12 is that there the coast is long enough that the subtleties being missed are not critical. The longer coast has sufficient "mass," if you will, to line up in only one spot, albeit, as seen in Figure 14, we might need the entire bandwidth in order to rotate it to the proper position on the globe.

On the bright side, the results of Figure 13 do establish the following corollary.

Corollary 1 *A bandwidth B greater than* 128 *is required for there to be at least hope of recovering Pangea. I.e.*,

$$\operatorname{Span}\left\{Y_l^m \mid 0 \le l < 128,\ |m| \le l\right\} \ne \textit{Pangea}\ .$$

8 Conclusions

We have developed and discussed an algorithm for the efficient computation of the Fourier transform of functions defined on $SO(3)$. Such efficient algorithms have uses in a wide vaiety of fields, such as searchable 3-D databases [30] and molecular biology [21].

This implementation differs from that of [28], as it allows easy use of efficient implementations of DWT algorithms, including those based on Dilts [5], and the more sophisticated orthogonal polynomial transform methods [7], which depend in part on the three-term recurrence the Wigner-d functions satisfy.

References

1. Biedenharn, L.C., Louck, J.D.: Angular Momentum in Quantum Mechanics. Addison-Wesley, Reading (1981)
2. Chirikjian, G.S., Kyatkin, A.B.: Engineering Applications of Noncommutative Harmonic Analysis: With Emphasis on the Rotation and Motion Groups. CRC Press, Boca Raton (2001)
3. Courant, R., Hilbert, D.: Methods of Mathematical Physics. Interscience Publishers, New York (1953)
4. Crowther, R.A.: The fast rotation function. In: Rossman, M.G. (ed.) The Molecular Replacement Method, pp. 173–178. Gordon and Breach, New York (1972)
5. Dilts, G.A.: Computation of spherical harmonic expansion coefficients via FFTs. J. Comput. Phys. **57**(3), 439–453 (1985)
6. Driscoll, J.R., Healy, D.: Computing Fourier transforms and convolutions on the 2-sphere (extended abstract). In: Proc. 34th IEEE FOCS (1989), pp. 344–349. Adv. in Appl. Math., vol. 15, pp. 202–250 (1994)
7. Driscoll, J.R., Healy, D., Rockmore, D.: Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs. SIAM J. Comput. **26**(4), 1066–1099 (1997)
8. Edmonds, A.R.: Angular Momentum in Quantum Mechanics. Princeton University Press, Princeton (1957)
9. Data Announcement 88-MGG-02: Digital relief of the Surface of the Earth. NOAA, National Geophysical Data Center, Boulder, Colorado (1988)
10. FFTW is a free collection of fast C routines for computing the Discrete Fourier Transform in one or more dimensions. It includes complex, real, symmetric, and parallel transforms, and can handle arbitrary array sizes efficiently; FFTW is available at www.fftw.org
11. Funkhouser, T., Min, P., Kazhdan, M., Chen, J., Halderman, A., Dobkin, D., Jacobs, D.: A search engine for 3D models. ACM Trans. Graph. 83–105 (2003)
12. Garcia, J., Valles, J.J., Ferreira, C.: Detection of three-dimensional objects under arbitrary rotations based on range images. Optic Express **11**(25), 3352 (2003)
13. Gentleman, W.M., Sande, G.: Fast Fourier Transforms—for fun and profit. Proc. 1966 Fall Joint Computer Conference AFIPS **29**, 563–578
14. Hansen, J.E.: Spherical Near-Field Antenna Measurements. Peter Peregrinus, London (1988)
15. Healy, D., Kim, P.: An empirical Bayes approach to directional data and efficient computation on the sphere. Ann. Stat. **24**(1), 232–254 (1996)
16. Healy, D., Hendriks, H., Kim, P.: Spherical deconvolution with application to geometric quality assurance. Technical Report, Department of Mathematics and Computer Science, Dartmouth College (1993)
17. Healy, D.M., Kostelec, P., Rockmore, D.: Towards safe and effective high-order Legendre transforms with applications to FFTs for the 2-sphere. Adv. Comput. Math. **21**(1–2), 59–105 (2004)
18. Healy, D., Rockmore, D., Kostelec, P., Moore, S.: FFTs for the 2-Sphere—Improvements and Variations. J. Fourier Anal. Appl. **9**(4), 341–385 (2003)
19. Kazhdan, M., Funkhouser, T., Rusinkiewicz, S.: Rotation invariant spherical harmonic representation of 3D shape descriptors. In: Kobbelt, L., Schröder, P., Hoppe, H. (eds.) Eurographics Symposium in Geometry Processing, pp. 167–175 (2003)

20. Kostelec, P.J., Maslen, D.K., Healy, D.M., Rockmore, D.N.: Computational harmonic analysis for tensor fields on the two-sphere. J. Comput. Phys. **162**, 514–535 (2000)
21. Kovacs, J.A., Wriggers, W.: Fast rotational matching. Acta Crystallogr. D Biol. Crystallogr. **58**(8), 1282–1286 (2002)
22. Maslen, D.: Efficient computation of Fourier transform on compact groups. J. Fourier Anal. Appl. **4**(1), 19–52 (1998)
23. Maslen, D., Rockmore, D., Generalized FFTs. In: Finkelstein, L., Kantor, W. (eds.) Proceedings of the DIMACS Workshop on Groups and Computation, June 7–10, 1995, pp. 183–237 (1997)
24. Maslen, D., Rockmore, D.: Separation of variables and the computation of Fourier transforms on finite groups, I. J. Am. Math. Soc. **10**(1), 169–214 (1997)
25. The National Geophysical Data Center (NGDC), located in Boulder, Colorado, is a part of the US Department of Commerce (USDOC), National Oceanic & Atmospheric Administration (NOAA), National Environmental Satellite, Data and Information Service (NESDIS). They are one of three NOAA National Data Centers, www.ngdc.noaa.gov
26. Nikiforov, A.F., Suslov, S.K., Uvarov, V.B.: Classical Orthogonal Polynomials of a Discrete Variable, Springer Series in Computational Physics. Springer, Berlin (1991)
27. Oppenheim, A.V., Schafer, R.: Digital Signal Processing. Prentice-Hall, Englewood Cliffs (1975)
28. Risbo, T.: Fourier transform summation of Legendre series and D-functions. J. Geod. **70**, 383–396 (1996)
29. Rokhlin, V., Tygert, M.: Fast algorithms for spherical harmonic expansions. SIAM J. Comput. **27**(6), 1903–1928 (2006)
30. The Princeton Shape Retrieval and Analysis Group, www.cs.princeton.edu/gfx/proj/shape/, whose goal is to "… investigate issues in shape-based retrieval and analysis of 3D models," has developed a 3D search engine as part of their work, shape.cs.princeton.edu
31. SpharmonicKit is a freely available collection of C programs for doing Legendre and scalar spherical transforms. Developed at Dartmouth College by S. Moore, D. Healy, D. Rockmore, and P. Kostelec, available at www.cs.dartmouth.edu/~geelong/sphere/
32. The *SOFT* Package is a freely available collection of C programs for doing Wigner-d transforms, as well as forward and inverse Fourier transforms of functions defined on the Rotation Group, $SO(3)$. The package also includes example routines for correlating functions defined on S^2, The package is available at www.cs.dartmouth.edu/~geelong/soft/
33. Varshalovich, D.A., Moskalev, A.N., Khersonskii, V.K.: Quantum Theory of Angular Momentum. World Scientific Publishing, Singapore (1988)
34. Vilenkin, N.J.: Special Functions and the Theory of Group Representations, Translations of Mathematical Monographs, vol. 22. Am. Math. Soc., Providence (1968)
35. Wandelt, B.D., Górski, K.M.: Fast convolution on the sphere. Phys. Rev. D **63**, 123002 (2001)
36. Wigner, E.P.: On matrices which reduce Kronecker products of representations of S.R. groups, unpublished (1951)
37. Wigner, E.P.: Group Theory and its Application to the Quantum Mechanics of Atomic Spectra. Academic Press, New York (1959)
38. Zelobenko, D.P.: Compact Lie groups and their representations, Transl. Math. Monogr., vol. 40. Am. Math. Soc., Providence (1973)