The Journal of Fourier Analysis and Applications

Volume 9, Issue 4, 2003

FFTs for the 2-Sphere–Improvements and Variations

D.M. Healy, Jr., D.N. Rockmore, P.J. Kostelec, and S. Moore

Communicated by John J. Benedetto

ABSTRACT. Earlier work by Driscoll and Healy [18] has produced an efficient algorithm for computing the Fourier transform of band-limited functions on the 2-sphere. In this article we present a reformulation and variation of the original algorithm which results in a greatly improved inverse transform, and consequent improved convolution algorithm for such functions. All require at most $O(N \log^2 N)$ *operations where* N *is the number of sample points. We also address implementation considerations and give heuristics for allowing reliable and computationally efficient floating point implementations of slightly modified algorithms. These claims are supported by extensive numerical experiments from our implementation in C on DEC, HP, SGI and Linux Pentium platforms. These results indicate that variations of the algorithm are both reliable and efficient for a large range of useful problem sizes. Performance appears to be architecture-dependent. The article concludes with a brief discussion of a few potential applications.*

Math Subject Classifications. primary: 33C55, 65T99; secondary: 42C10, 65T40.
Keywords and Phrases. spherical Fourier transform; spherical harmonics; fast Legendre transform; recurrence relations.
Acknowledgements and Notes. Earlier versions of this article have been circulated through some of the computational harmonic analysis community. (Preliminary versions of some of the results have also appeared in the "An FFT for the 2-sphere and Applications," Proc. of ICASSP-96, Volume 3, p. 1323–1326.) Nevertheless, this is the first appearance of this work in print, and this final version differs from the most recent ("FFTs for the 2-Sphere—Improvements and Variations," D. Healy, S. Moore, and D. Rockmore, Department of Computer Science, Dartmouth College PCS-TR96-292, May, 1996), mainly in the extensive numerical experiments that have been performed using newly optimized code on current platforms. These experiments are "reproducible" in the sense of Donoho—the software is easily obtained over the web, and to date has been used by numerous researchers on their home platforms.
D.M. Healy, Jr. was supported in part by ARPA as administered by the AFOSR under contract DOD-F4960-93-0567. D.N. Rockmore was supported in part by NSF DMS Awards 9404275 and P.J. Kostelec was supported by a Presidential Faculty Fellowship.

ISSN 1069-5869

1. Introduction

1.1 History

The numerical calculation of Fourier expansions and convolutions of functions on the 2-sphere are related problems which have been identified as important computational issues in many areas of applied science. For example, potential applications are found in astronomy [52], computer vision [38, 59], medical imaging [12], biology [48], statistical analysis of directional data [29, 28] and chemistry [43]. Protein surfaces may be represented with spherical harmonics, which enables efficient means of analyzing protein-protein interactions [20]. The article [27] contains many references for possible applications in physics.

A significant set of applications comes from the fields of numerical weather prediction and global circulation modeling. In these areas much of the computational effort is directed towards the numerical solution of partial differential equations in spherical geometry [9, 60, 66]. Use of spectral methods for these purposes requires efficient and reliable algorithms for computing spherical harmonic expansions—the lack of such an algorithm (until now) has been a serious bottleneck in pursuing a spectral method approach (see e. g., [60], p. 3416 and [66]).

These sorts of applications have motivated much of the research in fast algorithms for computing spherical harmonic expansions. Early work proposed approximate solutions to the problem of computing Fourier expansions on the 2-sphere [54]. Other related work includes the development of generalizations of spherical harmonic transforms and efficient Legendre projection methods [65] for uses where the spectral coefficients are not required. Also, there is the use of fast multipole methods for the computation of Legendre polynomial expansions, which gives part of the full spherical harmonic expansion [3]. The complexity of these algorithms scales linearly in the desired accuracy. We refer the interested reader to [63], where may be found a detailed and rigorous comparison and analysis of the performances of a variety of Legendre projection algorithms, including those based on the fast multipole method, as well as other methods [49].

A different approach was proposed in [18], which presents an exact, asymptotically fast approach to the problem of computing spherical harmonic coefficients. More specifically, an algorithm is given which in exact arithmetic permits efficient exact computation of the Fourier expansion and convolution of functions on the two-sphere, assuming that the functions have finite expansions in terms of spherical harmonics. The effects of finite precision arithmetic in the implementation were studied through a priori error estimates and numerical experiments for crucial steps in the algorithm. These results strongly suggest the possibility of an effective floating point implementation of the algorithm.

In a general setting, these algorithms can all be viewed as computational approaches to nonabelian harmonic analysis. Cast in this light, they have as their natural ancestor the now "classical" Fast Fourier Transform (FFT), first discovered by Gauss and later rediscovered and popularized by Cooley and Tukey (see [34] for a nice outline of much of the history, and [46] for a group theoretic context of this work). This family of algorithms efficiently computes the Fourier coefficients of a band-limited function on the circle, an abelian group. Its effective implementation has made possible a wealth of advances in many fields, most noticeably digital signal processing (see e. g., [21, 53]). A natural direction of generalization of the Cooley–Tukey FFT is the development of efficient and reliable algorithms to compute expansions of functions defined on finite or compact groups in terms of irreducible matrix coefficients [45]. In this article we present some particular results towards the development

of this program, especially as relates to the 2-sphere. For work in the nonabelian noncompact setting (esp. the motion group) motivated by applications to engineering, see [13], and for a survey of other results and applications of "generalized FFTs," see [47, 58].

This article continues and supplements the work in [18]. We give a reformulation and variation of that article's algorithm in terms of a sparse structured factorization of the appropriate Fourier transform matrix. Reordering and transposing provides a more efficient inverse transform than that presented in [18]; we now attain the same order of complexity as that fast forward transform. Efficient inverse and forward transforms combine to yield a faster convolution algorithm. This is the most efficient such algorithm known to date. Our new presentation resembles the filterbank technology currently of interest in many digital signal processing applications (see e. g., [67]). The algorithms are of more than theoretical interest. By slightly varying the basic algorithm (at little theoretical cost) we have obtained numerically reliable and computationally efficient implementations that are competitive with other algorithms at useful problem sizes. The relative performance of the algorithms appears to be architecture-dependent.

1.2 Main Idea

The Fourier transform of a function on the 2-sphere amounts to its L^2-projection onto the elements of a basis of *spherical harmonic functions.* This particular basis respects the rotational symmetries of the 2-sphere in much the same way that the familiar sines and cosines are adapted to translations of periodic functions on the real line.

The primary algorithmic tool presented in this article is an efficient algorithm for the computation of *discrete Legendre transforms.* For a given "bandwidth" $B > 0$ (cf. Section 2) these are the sums of the form

$$\widehat{\mathbf{s}}(\ell, m) = \sum_{k=0}^{2B-1} P_\ell^m(\cos\theta_k)[\mathbf{s}]_k \qquad |m| \le \ell = 0, 1, \ldots, B-1 \tag{1.1}$$

where P_ℓ^m is the associated Legendre function of degree ℓ and order m, $\theta_k = \frac{\pi(k+1/2)}{2B}$ and $\mathbf{s}$ is a data vector with k^{th} component $[\mathbf{s}]_k$, obtained from the samples of the original function which we wish to transform. Simply stated, these are inner products of a vector of sampled associated Legendre functions $\mathbf{P}_\ell^m$ against a data vector $\mathbf{s}$.

An obvious approach to evaluating the sums (1.1) is to compute them successively for the various degrees and orders. Computed in this way, each of these steps requires $2B$ multiplications and $2B-1$ additions. Since there are $n = B^2$ of these steps required to compute the full Fourier transform, this implies a total naive complexity of at most $4n^{3/2} = 4B^3$ operations.[1] We will refer to this as the **direct algorithm**.

In contrast, the results of this article provide the basic tools for algorithms which improve the asymptotic complexity of the complete set of Legendre transforms from $O(n^{3/2})$ to $O(n \log^2 n)$ ($n = B^2$). Variants of our algorithm still improve upon the better $O(n^{3/2})$ exact algorithms, beginning at problem sizes as small as $B = 256$ (cf. Section 6).

These fast algorithms use a divide and conquer approach, which may be familiar from the structure of many of the usual (abelian) FFT algorithms (see e. g., [15, 68]). In such an

[1] Here we assume the standard arithmetic complexity model which defines a single operation as a complex multiplication followed by a complex addition.

approach, the problem of computing projections onto Legendre functions is decomposed into smaller subproblems of a similar form. The subproblems are solved recursively, by further subdivision, and their solutions are combined to solve the original problem. The advantage to this approach derives from the fact that the costs of the smaller subproblems, together with the cost of splitting will be less than the cost of the direct approach.

To insure that the splitting actually results in subproblems of reduced complexity we apply two main ideas:

- **Using the Recurrence.** The associated Legendre functions satisfy a *three-term recurrence* (as do many systems of special functions). This drives the divide and conquer strategy, expressing the higher degree inner products in terms of inner products with sampled trigonometric polynomials of lower degree, which according to the following observation, can be computed more efficiently.
- **Smoothing and Subsampling.** The inner products of a data vector against sampled low degree trigonometric polynomials may actually be accomplished in fewer than B operations by using a "smoothed" data vector with fewer samples. In fact, only ℓ samples are needed to compute the inner product with a trigonometric polynomial of degree $\ell < B$.

Issues of numerical reliability are important in implementation and we will see how these considerations may be satisfied by developing variations of the basic algorithm. The main problem to guard against is that of pushing the recurrence too far.

1.3 Organization

The organization of the remainder of the article is as follows. In Section 2 we briefly recall the notation and some necessary technical background material on Fourier analysis and fast algorithms. In particular, we review Fourier analysis on the 2-sphere i. e., the theory of spherical harmonics. We continue with a discussion of the recurrences satisfied by these functions and of the technique of smoothing and subsampling mentioned above. Section 3 contains the main algorithmic results of the article. Here we explain how the techniques mentioned above (using the recurrence, smoothing and subsampling) may be combined to produce our basic divide and conquer transform algorithm. This algorithm has a natural formulation as a particular structured matrix factorization of a matrix containing sampled Legendre functions. This simultaneously provides a similar factorization of the transpose of the matrix for the forward transform, and as a result, we immediately obtain a fast inverse transform or synthesis algorithm and consequent fast convolution algorithm as well. This is all explained in Section 4. In Section 5 we discuss some simple variants of the basic algorithm, designed to give speed-ups in actual implementations. Numerical results supporting our claims of efficiency and reliability are presented in Section 6. In Section 7, we outline two possible applications of the algorithm. For the first application we show how our algorithm may be used towards the efficient computation of the bispectrum for band-limited functions. In the other application we present some experiments in using our algorithm for matched filtering of signals on the 2-sphere. We conclude in Section 8 with a summary and brief discussion of future work.

2. Background

This section collects the basic tools for the formulation and solution of the problem of efficient Fourier analysis on the 2-sphere. We first recall the definitions of Fourier analysis on the 2-sphere and the spherical harmonic functions. We present the discretization of the Fourier transform and indicate the complexity of standard algorithms for its evaluation. We show how the problem of a fast transform is reduced to the question of fast discrete Legendre transforms, and detail the main tools (the use of recurrence relations as well as smoothing and subsampling) used for their efficient evaluation.

2.1 Fourier Analysis on the 2-Sphere

As usual, S^2 denotes the 2-sphere or unit sphere in $\mathbf{R}^3$. A unit vector in $\mathbf{R}^3$ may be described by an angle θ, $0 \le \theta \le \pi$ measured down from the z-axis and an angle ϕ, $0 \le \phi < 2\pi$ measured counterclockwise off the x-axis; this representation is unique for almost all unit vectors. Thus, if $\omega \in S^2$, then we may write $\omega(\theta, \phi) = (\cos\phi \sin\theta, \sin\phi \sin\theta, \cos\theta)$.

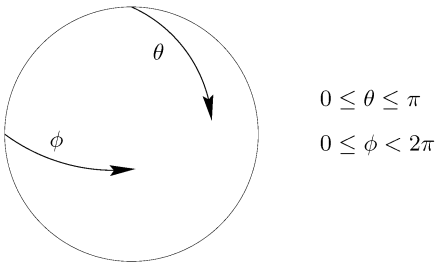


FIGURE 1 Parametrization of the 2-sphere.

Let $L^2(S^2)$ denote the Hilbert space of square integrable functions on the S^2. In coordinates, the usual inner product is given by

$$\langle f, h\rangle = \int_0^{\pi} \left[\int_0^{2\pi} f(\theta, \phi)\overline{h(\theta, \phi)}\, d\phi \right] \sin\theta \, d\theta \, . \tag{2.1}$$

As is well-known (see e. g., [70]), the **spherical harmonics** provide an orthonormal basis for $L^2(S^2)$. For any nonnegative integer ℓ and integer m with $|m| \le \ell$, the (ℓ, m)-**spherical harmonic** Y_ℓ^m is a harmonic homogeneous polynomial of degree ℓ. The harmonics of degree ℓ span a subspace of $L^2(S^2)$ of dimension $2\ell + 1$ which is invariant under the rotations of the sphere. The expansion of any function $f \in L^2(S^2)$ in terms of spherical harmonics is written

$$f = \sum_{\ell \ge 0} \sum_{|m| \le \ell} \widehat{f}(\ell, m) Y_\ell^m \tag{2.2}$$

and $\hat{f}(\ell, m)$ denotes the (ℓ, m)-**Fourier coefficient**, equal to $\langle f, Y_\ell^m \rangle$.

In the coordinates (θ, ϕ), Y_ℓ^m has a factorization,

$$Y_\ell^m(\theta, \phi) = k_{\ell,m} P_\ell^m(\cos\theta) e^{im\phi} \tag{2.3}$$

where P_ℓ^m is the **associated Legendre function** of degree ℓ and order m and $k_{\ell,m}$ is a normalization constant.

Consequently, separating variables according to (2.3) shows that the computation of the spherical harmonic transform can be reduced to a regular Fourier transform in the longitudinal coordinate ϕ followed by a projection onto the associated Legendre functions

$$\widehat{f}(\ell, m) = \left\langle f, Y_\ell^m \right\rangle = k_{\ell,m} \int_0^\pi \left[\int_0^{2\pi} e^{-im\phi} f(\theta, \phi)\, d\phi \right] P_\ell^m(\cos\theta) \sin\theta\, d\theta\,. \quad (2.4)$$

The associated Legendre functions satisfy a characteristic **three-term recurrence**

$$(\ell - m + 1) P_{\ell+1}^m(x) - (2\ell + 1) x P_\ell^m(x) + (\ell + m) P_{\ell-1}^m(x) = 0 \quad (2.5)$$

critical for the algorithms developed in this article.

In analogy with the case of functions on the circle, we say that $f \in L^2(S^2)$ is **band-limited** with **band-limit** or **bandwidth** $B \geq 0$ if $\widehat{f}(\ell, m) = 0$ for all $\ell \geq B$. For band-limited functions we have a simple quadrature (sampling) result which reduces the integrals (2.4) to finite weighted sums of a sampled data vector obtained from the integrand.

Theorem 1 (cf. [18], Theorem 3).
Let $f \in L^2(S^2)$ have bandwidth B. Then for each $|m| \leq \ell < B$,

$$\widehat{f}(\ell, m) = \frac{\sqrt{2\pi}}{2B} \sum_{j=0}^{2B-1} \sum_{k=0}^{2B-1} a_j^{(B)} f(\theta_j, \phi_k) e^{-im\phi_k} P_\ell^m(\cos\theta_j)$$

where the sample points are chosen from the equiangular grid: $\theta_j = \pi(2j+1)/4B$, $\phi_k = 2\pi k/2B$; and the weights $a_j^{(B)}$ (cf. Figure 2) play a role analogous to the $\sin\theta$ *factor in the integrals.*

Remark. Although we will use this formulation of the Sampling Theorem to provide the starting point for our fast algorithms, it is actually possible to give a sampling theorem which uses only B samples in the θ coordinate. These samples are then interpolated to give $2B - 1$ samples for use in the fast transform algorithms.

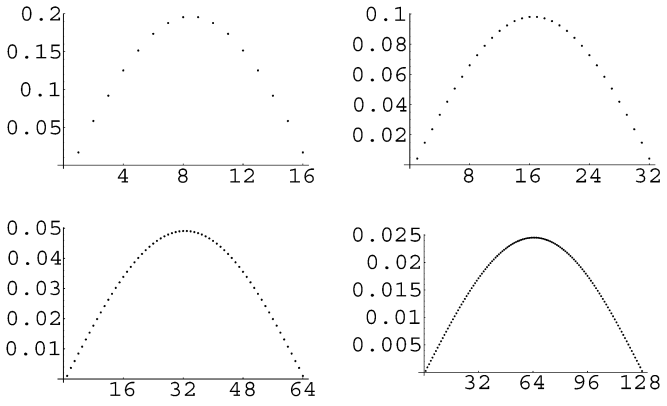


FIGURE 2 Plot of sample weights for range of problem sizes.

2.2 A Discrete Fourier Transform for S^2 and the Legendre Transform

The **Fourier transform** of a function f of bandwidth B is the collection of its Fourier coefficients,

$$\left\{ \hat{f}(\ell, m) \mid 0 \leq |m| \leq \ell < B \right\} .$$

The Sampling Theorem (Theorem 1) expresses the (ℓ, m)-Fourier coefficient of f as the finite sum

$$\widehat{f}(\ell, m) = \frac{\sqrt{2\pi}}{2B} \sum_{j=0}^{2B-1} \sum_{k=0}^{2B-1} a_j^{(B)} f(\theta_j, \phi_k) e^{-im\phi_k} P_\ell^m(\cos\theta_j) . \tag{2.6}$$

This method of computing the Fourier coefficients of f is called the **discrete Fourier transform,** or **DFT** of f. Notice that direct computation of each $\hat{f}(\ell, m)$ would require $4B^2$ operations and thus $O(B^4)$ in total.

More efficient algorithms use a separation of variables approach. We proceed by first summing over the k index, computing the inner exponential sums which depend only the indices j and m. We may do this efficiently for all m between $-B$ and B via the FFT (cf. [21]). The computation is completed by performing the requisite **discrete Legendre transforms**, which for a given order $m > 0$ we define as a set of sums

$$\sum_{k=0}^{N-1} [\mathbf{s}]_k P_\ell^m(\cos\theta_k) = \left\langle \mathbf{s}, \mathbf{P}_\ell^m \right\rangle ; \qquad \ell = m, m+1, \ldots, N-1 , \tag{2.7}$$

for an arbitrary input vector $\mathbf{s}$ with k^{th} component $[\mathbf{s}]_k$. Here we have introduced a discrete inner product notation and the convention that $\mathbf{P}_\ell^m$ denotes the vector comprised of the appropriate samples of the function $P_\ell^m(\cos\theta)$:

$$\mathbf{P}_\ell^m = \begin{pmatrix} P_\ell^m(\cos\theta_0) \\ \vdots \\ P_\ell^m(\cos\theta_{N-1}) \end{pmatrix} .$$

We may also say that (2.7) **computes the projection of s onto $\mathbf{P}_\ell^m$**.

The problem of a fast spherical harmonic transform is now reduced to the efficient calculation of these discrete Legendre transforms. Notice that even without a fast algorithm here, the separation of variables turns an $O(B^4)$ calculation into an $O(B^3)$ calculation.

In order to simplify the discussion of the basic idea for the efficient evaluation of the sums (2.7), we will specialize to the case $m = 0$, for which $P_\ell^m = P_\ell^0 = P_\ell$ is the **Legendre polynomial** of degree ℓ. For higher orders the algorithm generalizes directly.

As remarked in the introduction, our fast algorithm relies on two basic ideas,

(1) Using the Recurrence; (2) Smoothing and Subsampling.

We proceed by developing each of these techniques separately (Sections 2.3 and 2.4) and then show how they are combined to yield our algorithm (Section 3).

2.3 Using the Legendre Recurrence

The recurrence relation satisfied by $P_\ell(\cos\theta)$ is (for $\ell \geq 0$)

$$(\ell+1)\,P_{\ell+1}(\cos\theta) - (2\ell+1)\,\cos\theta\,P_\ell(\cos\theta) + (\ell)\,P_{\ell-1}(\cos\theta) = 0 \tag{2.8}$$

with initial conditions $P_0(\cos\theta) = 1$ and $P_{-1}(\cos\theta) = 0$. Consequently, the higher degree Legendre polynomials can be expressed in terms of those of lower degree as follows. For any fixed "level" L, iterating the recurrence formula (2.8) forward r steps produces trigonometric polynomials A_r^L and B_r^L such that

$$P_{L+r}(\cos\theta) = A_r^L(\cos\theta)\;P_L(\cos\theta) + B_r^L(\cos\theta)\;P_{L-1}(\cos\theta)\;, \tag{2.9}$$

for $r \geq 1$. We refer to A_r^L and B_r^L, as **shifted Legendre polynomials**, as they are generated by the following shifted form of the Legendre polynomial recurrence (2.8),

$$(L+r+1)\,p_{r+1}^L(\cos\theta) - (2L+2r+1)\,\cos\theta\,p_r^L(\cos\theta) + (L+r)\,p_{r-1}^L(\cos\theta) = 0 \tag{2.10}$$

with initial conditions $A_0^L = 1$, $A_{-1}^L = 0$ and $B_0^L = 0$, $B_{-1}^L = 1$, respectively. This is readily concluded by comparing a matrix formulation of (2.9),

$$\begin{pmatrix} P_{L+r+1} \\ P_{L+r} \end{pmatrix} = \begin{pmatrix} A_{r+1}^L & B_{r+1}^L \\ A_r^L & B_r^L \end{pmatrix} \begin{pmatrix} P_L \\ P_{L-1} \end{pmatrix}\;, \tag{2.11}$$

with the matrix form of the one-step recurrence (2.8)

$$\begin{aligned} \begin{pmatrix} P_{L+r+1} \\ P_{L+r} \end{pmatrix} &= \begin{pmatrix} \frac{2(L+r)+1}{L+r+1}\cos\theta & -\frac{L+r}{L+r+1} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} P_{L+r} \\ P_{L+r-1} \end{pmatrix} \\ &= \begin{pmatrix} \frac{2L+2r+1}{L+r+1}\cos\theta & -\frac{L+r}{L+r+1} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} A_r^L & B_r^L \\ A_{r-1}^L & B_{r-1}^L \end{pmatrix} \begin{pmatrix} P_L \\ P_{L-1} \end{pmatrix}\;. \end{aligned}$$

A similar argument gives the general recurrence satisfied by the shifted Legendre functions

$$\begin{pmatrix} A_{r+s}^L & B_{r+s}^L \\ A_{r+s-1}^L & B_{r+s-1}^L \end{pmatrix} = \begin{pmatrix} A_s^{L+r} & B_s^{L+r} \\ A_{s-1}^{L+r} & B_{s-1}^{L+r} \end{pmatrix} \cdot \begin{pmatrix} A_r^L & B_r^L \\ A_{r-1}^L & B_{r-1}^L \end{pmatrix}\;. \tag{2.12}$$

We make the convention that

$$A_r^0 = P_r, \quad \text{and} \quad B_r^0 = 0 \tag{2.13}$$

so that (2.12) subsumes the original Legendre recurrence (2.11).

From (2.10) we see that the degrees of the shifted Legendre polynomials A_r^L and B_r^L are r and $r-1$, respectively. The point of introducing these polynomials is that they allow us to rewrite projections onto high degree Legendre polynomials as sums of projections onto (shifted) Legendre polynomials of lower degree. More precisely, suppose that in the course of computing the inner products $\langle \mathbf{s}, \mathbf{P}_j \rangle$ (for $j \leq L$) we had stored the components of the vectors $\mathbf{s}^j = \mathbf{s}\mathbf{P}_j$, defined as the pointwise product of the vectors $\mathbf{s}$ and $\mathbf{P}_j$. The

recurrence (2.9) allows us to re-use this data to compute the projection of **s** onto a higher order function, P_{L+r} as follows:

$$\langle \mathbf{s}, \mathbf{P}_{L+r} \rangle = \left\langle \mathbf{s}, \left(\mathbf{A}_r^L \ \mathbf{P}_L + \mathbf{B}_r^L \ \mathbf{P}_{L-1} \right) \right\rangle = \left\langle \mathbf{s}^L, \mathbf{A}_r^L \right\rangle + \left\langle \mathbf{s}^{L-1}, \mathbf{B}_r^L \right\rangle \ . \tag{2.14}$$

If the vectors $\mathbf{s}^L = \mathbf{s}\mathbf{P}_L$ and $\mathbf{s}^{L-1} = \mathbf{s}\mathbf{P}_{L-1}$ have been stored, then (2.14) shows that the higher degree inner product can be computed as inner products of stored data and (precomputed) sampled values of the polynomials A_r^L and B_r^L, each of which have degree at most r, which is necessarily less than the degree of P_{L+r}.

Since A_r^L and B_r^L also satisfy a recurrence, this procedure can be repeated. Following this through yields a divide and conquer scheme for performing the full computation.

The motivation for the successive reductions is that the projections of a data vector onto lower degree trigonometric polynomials can be computed more efficiently by first lowpass filtering (smoothing) and then subsampling the data vector, so that ultimately the projection may be computed by a summation with fewer terms (cf. Lemma 1). This is the next topic.

2.4 Smoothing and Subsampling–Working in the "Cosine Transform" Domain

The "conquer" part of the divide and conquer algorithm described above is the computation of inner products of the form $\langle \mathbf{s}, \mathbf{Q}_n \rangle$ for a sequence of vectors $\mathbf{Q}_n$ varying over some range of n. Here **s** is a data vector, and $\mathbf{Q}_n$ is composed of samples of a trigonometric polynomial Q_n of degree n (in fact, a shifted Legendre polynomial). We examine the complexity of computing these inner products, and show that it grows linearly with n, modulo a fixed overhead. We will make use of a cosine transform representation of the vectors in the inner product. Put in the language of matrices, we will show that the matrix of the discrete Legendre transform may be brought to triangular form by means of the cosine transform matrix.

We begin by recalling that any vector of length N, **s**, may always be represented as uniform samples of a cosine series of degree less than N:

$$[\mathbf{s}]_k = \sum_{n=0}^{N-1} \sigma_n \cos(n\theta_k) \ , \qquad k = 0, \ldots, N-1 \ , \tag{2.15}$$

where, as before, $\theta_k = \frac{\pi(2k+1)}{2N}$.

The coefficients σ_n are obtained by computing the **discrete cosine transform** (DCT) of **s**. Explicitly, we let $\mathcal{C}_N$ denote the N-dimensional orthogonal DCT matrix (see e. g., [21], p. 386) comprised of normalized sampled cosines:

$$(\mathcal{C}_N)_{j,k} = b(j)\cos(j\theta_k) \qquad 0 \le j, k \le N-1 \ , \tag{2.16}$$

with normalization factors

$$b(0) = \sqrt{\frac{1}{N}} \quad \text{and} \quad b(j) = \sqrt{\frac{2}{N}} \text{ for } j = 1, \ldots, N \ . \tag{2.17}$$

In terms of $\mathcal{C} = \mathcal{C}_N$, the coefficients in (2.15) are given by

$$\sigma_n = \begin{cases} \sqrt{\frac{2}{N}}[\mathcal{C}\mathbf{s}]_n & \text{if } n \neq 0 \\ \sqrt{\frac{1}{N}}[\mathcal{C}\mathbf{s}]_n & \text{if } n = 0\,. \end{cases} \tag{2.18}$$

Remark. In practice, the set of cosine coefficients $\{[\mathcal{C}\mathbf{s}]_n \mid n = 0, \ldots, N-1\}$ can be obtained efficiently (in at most $\frac{3}{2}N \log N$ operations for N a power of 2) by a fast DCT algorithm, which amounts to a clever factorization of the matrix $\mathcal{C}$ (see [64] and the references contained therein).

The orthogonality of $\mathcal{C}$ implies $\langle \mathcal{C}\mathbf{s}, \mathcal{C}\mathbf{Q}\rangle = \langle \mathbf{s}, \mathbf{Q}\rangle$. The computational advantage of computing the inner product in the cosine transform domain comes from the fact that for any trigonometric polynomial Q of degree at most $N-1$, the cosine coefficients $[\mathcal{C}\mathbf{Q}]_n$ vanish for $n > \deg(Q)$ (cf. Lemma 1) and this reduces the number of operations required to compute the inner product with $\mathcal{C}\mathbf{s}$. In particular, this applies to the various Legendre functions we use.

To illustrate, the Legendre polynomial $Q = P_\ell$ is a trigonometric polynomial of degree ℓ. This is easily verified using the recurrence relation (2.8) with the initial conditions $P_0(\cos\theta) = 1$, $P_1(\cos\theta) = \cos\theta$. For example,

$$P_2(\cos\theta) = \frac{5}{3}\cos\theta\ \cos\theta + \frac{2}{3}1 = \frac{5}{3}\ \frac{\cos(2\theta)+1}{2} + \frac{2}{3}\,.$$

Consequently, for $n > \ell$, $[\mathcal{C}\mathbf{P}_\ell]_n = 0$ and the inner product sum $\langle \mathcal{C}\mathbf{P}_\ell, \mathcal{C}\mathbf{s}\rangle = \langle \mathbf{P}_\ell, \mathbf{s}\rangle$ can be computed as a sum of only $\ell + 1$ terms (instead of N),

$$\begin{aligned} \langle \mathbf{s}, \mathbf{P}_\ell\rangle &= \sum_{m=0}^{N-1} [\mathcal{C}\mathbf{s}]_m\, [\mathcal{C}\mathbf{P}_\ell]_m \\ &= \sum_{m=0}^{l} [\mathcal{C}\mathbf{s}]_m\, [\mathcal{C}\mathbf{P}_\ell]_m + \sum_{m=l+1}^{N-1} [\mathcal{C}\mathbf{s}]_m \cdot 0 \\ &= \sum_{m=0}^{l} [\mathcal{C}\mathbf{s}]_m\, [\mathcal{C}\mathbf{P}_\ell]_m\,. \end{aligned} \tag{2.19}$$

This shows that the computation of the low degree Legendre projections (i. e., projections of data onto P_ℓ for small ℓ) can be accomplished with very few operations, after the overhead of computing the cosine expansions of the data vector $\mathbf{s}$ and the vector of sampled Legendre polynomials $\mathbf{P}_\ell$.

In terms of matrix arithmetic (2.19) is equivalent to the observation that the matrix representing the discrete Legendre transform has a factorization as a product of a cosine transform matrix and a triangular matrix comprised of the Legendre polynomial cosine coefficients,

$$\left(\left(P_\ell(\cos\theta_j)\right)\right) = \begin{pmatrix} [\mathcal{C}\mathbf{P}_0]_0 & 0 & \cdots & 0 \\ [\mathcal{C}\mathbf{P}_1]_0 & [\mathcal{C}\mathbf{P}_1]_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \left[\mathcal{C}\mathbf{P}_{N-1}\right]_0 & \left[\mathcal{C}\mathbf{P}_{N-1}\right]_1 & \cdots & \left[\mathcal{C}\mathbf{P}_{N-1}\right]_{N-1} \end{pmatrix} \cdot \mathcal{C} \tag{2.20}$$

where $\mathcal{C}$ denotes the $N \times N$ orthogonal discrete cosine transform matrix defined in (2.16). If the cosine coefficients of the sampled Legendre polynomials ($[\mathcal{C}\mathbf{P}_j]_k$) are prestored, this approach is an alternative to the direct computation of the Legendre transform which has the same asymptotic complexity, but is faster for even moderate sized transforms, assuming the use of a fast DCT routine. (See the previous remark and reference on this issue). Figures 3 and 4 illustrate both of these approaches.

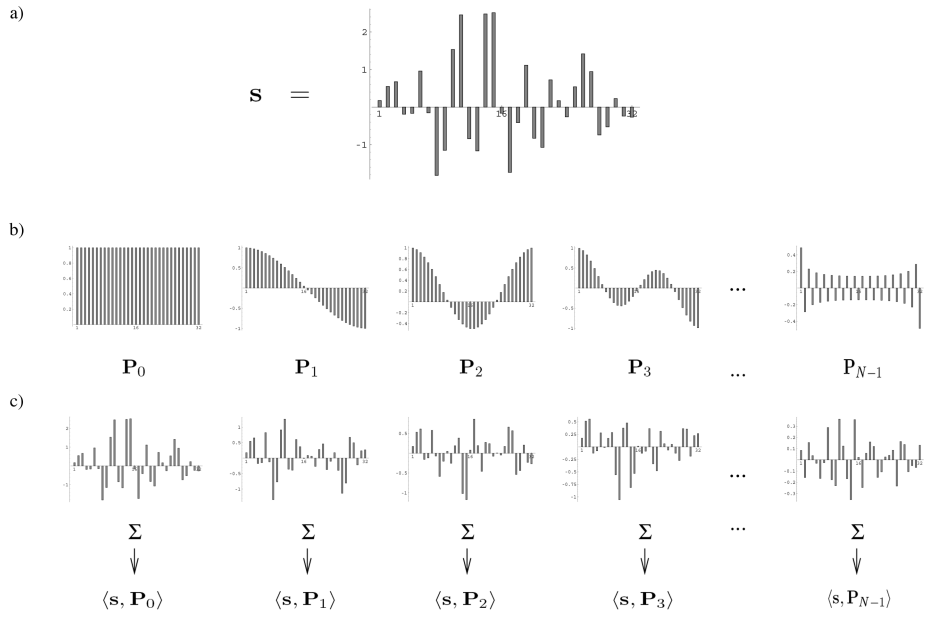


FIGURE 3 Graphical representation of a direct non-DCT discrete Legendre transform of sampled signal **s**: (a) Input of length N, **s**; (b) The vectors of sampled Legendre polynomials $\mathbf{P}_k$; (c) Pointwise multiplications $\mathbf{s}\ \mathbf{P}_k$ such that $\sum_j \left[\mathbf{s}\ \mathbf{P}_k\right]_j = \langle \mathbf{s}, \mathbf{P}_k \rangle$ in at most N operations indicated by the Σ notation.

Lemma 1 below gives a generalization of (2.19) appropriate for use in our fast algorithm, where we apply it to reduce the complexity of projections onto the lower degree Legendre functions. The lemma reformulates the earlier discussion of this section in terms of smoothing or filtering operators applied to the vectors involved in the inner products.

We define the **critically sampled lowpass operator** (of bandwidth p), denoted $\mathcal{L}_p^N$ (for $p < N$), by

$$\mathcal{L}_p^N = \mathcal{C}_p^{-1} \mathcal{T}_p^N \mathcal{C}_N \tag{2.21}$$

where $\mathcal{T}_p^N$ is the **truncation operator** that only keeps the first p coordinates of a given input vector. The effect of $\mathcal{L}_p^N$ is to first compute the cosine representation of a vector of length N, then remove all frequency components beyond p from **s**, (smoothing) and finally, to keep only those samples necessary to represent this smoothed version (subsampling). This is illustrated schematically in Figure 5 below. As indicated by the subscript and superscript, this operator takes sequences of length N to sequences of length p.

With the preceding notation, the necessary generalization of (2.19) is given by Lemma 1.

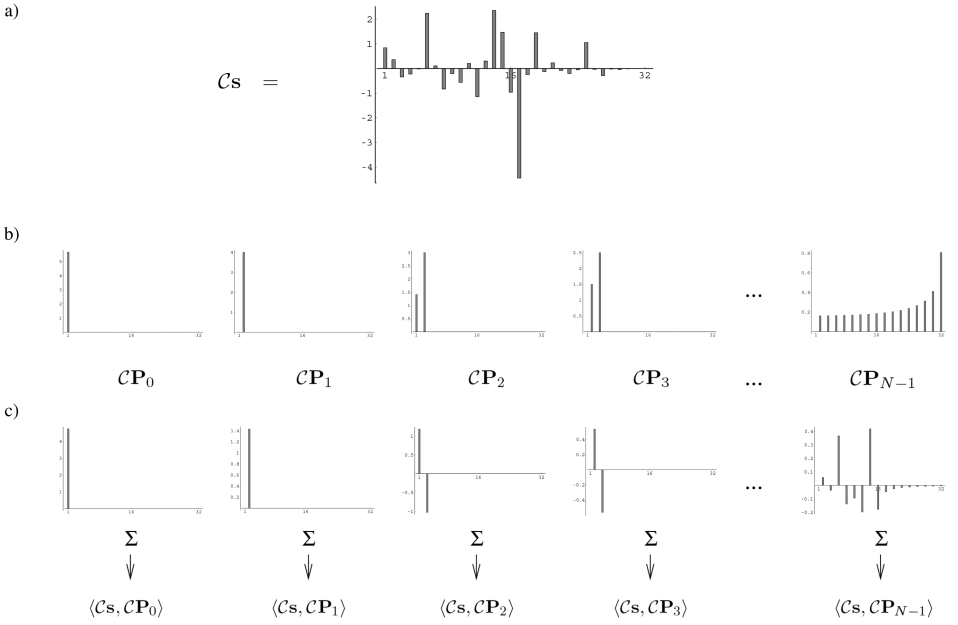


FIGURE 4 Graphical representation of discrete Legendre transform of Figure 3, using the DCTs of the signal **s** and the sampled Legendre polynomials: (a) Input $\mathcal{C}\mathbf{s}$ for **s** in Figure 3; (b) DCT of the vectors of sampled Legendre polynomials $\mathbf{P}_k$, $\mathcal{C}\mathbf{P}_k$. Notice that the DCT of a given Legendre polynomial has only as many nonzero coefficients as the degree; (c) Pointwise multiplications $\mathcal{C}\mathbf{s}\ \mathcal{C}\mathbf{P}_k$ such that $\sum_j \left[\mathcal{C}\mathbf{s}\ \mathcal{C}\mathbf{P}_k\right]_j = \langle \mathbf{s}, \mathbf{P}_k \rangle$ in at most k operations indicated by the Σ notation.

Lemma 1.

Let Q be a trigonometric polynomial of degree p,

$$Q(\cos\theta) = \sum_{m=0}^{p} \gamma_m \cos m\theta \ ,$$

and let **s** *be any sequence of length N with $N \geq p$. Then*

$$\langle \mathbf{s}, \mathbf{Q} \rangle = \sum_{k=0}^{N-1} [\mathbf{s}]_k \, Q(\cos\theta_k) = \sum_{j=0}^{p-1} \left[\mathcal{L}_p^N \mathbf{s}\right]_j \, Q\left(\cos\frac{N\theta_j}{p}\right) = \left\langle \mathcal{L}_p^N \mathbf{s}, \, \mathcal{L}_p^N \mathbf{Q} \right\rangle \ .$$

Proof.

$$\begin{aligned} \langle \mathbf{s}, \mathbf{Q} \rangle &= \langle \mathcal{C}_N \mathbf{s}, \mathcal{C}_N \mathbf{Q} \rangle \\ &= \sum_{k=0}^{N-1} [\mathcal{C}_N \mathbf{s}]_k [\mathcal{C}_N \mathbf{Q}]_k \\ &= \sum_{k=0}^{p} [\mathcal{C}_N \mathbf{s}]_k [\mathcal{C}_N \mathbf{Q}]_k \end{aligned}$$

since $[\mathcal{C}_N \mathbf{Q}]_k = 0$ for $k > p$. This last sum is the same as

$$\left\langle \mathcal{T}_p^N \mathcal{C}_N \mathbf{s}, \mathcal{T}_p^N \mathcal{C}_N \mathbf{Q} \right\rangle = \left\langle \mathcal{L}_p^N \mathbf{s}, \mathcal{L}_p^N \mathbf{Q} \right\rangle \tag{2.22}$$

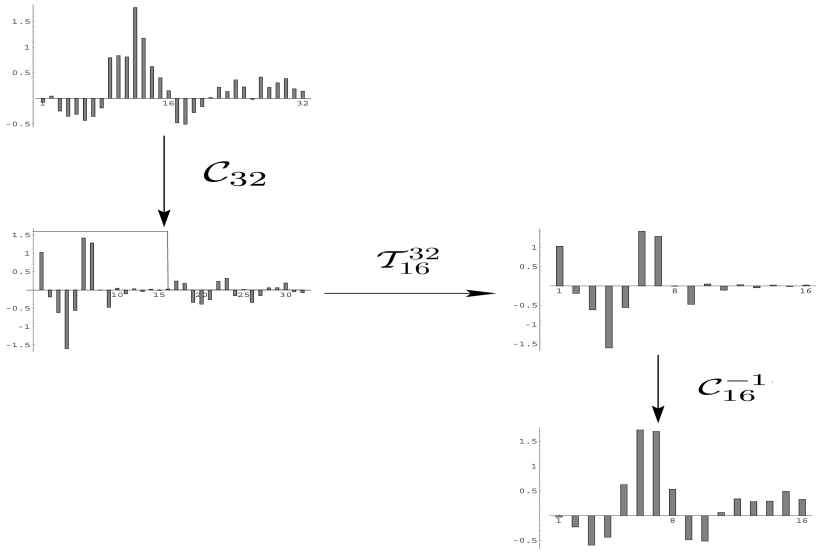


FIGURE 5 Smoothing and subsampling.

and it is immediately verified that $\mathcal{L}_p^N \mathbf{Q}$ is the same as the function Q sampled on the coarser grid. □

Note that the rightmost inner product is of sequences of length p, and since Q already has bandwidth p, the effect of applying $\mathcal{L}_p^N$ to Q is to simply sample Q on the coarser grid.

Lemma 2.

Let $0 < p \leq N$ be powers of 2. For an arbitrary input $\mathbf{s}$ *the computation* $\mathbf{s} \mapsto \mathcal{L}_p^N \mathbf{s}$ *can be accomplished in at most* $\frac{3}{2} N \log N + \frac{3}{2} p \log p$ *operations.*

Proof. As described above, $\mathcal{L}_p^N \mathbf{s}$ can be accomplished by taking the DCT of $\mathbf{s}$, a vector of length N, truncating the result to the first p coefficients, and then using the inverse DCT to evaluate the lowpassed function at p samples. A fast DCT (and inverse DCT) algorithm due to Stiedl and Tasche [64] requires $\frac{3}{2} m \log m$ operations to compute a DCT of length m. Since truncation requires no additional operations, the lemma follows. □

Lemma 3.

Let N be a power of 2 and $\mathbf{s}$ *a vector of length N. Suppose $F_\ell(\cos\theta)$ ($\ell = 0, \ldots, N-1$) satisfies a recurrence $a_\ell F_{\ell+1}(\cos\theta) - b_\ell \cos(\theta) F_\ell(\cos\theta) + c_\ell F_{\ell-1}(\cos\theta)$ with initial conditions $F_{-1} = 0$ and $F_0 = 1$. Then assuming the prestorage of the DCT of the F_ℓ, for an arbitrary input* $\mathbf{s}$*, the collection of inner products $\langle \mathbf{F}_\ell, \mathbf{s} \rangle$ can be computed in at most $\frac{3}{2} N \log N + \frac{N(N+1)}{2}$ operations, versus N^2 required by direct computation.*

Computation of any collection of discrete trigonometric polynomial projections by applying the DCT according to Lemma 1 will be referred to as computation by the **seminaive** approach.

The key property of (2.19) is that it shows that only $\ell + 1$ samples of a suitably modified data vector are needed to compute the inner products $\langle \mathbf{s}, \mathbf{P}_\ell \rangle$. The generalization of this to the various associated Legendre functions and their shifted forms permits us to

reduce the complexity of the small Legendre transform problems obtained by applying a divide and conquer approach to the original problem. As we shall see in Section 3.1, these subproblems compute inner products of data with (shifted) Legendre polynomials of at most half the degree of the inner products in the original problem.

3. Fast Discrete Legendre Transforms

In Section 2 we saw that the problem of computing fast spherical harmonic expansions for band-limited functions may be reduced to the efficient calculation of discrete Legendre transforms defined in (2.7).

In this section we show how this is done. We simplify the discussion by presenting the case of the Legendre polynomial transform, $m = 0$, For higher orders the algorithm generalizes directly.

The problem of interest is then to efficiently compute

$$\sum_{k=0}^{N-1} [\mathbf{s}]_k P_\ell(\cos\theta_k) = \langle \mathbf{s}, \mathbf{P}_\ell \rangle; \qquad \ell = 0, 1, \ldots, N-1\,, \tag{3.1}$$

where $\mathbf{s}$ represents a data vector which we think of as sampled values of a function f which we wish to transform. To do this we apply the two basic notions, the Legendre recurrence, and smoothing, which we have considered in Sections 2.3 and 2.4. In this section we show how they are combined in a divide and conquer algorithm for efficient computation of the Legendre transform. This is our main theoretical result and it can be found in Section 3.3 as Theorem 2.

3.1 Fast Discrete Legendre Transform Via Divide and Conquer

Sections 2.3 and 2.4 provide the ingredients that make up our divide and conquer algorithm for computing discrete Legendre transforms. Schematically, the idea is quite simple to describe: The recurrence relations satisfied by the (shifted) Legendre polynomials (2.9) permit projection of data onto a collection of Legendre functions to be computed as linear combinations of similar projections onto Legendre functions of lower degree. The projections at lower degree require fewer samples (cf. Lemma 1), thereby taking on the form of problems of smaller size.

A uniform formulation of the problem allows us to divide the original problem (3.1) into a low degree and high degree subproblem, each of which can be shown to be equivalent to Legendre transforms of half the size. Let $T(N)$ denotes the complexity of a discrete Legendre transform of size N. Then this simple description gives rise to the usual divide and conquer recurrence $T(N) = 2T(N/2) + S(N)$ (see e. g., [15]), where $S(N)$ represents the overhead cost of rewriting the problem of size N as two subproblems of size $N/2$. Efficiency derives from the ability to perform the reduction quickly. In particular we shall see that $S(N)$ is $O(N \log N)$, so that iterating this subdivision procedure yields an $O(N \log^2 N)$ algorithm for computing the original discrete Legendre transform.

Before giving the general framework, we illustrate the details of the basic idea by applying it once to split the original problem of computing the N Legendre projections $\langle \mathbf{s}, \mathbf{P}_\ell \rangle$, $(0 \le \ell < N)$ into two subproblems of size $N/2$. To do this, we proceed by dividing the original problem into two separate computations:

- **Low degree transform coefficients**: $\langle \mathbf{s}, \mathbf{P}_r \rangle$, $0 \le r < \frac{N}{2}$.
- **High degree transform coefficients**: $\langle \mathbf{s}, \mathbf{P}_{\frac{N}{2}+r} \rangle$, $0 \le r < \frac{N}{2}$.

Although each subproblem is a collection of $N/2$ projections, these are not as yet problems of size $N/2$ since both use input of size N. (Recall that the original problem assumes input of size N). Reduction of the problem size will come from applying Lemma 1 which showed that any trigonometric polynomial projection $\langle \mathbf{s}, \mathbf{Q} \rangle$ can be computed as a sum of length $\deg(Q)$ by smoothing. This fact can be used immediately for the low degree projections, yielding the equivalent set of inner products $\langle \mathcal{L}^N_{\frac{N}{2}} \mathbf{s}, \mathcal{L}^N_{\frac{N}{2}} \mathbf{P}_\ell \rangle$ $(0 \le \ell < \frac{N}{2})$, which is a discrete Legendre transform of size $N/2$. That is, we are projecting onto $N/2$ Legendre functions using inner products of length $N/2$.

To apply the same idea to the high degree projections, we use the recurrence (2.9) which allows $\langle \mathbf{s}, \mathbf{P}_{\frac{N}{2}+r} \rangle$, $0 \le r < \frac{N}{2}$ to be rewritten as a sum of projections onto lower degree shifted Legendre polynomials,

$$\begin{aligned} \left\langle \mathbf{s}, P_{\frac{N}{2}+r} \right\rangle &= \left\langle \mathbf{s}, \mathbf{A}_r^{\frac{N}{2}} \mathbf{P}_{\frac{N}{2}} + \mathbf{B}_r^{\frac{N}{2}} \mathbf{P}_{\frac{N}{2}-1} \right\rangle \\ &= \left\langle \mathbf{s}^{\frac{N}{2}}, \mathbf{A}_r^{\frac{N}{2}} \right\rangle + \left\langle \mathbf{s}^{\frac{N}{2}-1}, \mathbf{B}_r^{\frac{N}{2}} \right\rangle \end{aligned} \tag{3.2}$$

where we have maintained the notation from Section 2.3, writing $\mathbf{s}^{\frac{N}{2}} = \mathbf{s}\mathbf{P}_{\frac{N}{2}}$ and $\mathbf{s}^{\frac{N}{2}-1} = \mathbf{s}\mathbf{P}_{\frac{N}{2}-1}$. Since for $r = 0, \ldots, \frac{N}{2} - 1$ the shifted Legendre polynomials $A_r^{\frac{N}{2}}$ and $B_r^{\frac{N}{2}}$ all have degree less than $N/2$, again Lemma 1 may be applied and (3.2) may be computed as a sum of inner products of length $N/2$,

$$\left\langle \mathcal{L}^N_{\frac{N}{2}} \mathbf{s}^{\frac{N}{2}}, \mathcal{L}^N_{\frac{N}{2}} \mathbf{A}_r^{\frac{N}{2}} \right\rangle + \left\langle \mathcal{L}^N_{\frac{N}{2}} \mathbf{s}^{\frac{N}{2}-1}, \mathcal{L}^N_{\frac{N}{2}} \mathbf{B}_r^{\frac{N}{2}} \right\rangle . \tag{3.3}$$

While the reductions at low and high degree look slightly different [the former requires only smoothing, but the latter requires both smoothing and the use of the recurrence (2.9)], Section 3.2 will provide a uniform formulation in which both sets of projections become instances of a Legendre transform of size $N/2$.

Finally, it is critical that the reduction to a smaller problem size may be accomplished efficiently. Setting $\mathcal{L} = \mathcal{L}^N_{\frac{N}{2}}$, if the data $\mathcal{L}\mathbf{P}_\ell$, $\mathcal{L}\mathbf{A}_r^{\frac{N}{2}}$, and $\mathcal{L}\mathbf{B}_r^{\frac{N}{2}}$ are all stored, then only the quantities $\mathcal{L}\mathbf{s}$, $\mathcal{L}\mathbf{s}^{\frac{N}{2}}$, and $\mathcal{L}\mathbf{s}^{\frac{N}{2}-1}$ need be computed. According to Lemma 2 this requires at most $3[\frac{3}{2}(N \log N + \frac{N}{2} \log \frac{N}{2})]$ operations.

Remark. In order to simplify the remaining formulas, for the remainder of the article we will continue to write $\mathcal{L}$ for $\mathcal{L}^N_{\frac{N}{2}}$, leaving N to be determined by context.

3.2 General Legendre Transforms—A Uniform Problem Formulation

The preceding discussion gives the basic idea behind the divide and conquer approach. The process indicated above must be repeated, recursively subdividing the original problem into smaller and smaller subproblems. In order to see how the pieces fit together, we

need a uniform description of the computational "unit" encountered at each division. This motivates the following definition.

Definition 1. For integers $M > 0$ and $L \geq 0$, define the $M \times 2M$ **shifted Legendre transform matrix,** LT_M^L by

$$LT_M^L = \begin{pmatrix} \left(\mathbf{A}_0^L\right)^t & \left(\mathbf{B}_0^L\right)^t \\ \left(\mathbf{A}_1^L\right)^t & \left(\mathbf{B}_1^L\right)^t \\ \vdots & \vdots \\ \left(\mathbf{A}_{M-1}^L\right)^t & \left(\mathbf{B}_{M-1}^L\right)^t \end{pmatrix} \tag{3.4}$$

where $\mathbf{A}_r^L$ and $\mathbf{B}_r^L$ are M-vectors obtained as appropriately sampled versions of A_r^L and B_r^L, the shifted Legendre polynomials defined in Section 2.3. (i. e., $\left[\mathbf{A}_r^L\right]_k = A_r^L(\cos\theta_k)$, where $\theta_k = \frac{\pi(2k+1)}{2M}$.)

The **Legendre transform (LT) of size** M **and shift** L computed from input data vectors $\mathbf{z}_0$ and $\mathbf{z}_1$ each of length M, is the matrix-vector multiplication

$$LT_M^L \cdot \mathbf{Z} \tag{3.5}$$

for $\mathbf{Z} = \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_0 \end{pmatrix}$.

The original Legendre transform of a data vector $\mathbf{s}$ of length N defined in Equation (3.1) may be written using Definition 1 as $LT_N^0 \cdot \begin{pmatrix} \mathbf{s} \\ \mathbf{0} \end{pmatrix}$ with $\mathbf{0}$ denoting the 0-vector of length N. This follows from the definitions of the 0-shifted Legendre polynomials (2.13),

$$\left\langle \mathbf{A}_r^0, \mathbf{s} \right\rangle + \left\langle \mathbf{B}_r^0, \mathbf{0} \right\rangle = \langle \mathbf{P}_r, \mathbf{s} \rangle + 0 = \langle \mathbf{P}_r, \mathbf{s} \rangle \ . \tag{3.6}$$

Likewise, the description of the splitting of the problem into low and high degree transforms may now be restated formally: the LT of size N may instead be computed as two LTs, each of size $N/2$. In particular, the low and high order transforms are computed as $LT_{\frac{N}{2}}^0 \cdot \begin{pmatrix} \mathcal{L}\mathbf{s} \\ \mathcal{L}\mathbf{0} \end{pmatrix}$ and $LT_{\frac{N}{2}}^{\frac{N}{2}} \cdot \begin{pmatrix} \mathcal{L}\mathbf{s}^{\frac{N}{2}} \\ \mathcal{L}\mathbf{s}^{\frac{N}{2}-1} \end{pmatrix}$, respectively. In this way the original discrete Legendre transform may be reduced to the computation of two smaller LTs.

In order to actually carry this out, the input for the two smaller LTs must be calculated (efficiently) from the data for the original full sized problem. To give this step a uniform description we write the input as $\mathbf{Z} = \begin{pmatrix} \mathbf{s}^0 \\ \mathbf{s}^{-1} \end{pmatrix} = \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_0 \end{pmatrix}$ with $\mathbf{s}^0 = \mathbf{s}$ and $\mathbf{s}^{-1} = \mathbf{0}$. We define **splitting operators** S^ϵ ($\epsilon = 0, 1$) which appropriately weight and then lowpass the input to provide the data for the low and high order transforms, respectively according to

$$\begin{aligned} \mathbf{Z}^\epsilon &= \mathcal{S}^\epsilon \mathbf{Z} && (3.7) \\ &= (\mathcal{L} \oplus \mathcal{L}) \cdot \mathcal{M}^\epsilon \mathbf{Z} && (3.8) \\ &= \begin{pmatrix} \mathcal{L} & \mathbf{0} \\ \mathbf{0} & \mathcal{L} \end{pmatrix} \cdot \begin{pmatrix} \operatorname{diag} \mathbf{A}^0_{\epsilon \frac{N}{2}} & \operatorname{diag} \mathbf{B}^0_{\epsilon \frac{N}{2}} \\ \operatorname{diag} \mathbf{A}^0_{\epsilon \frac{N}{2}-1} & \operatorname{diag} \mathbf{B}^0_{\epsilon \frac{N}{2}-1} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_0 \end{pmatrix} . && (3.9) \end{aligned}$$

Thus, the matrix M^ϵ is a 2×2 block matrix with $N \times N$ diagonal blocks defined by sampled values of the shifted Legendre polynomials. By convention, $\mathcal{L}$ is the $N/2 \times N$ lowpass operator. Notice that when $\epsilon = 0$, $\mathcal{M}^0$ reduces to the identity matrix. Figure 6 represents this first divide and conquer step.

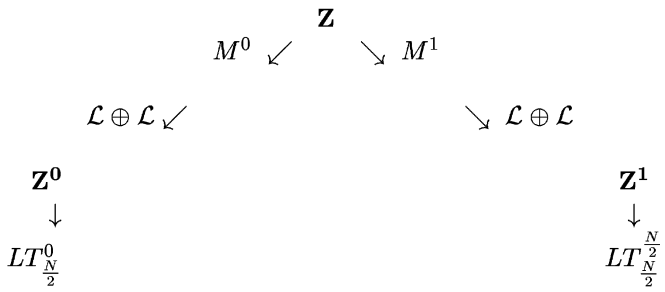


FIGURE 6 Illustration of the first split. The Legendre transform computed as two half-sized Legendre transforms of modified data.

To count carefully the number of operations used to accomplish the reduction recall first that $\mathbf{A}_0^0$ is a vector of all 1's, while $\mathbf{z}_0$, $\mathbf{B}_r^0$ and all negative subscripts give 0-vectors. Using Lemma 2 it is now easy to see that at most $2N + 3N \log N + 3\frac{N}{2} \log \frac{N}{2}$ operations are required. If we stopped subdividing at this point and efficiently completed the computation by working with the cosine transforms, then assuming that the cosine transforms of the shifted Legendre polynomials were stored, at most $\frac{9}{4}N \log \frac{N}{2} + \frac{3N^2}{8} - N$ operations would be needed to compute in this manner. This is opposed to $\frac{3}{2}N \log N + \frac{N}{2}(N-1)$ for a complete semi-naive approach. Hence we obtain an advantage for $N \geq 256$. Of course, the real algorithmic advantage is obtained by performing this split recursively. This is explained in the next subsection.

Before leaving this subsection, we record the complexity of the application of the shifted Legendre matrices. It follows from our observations in Section 2 that there is a semi-naive approach to this calculation, as observed in the following lemma.

Lemma 4.

Assuming that the cosine transforms $\mathcal{C}\mathbf{A}_r^L$ and $\mathcal{C}\mathbf{B}_r^L$ are prestored $(0 \leq r < M)$, then an LT of size M and shift L can be computed in at most $M^2 + 3M \log M + 2M$ operations.

Proof. Using [64], at most $2 \cdot (\frac{3}{2}M \log M)$ operations are needed to compute the DCTs of $\mathbf{z}_0$ and $\mathbf{z}_1$. Having done that, an additional $2 \cdot \frac{M(M+1)}{2}$ are needed to compute the inner products $\{\langle \mathcal{C}\mathbf{A}_r^L, \mathcal{C}\mathbf{z}_1\rangle, \langle \mathcal{C}\mathbf{B}_r^L, \mathcal{C}\mathbf{z}_0\rangle \;\big|\; r = 0, \ldots, M-1\}$ and an additional M operations to add together each pair. □

Remark. Lemma 4 is precisely the computation of a LT of size M and shift L by the semi-naive algorithm. Notice that it too may be cast as a particular matrix factorization,

$$LT_M^L \cdot \mathbf{Z} = \left(LT_M^L (\mathcal{C}_M \oplus \mathcal{C}_M)^t\right) \cdot (\mathcal{C}_M \oplus \mathcal{C}_M) \cdot \mathbf{Z} \tag{3.10}$$

recalling that we use the orthogonal matrix $\mathcal{C}_M$ (2.16) to effect the DCT and assume that the first factor is precomputed.

3.3 Recursive Subdivision

For a complete recursive subdivision we need the following lemma.

Lemma 5 (Splitting Lemma).
Let M be a positive integer divisible by two.

(i) *A Legendre transform of size M can be computed as two Legendre transforms of size $M/2$. Specifically, multiplication of LT_M^L against a given input can instead be computed as the separate multiplications of $LT_{\frac{M}{2}}^{L}$ and $LT_{\frac{M}{2}}^{L+\frac{M}{2}}$ against new inputs obtained from the original input.*

(ii) *Let $\mathbf{Z} = \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_0 \end{pmatrix}$ be the initial data for a LT of size M and shift L. Then at most $8M + 6M \log M + 3M \log \frac{M}{2}$ operations are needed to compute the necessary inputs for the pair of LTs at size $M/2$ which together, compute the original LT. In particular, if $\mathbf{Z}^\epsilon = \begin{pmatrix} \mathbf{z}_1^\epsilon \\ \mathbf{z}_0^\epsilon \end{pmatrix}$ $(\epsilon = 0, 1)$ are the necessary input for the half-sized LTs which compute the low order $(\epsilon = 0)$ and high order $(\epsilon = 1)$ transforms, then*

$$\begin{aligned} \mathbf{Z}^\epsilon &= \mathcal{S}_{L,M}^\epsilon \cdot \mathbf{Z} && (3.11)\\ &= (\mathcal{L} \oplus \mathcal{L}) \cdot \mathcal{M}^\epsilon \cdot \mathbf{Z} && (3.12)\\ &= \begin{pmatrix} \mathcal{L} & \mathbf{0} \\ \mathbf{0} & \mathcal{L} \end{pmatrix} \cdot \begin{pmatrix} \operatorname{diag} \mathbf{A}^L_{\epsilon \frac{M}{2}} & \operatorname{diag} \mathbf{B}^L_{\epsilon \frac{M}{2}} \\ \operatorname{diag} \mathbf{A}^L_{\epsilon \frac{M}{2}-1} & \operatorname{diag} \mathbf{B}^L_{\epsilon \frac{M}{2}-1} \end{pmatrix} \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_0 \end{pmatrix}. && (3.13) \end{aligned}$$

(iii) *The matrix LT_M^L has the factorization*

$$LT_M^L = \begin{pmatrix} LT_{\frac{M}{2}}^{L+\frac{M}{2}} & \\ & LT_{\frac{M}{2}}^{L} \end{pmatrix} \cdot \begin{pmatrix} \mathcal{S}_{L,M}^1 \\ \mathcal{S}_{L,M}^0 \end{pmatrix} \qquad (3.14)$$

where $\mathcal{S}_{L,M}^0$ and $\mathcal{S}_{L,M}^1$ are defined as in (ii).

Proof. The low degree projections in the size M problem, $\langle \mathbf{A}_k^L, \mathbf{z}_1 \rangle + \langle \mathbf{B}_k^L, \mathbf{z}_0 \rangle$, $0 \le k < M/2$ only involve shifted Legendre polynomials of degree less than $M/2$. Consequently,

according to Lemma 1 these inner products may be computed as inner products of the lowpassed data with subsampled versions of the shifted Legendre polynomials, thereby reducing it to a LT of size $M/2$.

To reduce the set of high order projections to a LT of half the size, we apply the recurrence formula (2.12) to obtain

$$\begin{aligned}\mathbf{A}^L_{\frac{M}{2}+k} &= \mathbf{A}_k^{L+\frac{M}{2}}\mathbf{A}^L_{\frac{M}{2}} + \mathbf{B}_k^{L+\frac{M}{2}}\mathbf{A}^L_{\frac{M}{2}-1}\,,\\ \mathbf{B}^L_{\frac{M}{2}+k} &= \mathbf{A}_k^{L+\frac{M}{2}}\mathbf{B}^L_{\frac{M}{2}} + \mathbf{B}_k^{L+\frac{M}{2}}\mathbf{B}^L_{\frac{M}{2}-1}\,.\end{aligned}$$

Consequently, we rewrite the high order transforms as

$$\begin{aligned}\left\langle \mathbf{A}^L_{\frac{M}{2}+k}, \mathbf{z}_1\right\rangle + \left\langle \mathbf{B}^L_{\frac{M}{2}+k}, \mathbf{z}_0\right\rangle &= \left\langle \mathbf{A}_k^{L+\frac{M}{2}}, \mathbf{A}^L_{\frac{M}{2}}\mathbf{z}_1 + \mathbf{B}^L_{\frac{M}{2}}\mathbf{z}_0\right\rangle\\ &\quad + \left\langle \mathbf{B}_k^{L+\frac{M}{2}}, \mathbf{A}^L_{\frac{M}{2}-1}\mathbf{z}_1 + \mathbf{B}^L_{\frac{M}{2}-1}\mathbf{z}_0\right\rangle\,. \qquad (3.15)\end{aligned}$$

Since both $A_k^{L+\frac{M}{2}}$ and $B_k^{L+\frac{M}{2}}$ have degree less than $\frac{M}{2}$ we can lowpass both sides of each inner product in (3.15) and obtain

$$\begin{aligned}\left\langle \mathbf{A}^L_{\frac{M}{2}+k}, \mathbf{z}_1\right\rangle + \left\langle \mathbf{B}^L_{\frac{M}{2}+k}, \mathbf{z}_0\right\rangle &= \left\langle \mathcal{L}\mathbf{A}_k^{L+\frac{M}{2}}, \mathcal{L}\left(\mathbf{A}^L_{\frac{M}{2}}\mathbf{z}_1 + \mathbf{B}^L_{\frac{M}{2}}\mathbf{z}_0\right)\right\rangle\\ &\quad + \left\langle \mathcal{L}\mathbf{B}_k^{L+\frac{M}{2}}, \mathcal{L}\left(\mathbf{A}^L_{\frac{M}{2}-1}\mathbf{z}_1 + \mathbf{B}^L_{\frac{M}{2}-1}\mathbf{z}_0\right)\right\rangle\,. \qquad (3.16)\end{aligned}$$

This completes the proof of (i). The proof of (ii) follows immediately. The stated form of $\mathbf{Z}^\epsilon$ follows from simply rewriting the collection of linear equations describing the new data. The complexity result follows from the fact that application of $\mathcal{M}^\epsilon$ requires at most $4M$ operations and application of $\mathcal{L}\oplus\mathcal{L}$, an additional $3(M\log M + \frac{M}{2}\log\frac{M}{2})$ operations (cf. Lemma 2).

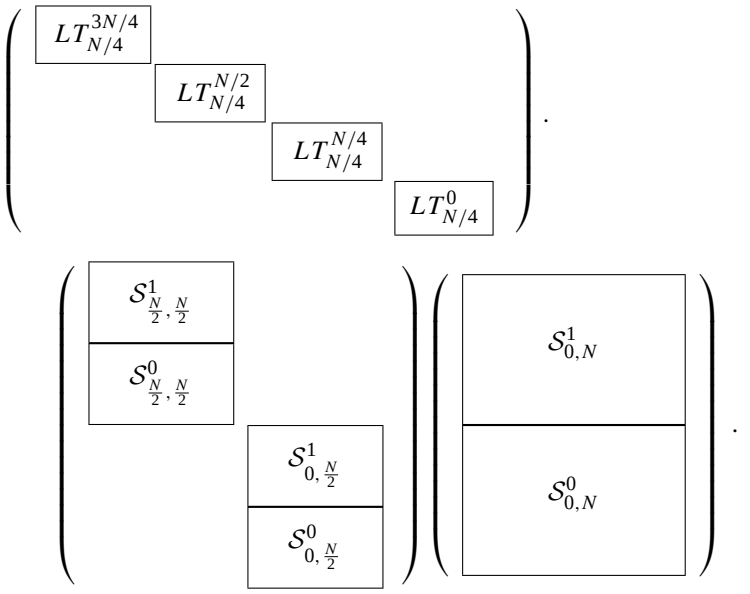
Finally, (iii) is simply a restatement of (ii) in the setting of matrix arithmetic. □

Using Lemma 5 we can now describe the full algorithm in a succinct way. Starting with the original Legendre transform written as LT_N^0, part (iii) implies that this matrix factors as

$$LT_N^0 = \begin{pmatrix} LT^{\frac{N}{2}}_{\frac{N}{2}} & \\ & LT^0_{\frac{N}{2}}\end{pmatrix}\begin{pmatrix}\mathcal{S}^1_{0,N}\\ \mathcal{S}^0_{0,N}\end{pmatrix}\,. \qquad (3.17)$$

Now we apply Lemma 5 to the half-sized LT matrices, $LT^0_{\frac{N}{2}}$ and $LT^M_{\frac{N}{2}}$ producing the

factorization of LT_N^0 as



To keep track of the factorization as we continue splitting, we'll use a binary tree-based indexing notation, indicated schematically in Figure 7. Given initial data $\mathbf{s}$ of length N, define the block vector of length $2N$, $\mathbf{Z} = \begin{pmatrix} \mathbf{s} \\ \mathbf{0} \end{pmatrix}$. We will factor the matrix LT_N^0 as a product of $\log N$ block diagonal matrices such that for each k, $1 \le k \le \log N$, the k^{th} matrix has 2^k blocks made up of splitting matrices of the type described in the Splitting Lemma (Lemma 5). These splitting matrices will be indexed by binary k-tuples $\vec{\epsilon} = (\epsilon_1, \epsilon_2, \ldots, \epsilon_k)$, $\vec{\epsilon} \in \{0,1\}^k$, denoted as $\mathcal{S}^{\vec{\epsilon}}$ and defined by

$$\mathcal{S}^{\vec{\epsilon}} = (\mathcal{L} \oplus \mathcal{L}) \cdot M^{\vec{\epsilon}} \tag{3.18}$$

for

$$\mathcal{M}^{\vec{\epsilon}} = \mathcal{M}^{(\epsilon_1,\epsilon_2,\ldots,\epsilon_k)} = \begin{pmatrix} \operatorname{diag} \mathbf{A}^{L(\epsilon_1,\ldots,\epsilon_{k-1})}_{\epsilon_k \frac{N}{2^{k-1}}} & \operatorname{diag} \mathbf{B}^{L(\epsilon_1,\ldots,\epsilon_{k-1})}_{\epsilon_k \frac{N}{2^{k-1}}} \\ \operatorname{diag} \mathbf{A}^{L(\epsilon_1,\ldots,\epsilon_{k-1})}_{\epsilon_k \frac{N}{2^{k-1}}-1} & \operatorname{diag} \mathbf{B}^{L(\epsilon_1,\ldots,\epsilon_{k-1})}_{\epsilon_k \frac{N}{2^{k-1}}-1} \end{pmatrix} \begin{matrix} \uparrow \\ 2 \cdot \frac{N}{2^{k-1}} \\ \downarrow \end{matrix} \quad . \tag{3.19}$$

$$\overset{\longleftarrow \quad 2 \cdot \frac{N}{2^{k-1}} \quad \longrightarrow}{}$$

and $L(\vec{\epsilon}) = L(\epsilon_1, \epsilon_2, \ldots, \epsilon_m) = \epsilon_1 \frac{N}{2} + \epsilon_2 \frac{N}{4} + \cdots + \epsilon_m \frac{N}{2^m}$.

With this notation and Lemma 5 we now have the following.

Theorem 2.

Let $N = 2^r$ and let $\mathbf{s}$ be any vector of length N.

(i) The Legendre transform of $\mathbf{s}$ may be computed from $LT_N^0 \cdot \begin{pmatrix} \mathbf{s} \\ \mathbf{0} \end{pmatrix}$ in $O(N \log^2 N)$

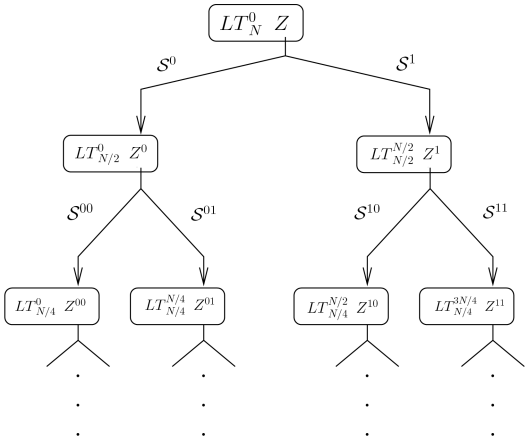


FIGURE 7 Schematic illustration of the computation of the Legendre transform by recursive splitting.

operations via the factorization: $LT_N^0 = \mathcal{E}\mathcal{F}_{r-1}\cdots\mathcal{F}_1$, *where*

$$\mathcal{F}_j = \begin{pmatrix} \mathcal{S}^{(1,\ldots,1,1)} & & & \\ \mathcal{S}^{(1,\ldots,1,0)} & & & \\ & \mathcal{S}^{(1,\ldots,0,1)} & & \\ & \mathcal{S}^{(1,\ldots,0,0)} & & \\ & & \ddots & \\ & & & \mathcal{S}^{(0,\ldots,0,1)} \\ & & & \mathcal{S}^{(0,\ldots,0,0)} \end{pmatrix} \tag{3.20}$$

and the 2^j *splitting matrices* $\mathcal{S}^{\vec{\epsilon}}$ *are defined as in* (3.18), *and*

$$\mathcal{E} = LT_2^{N-2} \oplus LT_2^{N-4} \oplus \cdots \oplus LT_2^2 .$$

(ii) Assuming the precomputation and storage of the masks $\mathcal{M}^{\vec{\epsilon}}$ [*see* (3.19)] *as well as the cosine transforms of the shifted Legendre polynomials* $A_r^{L(\vec{\epsilon})}$, $B_r^{L(\vec{\epsilon})}$, *then the complexity of the Legendre transform of* $\mathbf{s}$ *is* $O(N\log^2 N)$. *More precisely, at most*

$$8\cdot\frac{N}{2} + \sum_{a=1}^{r-1}\left(2^a-1\right)\cdot 2\cdot\frac{3}{2}\left[\frac{N}{2^a}\log\frac{N}{2^a} + \frac{N}{2^{a-1}}\log\frac{N}{2^{a-1}}\right] + \left(2^a-2\right)\cdot 4\cdot\frac{N}{2^{a-1}}$$

operations are needed to compute the Legendre transform of $\mathbf{s}$.

Proof. Part (i) follows directly from a recursive application of Lemma 5. As for (ii), the first term in the computation follows from the fact that $\mathcal{E}$ is block diagonal with $\frac{N}{2}$ 2×4 matrices on the diagonal, each requiring 8 operations to multiply by a vector of length 4. The summation which is the second term is the complexity of the successive multiplications of the $\mathcal{F}_j$. This follows directly from the form of the $\mathcal{S}^{\vec{\epsilon}}$. In particular noting that in general (for $\vec{\epsilon} \in \{0, 1\}^a, \vec{\epsilon} \neq \mathbf{0}$), $2 \cdot \frac{3}{2}\left[\frac{N}{2^a} \log \frac{N}{2^a} + \frac{N}{2^{a-1}} \log \frac{N}{2^{a-1}}\right] + 4 \cdot \frac{N}{2^{a-1}}$ operations are required to apply $\mathcal{S}^{\vec{\epsilon}}$ to an arbitrary input. When $\vec{\epsilon} = \mathbf{0}$, we may take advantage of the fact that the lower half of the input vector will be all 0 and that the relevant diagonal blocks of $\mathcal{M}^{\vec{\epsilon}}$ will be the identity and all 0's as well. Summing the complexities at each level gives (ii). □

The inductive nature of Theorem 2 shows that the algorithm is simply given by the successive application of the splitting operators to the (changing) input. In practice, this splitting is applied as long as it offers computational advantage. At this point the calculation may be concluded by applying the shifted Legendre matrices according to the semi-naive algorithm.

4. Fast Fourier Transform, Inverse Transform, and Convolution for the 2-Sphere

The algorithm presented in the previous section enables us to write a fast algorithm for the efficient calculation of spherical harmonic expansion, or Fourier transform, of a function on S^2 in terms of its samples on a regular grid. This is readily converted into a fast inversion algorithm (transforming a set of Fourier coefficients into sample values) improving on the results of [18]. These two algorithms combine to give an improved convolution algorithm.

4.1 Fast Fourier Transform for S^2

Collecting the results of previous sections gives the following theorem.

Theorem 3.

Let B be a power of 2 and $n = B^2$. If $f(\theta, \phi)$ is in the span of $\{Y_\ell^m \mid |m| \leq \ell < B, \}$, then the n Fourier coefficients $\hat{f}(\ell, m)$ for $\ell < B, |m| \leq \ell$ can be computed in $O(n \log^2 n)$ operations from the $4n$ sampled values $f((2j+1)\pi/4B, 2\pi k/2B)$, $0 \leq j, k \leq 2B-1$, using a precomputed data structure of size $O(n \log^2 n)$.

Proof. From the Sampling Theorem we know that the Fourier transform may be computed by means of finite sums,

$$\hat{f}(\ell, m) = \sum_{j=0}^{2B-1} \sum_{k=0}^{2B-1} a_j^{(B)} f(\theta_j, \phi_k) Y_\ell^m(\theta_j, \phi_k)$$

where $\theta_j = \pi(2j+1)/4B$ and $\phi_k = 2\pi k/2B$, and the $a_j^{(B)}$ are as defined in the Sampling Theorem. Rewriting, we obtain

$$\hat{f}(\ell, m) = q_\ell^m \sum_{j=0}^{2B-1} a_j^{(B)} P_\ell^m(\cos\theta_j) \sum_{k=0}^{2B-1} e^{-im\phi_k} f(\theta_j, \phi_k) \,,$$

where the q_ℓ^m are the normalization coefficients for the spherical harmonics. The inner sums (defined as $\breve{f}(\theta_j, m)$) are computed for each fixed j and for all m in the appropriate range by means of a fast Fourier transform. This will require $O(B^2 \log B)$ operations.

It remains to compute

$$\hat{f}(\ell, m) = q_\ell^m \sum_{j=0}^{2B-1} a_j^{(B)} \breve{f}(\theta_j, m) P_\ell^m(\cos\theta_j)$$

which has the form of an associated Legendre transform for each fixed m. Each transform can be accomplished in $O(B \log^2 B)$ operations, and since m ranges over $2B-1$ values, the total number of operations needed to compute all values of $\hat{f}(\ell, m)$ is $O(B^2 \log^2 B)$. □

4.2 Fast Inversion

For our purposes, the inverse transform is the map which takes as input a set of complex numbers $c_{\ell,m}$, interpreted as Fourier coefficients in a spherical harmonic expansion and returns a set of sample values. The discussion is restricted to band-limited functions so we consider only the case in which the $c_{\ell,m}$ vanish for $\ell \geq B$ for some bandwidth $B > 0$.

Definition 2. The **discrete inverse spherical Fourier transform** with bandwidth $B = 2^b$ maps a collection of complex coefficients $c_{\ell,m}$, $0 \leq m \leq \ell < B$ to a collection of samples values via the formula

$$f(\theta_j, \phi_k) = \sum_{\ell \geq 0} \sum_{|m| \leq \ell} c_{\ell,m} Y_\ell^m(\theta_j, \phi_k) \ ,$$

where the recovery is on the equiangular grid of the sampling theorem, $\theta_j = \frac{(j+1/2)\pi}{2B}$, $j = 0, \ldots, 2B-1$; and $\phi_k = \frac{(k+1/2)\pi}{B}$, $k = 0, \ldots, 2B-1$.

Setting $n = B^2$, this map transforms a collection of $O(n)$ Fourier coefficients to $O(n)$ samples of the function with the associated spherical harmonic expansion. Computed directly, this would require $O(n^2)$ calculations, or $O(n^{3/2})$ by using a little reorganization (cf. [18], Theorem 9). Our new result is that in fact, this too may be accomplished in $O(n \log^2 n)$ operations.

Theorem 4.

Let $B = 2^r$ be a fixed bandwidth. The discrete inverse spherical Fourier transform for this bandwidth may be computed in $O(n \log^2 n)$ operations, where $n = B^2$.

Proof. Let the Fourier coefficients $c_{\ell,m}$, $0 \leq m \leq \ell < B$ be given. Our goal is to compute the collection of samples of the inverse spherical Fourier transform

$$f(\theta_j, \phi_k) = \sum_{\ell=0}^{B-1} \sum_{|m| \leq \ell} c_{\ell,m} Y_\ell^m(\theta_j, \phi_k) \ .$$

Using the definition of the Y_ℓ^m we rewrite this as

$$\sum_{|m|<B} e^{-im\phi_j} \sum_{\ell=|m|}^{B-1} c_{\ell,m} \, q_\ell^m \, P_\ell^m(\cos\theta_k) \ , \tag{4.1}$$

where q_ℓ^m denotes the appropriate normalization constant. Let $h(\theta_k, m)$ denote the inner sum

$$h(\theta_k, m) = \sum_{\ell=|m|}^{B-1} c_{\ell,m}\, q_\ell^m\, P_\ell^m(\cos\theta_k)\ .$$

Computed directly, each of the $4B^2$ values for h would require a summation of $O(B)$ terms for a naive complexity of $O(B^3)$. Instead we show how a simple "adaptation" of the ideas of Section 3 yields a more efficient algorithm. Observe that the column vector $\mathbf{h}^m = (h(\theta_0, m), \ldots, h(\theta_{2B-1}, m))^t$ is obtained as a matrix-vector product

$$\mathbf{h}^m = \begin{pmatrix} h(\theta_0, m) \\ h(\theta_1, m) \\ \vdots \\ h(\theta_{2B-1}, m) \end{pmatrix} = \begin{pmatrix} P_m^m(\cos\theta_0) & \cdots & P_{B-1}^m(\cos\theta_0) \\ P_m^m(\cos\theta_1) & \cdots & P_{B-1}^m(\cos\theta_1) \\ \vdots & \vdots & \vdots \\ P_m^m(\cos\theta_{2B-1}) & \cdots & P_{B-1}^m(\cos\theta_{2B-1}) \end{pmatrix} \cdot \begin{pmatrix} c_{m,m} q_m^m \\ c_{m+1,m} q_m^{m+1} \\ \vdots \\ c_{B-1,m} q_m^{B-1} \end{pmatrix}$$

$$= \tilde{\mathcal{P}}_B^m \cdot \tilde{\mathbf{c}^m}$$

where $\tilde{\mathcal{P}}_B^m$ and $\tilde{\mathbf{c}^m}$ are defined by the above equations. Notice that $(\tilde{\mathcal{P}}_B^m)^t$ is the Vandermonde-like matrix for the order m associated Legendre functions, and as such is the transpose of the matrix whose application to a fixed set of sample values yields the associated Legendre transform.

The key observation we now make is that the algorithm described in Section 3 for computing the Fourier transform gives a factorization of the matrix $\tilde{\mathcal{P}}_B^m$ as a product of matrices whose structure admit efficient multiplication against an arbitrary vector. More precisely, Section 3 shows that we have a factorization

$$\tilde{\mathcal{P}}_B^m = \mathcal{M}_{r-1} \cdots \mathcal{M}_1 \mathcal{M}_0$$

such that a total of $O(B \log^2 B)$ operations are required to successively multiply the $\mathcal{M}_i$ against an arbitrary vector. A similar complexity result holds if instead, the *transpose* of each of the matrices is considered, and the product is reordered. Since $(\tilde{\mathcal{P}}_B^m)^t = \mathcal{M}_0^t \mathcal{M}_1^t \cdots \mathcal{M}_{r-1}^t$ it follows that the inverse transform may be computed in $O(B \log^2 B)$ operations as well.

To complete the computation, an abelian FFT is performed to compute the sums $\sum_m h(\theta_k, m) e^{-im\phi_k}$ in $O(B \log B)$ operations for each k, for a total additional cost of $O(B^2 \log B)$. Denoting the total number of samples by $n = B^2$, we may write the total cost as $O(n \log^2 n)$, as desired. □

4.3 A Fast Spherical Convolution Algorithm

Taken together, the fast forward and inverse transforms allow for the convolution of two band-limited functions in $L^2(S^2)$ to be computed efficiently—and exactly (in exact arithmetic).

Defining convolution for two functions in $L^2(S^2)$ uses the structure of S^2 as a quotient of the group $SO(3)$. Generalizing the case of the circle, the **left convolution of h by f for** $f, h \in L^2(S^2)$, is defined as

$$f \star h(\omega) = \int_{g \in SO(3)} f(g\eta) h\left(g^{-1}\omega\right)\, dg\ . \tag{4.2}$$

In (4.2) dg denotes the (essentially) unique invariant volume form on $SO(3)$ and η denotes the north pole.

By combining the fast expansion and synthesis algorithms of Sections 3 and 4.2, a fast convolution algorithm is obtained. Given the sample values of two functions $f, g \in L^2(S^2)$ of bandwidth B, we now give an $O(n \log^2 n)$ algorithm ($n = B^2$) to compute the sample values of the convolution $f \star g$ thereby improving on the $O(n^{1.5})$ algorithm in [18].

As in the more familiar case of convolution on the circle via the abelian FFT [8], the spherical convolution algorithm may be decomposed into three basic steps:

(1) Computation of a forward transform;
(2) Pointwise multiplication of the appropriate transforms;
(3) Computation of the inverse transform of the result of step (2).

We have shown how to accomplish Steps (1) and (3) efficiently. We need only satisfy Step (2). This uses the following relationship between the transform and the convolution.

Theorem 5 ([18], Theorem 1).
Let $f, h \in L^2(S^2)$. Then

$$\widehat{(f \star h)}(l, m) = 2\pi \sqrt{\frac{4\pi}{2l+1}}\, \hat{f}(l, m)\hat{h}(l, 0) \ .$$

In particular, the convolution of functions of bandwidth B, yields a function of bandwidth B.

Theorem 6.
Let $f, g \in L^2(S^2)$ such that by f and g are band-limited with bandwidth B. Then the $n = O(B^2)$ sample values of the convolution $f \star g$ at the points (θ_j, ϕ_k) where $\theta_j = \frac{\pi(j+1/2)}{2B}$ and $\phi_k = \frac{\pi(k+1/2)}{B}$ may be computed in $O(n \log^2 n)$ operations, versus the $O(n^2)$ operations required by direct computation.

Proof. Using the algorithm described in Section 3, compute the Fourier coefficients of f and g. Compute the pointwise products according to Theorem 5 for the convolution $f \star g$ and finally, compute the inverse transform for the Fourier coefficients of the convolution according to Theorem 4. □

5. Variations of the Basic Algorithm

The recursive splitting described in Section 3 computes the discrete Legendre transform of a vector of length N in $O(N \log^2 N)$ operations, assuming that the splitting is carried out as far as possible, i. e., to $\log N$ levels. By applying the analogous algorithms to the samples of a function on S^2 with bandwidth B, for the remaining Legendre functions P_ℓ^m, $m \neq 0$, the projections onto the spherical harmonics Y_ℓ^m, for each ℓ in the range $0 \leq \ell < B$ and $|m| \leq \ell$ are then efficiently obtained. This is the "basic" Driscoll–Healy algorithm, which we subsequently refer to as the **DH algorithm** or **DH**. Our discussion shows that **DH** computes the Fourier coefficients of a function on S^2 of bandwidth B in $O(n \log^2 n)$ operations for $n = B^2$.

This is an asymptotic result, exact in exact arithmetic. For application to actual problems of moderate size, we must consider issues of numerical reliability and compu-

tational efficiency in a floating point implementation. We must also demonstrate that an implementation can obtain real speed-ups over existing algorithms at useful problem sizes.

We now turn to a discussion of some simple variations of our basic approach which we use to obtain fast and reliable algorithms at various moderate problem sizes. Again we specialize to the Legendre polynomial case.

• **Variation 1.** Using the reverse recurrence: The **DH-Mid** algorithm.

This simple variation reduces the complexity by approximately one-half. The three-term recurrence (2.8) is a forward recurrence. This also gives rise to the *reverse* recurrence

$$P_{\ell-1}(x) = \frac{2\ell+1}{\ell} x P_\ell(x) - \frac{\ell+1}{\ell} P_{\ell+1}(x) \ . \tag{5.1}$$

Using (5.1) in a way analogous to the use of (2.8) we may define the **reverse shifted Legendre polynomials** in analogy with the functions A_r^L and B_r^L defined for the forward direction (2.9). For any fixed level L, iterating the recurrence formula (5.1) back r steps produces trigonometric polynomials A_{-r}^L and B_{-r}^L such that

$$P_{L-r}(\cos\theta) = A_{-r}^L(\cos\theta)\ P_{L-1}(\cos\theta) + B_{-r}^L(\cos\theta)\ P_L(\cos\theta) \ , \tag{5.2}$$

for $r \geq 2$. This may be used in several ways to reduce computations.

First, this can provide a "balanced" version of the semi-naive algorithm. Section 3 shows that, given the vectors $\mathbf{s}^{L-1}$ and $\mathbf{s}^L$, the Legendre coefficient of degree $L+r$ can be computed in $O(r)$ operations by passing to the cosine transform domain and forming the inner product with the cosine transform vector of the shifted Legendre functions of degree at most r. Similarly, the Legendre coefficient of degree $L-r$ can be computed in $O(r)$ operations using reverse shifted Legendre polynomials of degree bound r. Again, simply form the inner products of the cosine transforms of $\mathbf{s}^{L-1}$ and $\mathbf{s}^L$ with the cosine transforms of the reverse shifted Legendre polynomials. In this way, we can compute a semi-naive algorithm for the Legendre coefficients in the range $[L-r, L+r]$ which is "balanced" in the sense that the number of operations required to compute any given transform in this interval depends only on the distance of the index from L. Furthermore, only the cosine transform coefficients of $\mathbf{s}^{L-1}$ and $\mathbf{s}^L$ of degree r or less are used to compute these Legendre coefficients.

We may also use this approach to expedite the splitting steps of the algorithm by starting in the middle of the sequence and moving out to both the left and right from the initial data. This is done as follows. From the initial data $\mathbf{s}^0$ we compute the lowpassed subsampled sequences $\mathbf{s}^{N/2}$, $\mathbf{s}^{N/2+1}$, each of length $N/2$. Then $\mathbf{s}^{3N/4}$ and $\mathbf{s}^{3N/4+1}$ are computed as before, but by using the three-term recurrence in the reverse direction (5.1) the sequences $\mathbf{s}^{N/4}$ and $\mathbf{s}^{N/4+1}$ can be computed. The additional savings come from using the initial data to compute **four** new sequences instead of two.

At the next stage, each of the pairs $\mathbf{s}^{N/4}$, $\mathbf{s}^{N/4+1}$ and $\mathbf{s}^{3N/4}$, $\mathbf{s}^{3N/4+1}$ then act similarly as initial data for obtaining the sequences $\mathbf{s}^j$, $j = N/8, N/8+1, 3N/8, 3N/8+1$ and $\mathbf{s}^j$, $j = 5N/8, 5N/8+1, 7N/8, 7N/8+1$, respectively. The recursion continues down to some base case. We call this algorithm the **DH-Mid** algorithm.

Asymptotically and theoretically, a full divide and conquer strategy is optimal. However, in actual implementations overhead costs can accrue and it is often the case that for smaller problem sizes divide and conquer is no longer advantageous. Below this "breakeven point" a more direct approach may be faster. We may successfully address this issue with some of the simple variants discussed below. This basic idea uses a simple truncation of the splitting of the basic algorithm at an appropriate level.

• **Variation 2.** The Bounded **DH-Mid** Algorithm.

The semi-naive algorithm described in Section 3.2 is quite competitive in speed for moderate problems ($N \leq 256$). We need to take this into consideration when optimizing algorithms for problems of moderate size encountered in applications ($256 \leq N \leq 1024$). In this range, we will find it useful to stop the splitting when the resulting subproblems reach a given size.

Consequently, in this further variation, we use **DH-Mid** but only split down to a fixed level k, i. e., to a point at which all of the most recently computed sequences $\mathbf{s}^j$ have length $2k$. At this point we switch over to a semi-naive approach to compute the remaining Legendre transforms. For example, suppose the recursion halts after we have computed a group of length $2k$ sequences which includes $\mathbf{s}^r$, $\mathbf{s}^{r+1}$ and $\mathbf{s}^{r+k}$, $\mathbf{s}^{r+k+1}$. By using a semi-naive method and the shifted Legendre and reverse shifted Legendre functions, $\mathbf{s}^r$, $\mathbf{s}^{r+1}$ can be used to compute the vectors $\mathbf{s}^j$, $j = r-k/2+1, r-k/2+2, \ldots, r-1, r+2, \ldots, r+k/2-1$. Similarly, $\mathbf{s}^{r+k}$, $\mathbf{s}^{r+k+1}$ can be used to compute $\mathbf{s}^j$, $j = r+k/2, r+k/2+1, \ldots, r+k-1, r+k+2, \ldots, r+k+k/2-1$. The value of k can be chosen for a given problem size so as to minimize the number of operations.

• **Variation 3.** Simple split algorithm.

Here the idea is to immediately split the original problem of length N into C subproblems and run a semi-naive approach on each of the resulting subproblems. The input data for each of the subproblems is computed directly from the original input data by multiplying all N samples of $\mathbf{s}$ onto the appropriate samples of the Legendre functions $\mathbf{P}_\ell$. In terms of the tree description of **DH** or **DH-Mid** (cf. Figure 7) this simply truncates the tree immediately by computing only one level. In contrast, the other algorithms we have described compute reduced size input data vectors for the subproblems recursively from the coarser splitting at the previous level of the tree.

This has the advantage of simplicity, but at the expense of increased complexity. Indeed, we shall see that

$$O\left(N^{\frac{3}{2}} \log^{\frac{1}{2}} N\right) \tag{5.3}$$

are required for the full transform.

From our description of the backward recurrence given above, we know that, after computing the cosine transforms of the vectors $\mathbf{s}^{L-1}$ and $\mathbf{s}^L$, each of the Legendre coefficients with degree in the interval $[L-r, L+r]$ can be computed in $O(r)$ further operations. This is done by forming the inner product against the cosine transform vector of the appropriate shifted or reverse shifted Legendre functions, each of which has degree less than or equal to r. Furthermore, only the cosine transform elements of $\mathbf{s}^{j-1}$ and $\mathbf{s}^j$ of degree r or less are used to compute the Legendre coefficients with degree in the interval $[L-r, L+r]$. In this way, we can obtain a very simple algorithm by splitting the original problem into subproblems in which matrices derived from (reverse) shifted Legendre functions are applied to some fixed number C of cosine transform vector pairs $\mathcal{C}\mathbf{s}_{L-1}, \mathcal{C}\mathbf{s}_L$, where L is evenly spaced in the range $[0, N-1]$. Using analysis and experimentation, a value of C can be chosen which minimizes the execution time.

Specifically, given a fixed value for C, we can evenly space the values of L by letting

$$L = \frac{(2j+1)N}{2C} + 1, \quad j = 0, \ldots C-1\,. \tag{5.4}$$

It is easy to see that, given $\mathcal{C}\mathbf{s}_{L-1}, \mathcal{C}\mathbf{s}_L$ as well as all of the required cosine transformed shifted Legendre vectors, $\mathcal{C}\mathbf{A}_r^L, \mathcal{C}\mathbf{N}_r^L$, $(0 < |r| < \frac{N}{2C})$ then

$$\frac{N^2}{4C^2} + \frac{N}{2C} - 2$$

operations are needed to compute the Legendre coefficients $\langle f, P_{L+r} \rangle$ for $|r| < \frac{N}{2C}$. With C vector pairs $\{\mathcal{C}\mathbf{s}_{L-1}, \mathcal{C}\mathbf{s}_L\}$ the total cost of computing Legendre coefficients using shifted Legendre functions is

$$\frac{N^2}{4C} + \frac{N}{2} - 2C \ . \tag{5.5}$$

Note that the assumption that the data structure of cosine transformed shifted Legendre vectors is available, i. e., precomputed and stored, does not change the order of complexity of the algorithm. All of these vectors can be computed in $O(\frac{N^2}{C})$, and stored in a data structure of size $O(\frac{N^2}{C})$.

It remains to determine the complexity of computing the C vector pairs $\{\mathcal{C}\mathbf{s}_{L-1}, \mathcal{C}\mathbf{s}_L\}$. These vectors are obtained by the fast cosine transform of $f\mathbf{P}_L$, where f and P_L are sequences of length $2N$. Also, only the first $\frac{N}{2C}$ coefficients of the cosine series are needed. Using a cosine transform algorithm derived from [64], each vector pair requires

$$2N \ \log \ \tfrac{N}{2C}$$

multiplications, which gives a total cost for all vector pairs of

$$2NC \ \log \ \frac{N}{2C} \ . \tag{5.6}$$

Combining Equations (5.5) and (5.6) gives the total cost of computing the Legendre coefficients as

$$\frac{N^2}{4C} + 2NC \ \log \ \frac{N}{2C} + \frac{N}{2} - 2C \tag{5.7}$$

in terms of the number of multiplications. Minimizing for C in Equation (5.7) gives

$$C = O\left(\frac{N^{\frac{1}{2}}}{\log^{\frac{1}{2}} N}\right)$$

and substituting this value of C into Equation (5.7) gives

$$O\left(N^{\frac{3}{2}} \ \log^{\frac{1}{2}} N\right) \tag{5.8}$$

for the cost of the algorithm.

This algorithm may be an attractive candidate for parallelization as the computation of each vector pair, and the Legendre coefficients derived from them, are independent of other vector pairs, but the same sequence of arithmetic operations is used.

• **Variation 4.** Hybrid algorithms.

In this approach, the semi-naive algorithm is used to compute Legendre transforms of degrees m (where m is the order of the transform) through r for some fixed bound r and then the simple-split algorithm is used to compute the remaining Legendre transforms of degrees $r + 1$ through $N - 1$.

While the order of complexity for this algorithm is greater than the **DH** variants, the constants and the overhead are small, and we will show in Section 6 that this algorithm performs quite well in practice.

6. Numerical Results

We focused our attention on the semi-naive, simple split and hybrid algorithms. Initial testing of the various, non-direct, algorithms revealed that for moderate problem sizes, regardless of the platform, the basic **DH** algorithm was the slowest. Though theoretically the basic algorithm is optimal, we found the cost of applying the smoothing operator to be expensive. This was one reason why variations of the basic algorithm were developed. Another reason, which will be discussed later, concerned stability.

Experiments were performed on a plethora of platforms of various vintages: a DEC Alpha 500/200, an HP Exemplar X-Class SPP2000/64, a SGI Origin 2000, and a Linux Pentium 3 workstation. Even though the HP and SGI machines both have parallel architectures, our C code never took advantage of this. All tests were run on a single processor. The code, freely available for download as the software package SpharmonicKit [61], was compiled using the native compiler with available optimizations. Furthermore, when testing on all but the Linux platform, we were able to "patch in" the very efficient FFT and DCT routines provided by FFTPACK [22] and hence improve the performance of all the algorithms tested. We were unable to use FFTPACK on Linux because the GNU g77 compiler lacked the options necessary to properly compile and link the Fortran code with the C code. In all the results we present, it will be clearly stated which reflect FFTPACK "enhanced" code and which do not. Of course, we can expect similar improvements in performance, on all platforms,

The semi-naive algorithm was chosen as the standard against which we would measure the performance of the various **DH**-based algorithms. Previous work [17] has shown the semi-naive algorithm to be both stable and faster than the direct algorithm.

When conducting timing tests, to average out timing variations due to multiprocessing and discretization, we would execute the algorithm one thousand times. Unless stated otherwise, it is the CPU time (in seconds) of the thousand iterations we report. Furthermore, we assume all precomputation and storage of the cosine transforms of the shifted Legendre polynomials prior to timing. To measure the error of the various algorithms, we employ the following procedure:

1. Select a bandwidth B and order m.
2. Generate a set of random Legendre coefficients $\hat{f}_m^m, \hat{f}_{m+1}^m, \ldots, \hat{f}_{B-1}^m$, normally distributed with mean 0 and standard deviation 1, using the *Mathematica*© package *NormalDistribution.m*.
3. Synthesize the function
$$f(\cos\theta_k) = \sum_{l=m}^{B-1} \hat{f}_l^m \tilde{P}_l^m(\cos\theta_k)$$
where $k = 0, \ldots, 2B-1$ and $\theta_k = \frac{\pi(2k+1)}{4B}$.
4. Apply the algorithm to this synthesized function, generating a new set of Legendre coefficients $\hat{g}_m^m, \hat{g}_{m+1}^m, \ldots, \hat{g}_{B-1}^m$.
5. Compute the error as
$$\max_l \left\| \hat{f}_l^m - \hat{g}_l^m \right\| .$$
6. Repeat steps 2-5 ten times.

7. Compute the average and relative error over the ten trials.

The synthesis step (Step 3) will necessarily introduce some error which we have empirically determined to be on the order of 10^{-9} in both the mean and standard deviation when bandwidth $B = 1024$.

In the matter of the simple-split algorithm, we would always choose the number of splits which minimized runtime *and* average error. This would be determined by trial and error. The number of splits used would not necessarily minimize the total number of operations (as discussed in Section 5). A general lesson we learned is that theoretical, optimal algorithms do not always yield the most cache-friendly algorithms. Inefficient cache-usage can render "fast" algorithms slow.

With regards to the hybrid algorithm the **switch point** of the algorithm, i. e., the degree sw of the coefficient at which we switched from the semi-naive to simple split algorithm was determined (after much experimentation) by the formula

$$sw = \begin{cases} m + \frac{B-m}{2} & m < \frac{B}{16} \\ m + \frac{3(B-m)}{4} & \text{otherwise} \end{cases}$$

where m is the order of the transform and $B \leq 1024$ is the bandwidth of the problem. Originally, the above formula was used on all platforms and at all bandwidths B. However, we discovered that at problem size $B = 1024$ we could increase the simple-split portion of the hybrid algorithm on the HP, SGI and Linux machines (but *not* the DEC) and get a bigger "win" over the pure semi-naive algorithm. Therefore, we used the following separate formula when running bandwidth $B = 1024$ on those non-DEC Alpha platforms:

$$sw = m + \frac{1024 - m}{2} .$$

Choosing this sw seemed to minimize the runtime. Of course, your mileage may vary.

We first give results for order $m = 0$ transforms. In Figure 8 we plot the ratios of the running times of the various algorithms versus the semi-naive algorithm on the four platforms. The runtimes themselves are plotted in Figure 9. All but the Linux results in Figures 8 and 9 reflect use of FFTPACK.

The dependence of an algorithm's performance on architecture is quite apparent. On the DEC, the simple-split and hybrid algorithms gain an advantage over the semi-naive algorithm between bandwidths $B = 256$ and $B = 512$. On the HP and SGI, the simple-split and hybrid algorithms are competitive with the semi-naive algorithm starting at approximately $B = 512$. And on the Linux, it's a definite win for the hybrid and simple-split algorithms at $B = 256$. A transform of size $B = 1024$ is necessary for the basic algorithm to be competitive with the semi-naive algorithm on all platforms. Table 2.1 gives the average and relative errors of the algorithms on the DEC Alpha. These errors are representative of what was measured on all platforms.

Figure 8 represents remarkable evidence of how computer architecture can effect performance. Keep in mind that identical C code was compiled on all machines. The C code was not written to take advantage of any particular architectural features. In fact, the C code was developed on the DEC.

Next, in Figures 10 and 11 we give the timing results for order $m = \frac{B}{2}$ transforms. As before, all but the Linux results in Figures 10 and 11 reflect use of FFTPACK. Errors are given in Table 2.2.

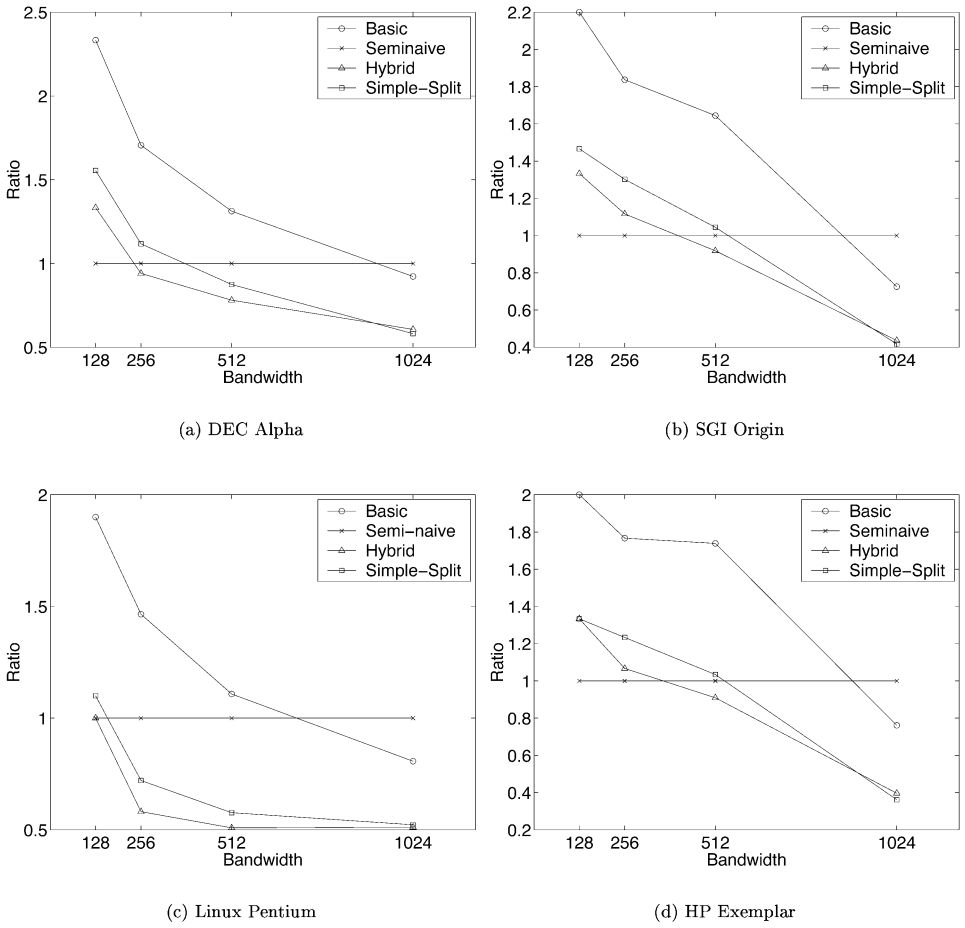
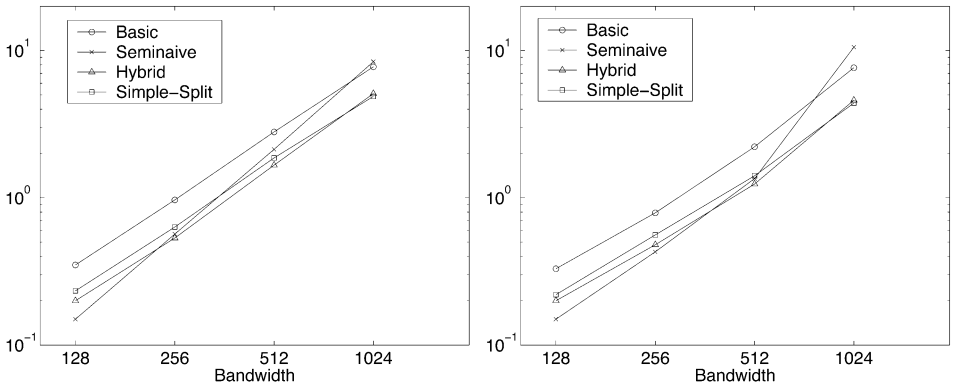


FIGURE 8 DLT runtime ratios vs. Semi-naive, Order $m = 0$.

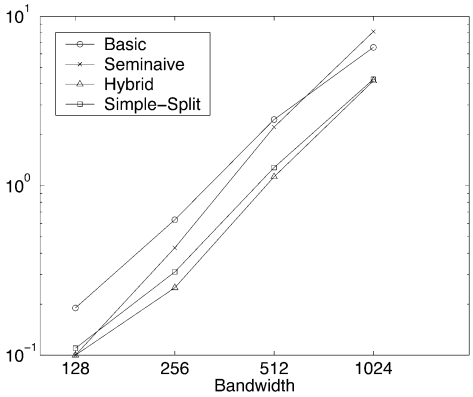
When doing transforms of order $m = B/2$, the basic algorithm was not tested because we found it to be numerically unstable at high orders. The shifted Legendre polynomials proved to be the source of the instabilities. To illustrate, suppose the bandwidth is $B = 1024$ and the order is $m = 512$. Using the basic algorithm, the first recursive splitting involves multiplying the weighted signal by the shifted Legendre polynomial $\mathbf{A}_{256}^{512}$. As Figure 12 makes clear, multiplying by $\mathbf{A}_{256}^{512}$ would effectively zero out the sample values in the middle and stretch those near the end points to absurd values. For moderate and larger problem sizes, beginning at approximately order $m = 4$, too great a shift will introduce instabilities.

This is why we do not test the basic **DH** and **DH-Mid** algorithms. Their "divide-and-conquer" structure is not only expensive computationally (as opposed to theoretically optimal), but it also requires use of shifted Legendre polynomials. At present, the means to achieve a stable **DH** and **DH-Mid** algorithm still eludes us. Of course, one could use "stability bypass operations," a technique first developed in [50], or the philosophically related stabilization procedure proposed in [55], to overcome either algorithm's instabilities. However, these techniques degrade somewhat the runtime performance and make them less competitive than the semi-naive algorithm, at least at the bandwidths we investigated. So

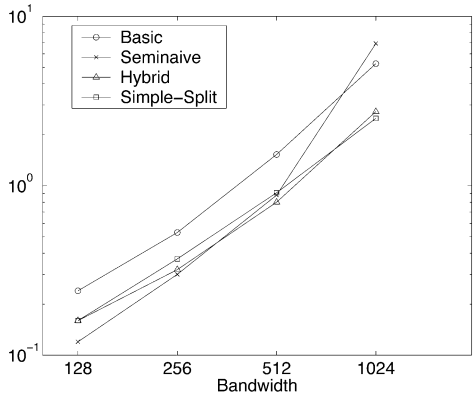


(a) DEC Alpha

(b) SGI Origin



(c) Linux Pentium



(d) HP Exemplar

FIGURE 9 DLT runtimes (in CPU seconds) for 1000 iterations, Order $m = 0$.

the moral we take from this lesson is this: to use shifted Legendre polynomials, take care not to shift too far with them.

The simple split algorithm can still be used with reasonable accuracy. Provided that enough initial splits are made, the numerically unsound shifted Legendre polynomials can be avoided. However, while more initial splits imply shifts of smaller distance, they do come with the penalty of increasing execution time. Recall that more splits require more discrete cosine transforms, and we have found these to be expensive. This can be seen in the Figure 10. At order $m = \frac{B}{2}$, the simple-split algorithm takes significantly longer to run than the semi-naive algorithm. This difference comes completely from the cost of doing cosine transforms. Profiling the code on the DEC at bandwidth $B = 1024$ reveals that the semi-naive algorithm runs in roughly 45% of the time the simple-split algorithm spends simply doing cosine transforms. To achieve a numerically sound answer with the simple-split algorithm at $B = 1024$, $m = 512$, 32 splits must be performed.

The hybrid algorithm, however, does remains competitive with semi-naive at order $m = \frac{B}{2}$, both in terms of runtime and accuracy, on the SGI and HP platforms at $B = 1024$, and at $B = 512$ on the Linux Pentium. Even though the hybrid algorithm uses shifted Legendre polynomials, it uses only relatively "nice" shifted Legendre polynomials. The

TABLE 2.1

Order $m = 0$ DLT Average (First Row) and Relative (Second Row) Errors on a DEC Alpha Workstation

Bandwidth	Semi-naive	Basic	Simple split	Hybrid
128	5.9138e-11	5.9146e-11 (1)	5.9130e-11 (1)	5.9120e-11 (1)
	4.4843e-09	4.4860e-09	4.4802e-09	4.4794e-09
256	2.0897e-10	2.0884e-10 (1)	2.0887e-10 (1)	2.0885e-10 (1)
	3.6380e-08	3.6388e-08	3.6230e-08	3.6128e-08
512	5.8493e-10	5.8475e-10 (1)	5.8496e-10 (2)	5.8499e-10 (2)
	3.6618e-07	3.6625e-07	3.6670e-07	3.6665e-07
1024	3.0777e-09	3.0760e-09 (2)	3.0764e-09 (3)	3.0771e-09 (3)
	6.4634e-06	6.4683e-06	6.4645e-06	6.4639e-06

For the basic algorithm, the number in parentheses refers to how many levels of recursive splitting were performed. Likewise, for the simple split and hybrid algorithms, the number in parentheses refers to how many splits were done.

TABLE 2.2

Order $m = B/2$ DLT Average (First Row) and Relative (Second Row) Errors on an SGI Origin Workstation

Bandwidth	Semi-naive	Simple split	Hybrid
128	7.4787e-11	4.5475e-09 (3)	7.4795e-11 (1)
	3.9475e-09	1.3638e-07	3.9471e-09
256	2.4867e-10	8.2479e-10 (8)	2.4866e-10 (1)
	2.3683e-08	2.2254e-08	2.3683e-08
512	1.1740e-09	5.5436e-09 (16)	1.1755e-09 (2)
	8.9257e-08	9.9039e-08	8.9252e-08
1024	4.6725e-09	1.9154e-07 (32)	4.6725e-09 (7)
	9.9888e-07	1.4598e-06	1.0024e-06

For the simple split and hybrid algorithms, the number in parentheses refers to how many splits were done.

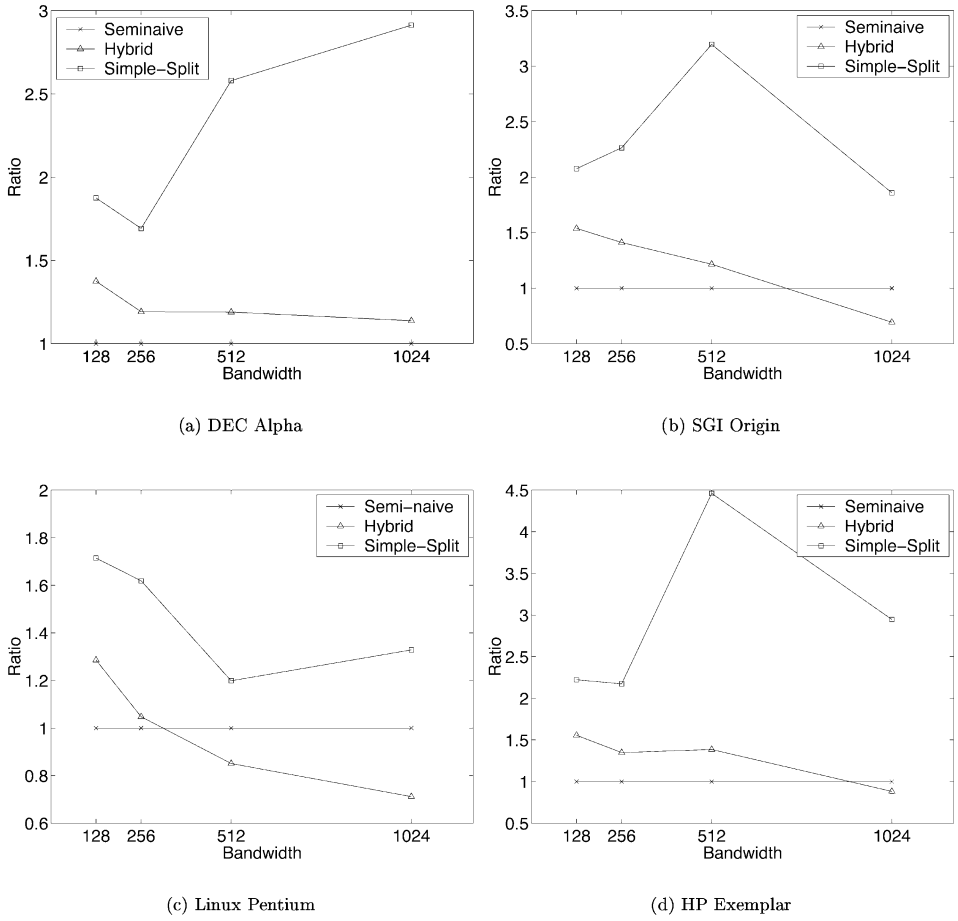
rate at which $\max_{x\in[-1,1]} \left|\mathbf{A}_r^L(x)\right| \to \infty$ as $r \to \infty$ is slower for large values of L than for small. This also holds true for $\mathbf{B}_r^L(x)$. For example, assuming bandwidth $B = 1024$ and order $m = 512$, we have the following maximum values:

$$\max_{x\in[-1,1]} \left|\mathbf{A}_{10}^{768}(x)\right| \approx 3612$$

and

$$\max_{x\in[-1,1]} \left|\mathbf{A}_{10}^{512}(x)\right| \approx 6 \times 10^{11} .$$

This is the case because the initial conditions for the recurrence relations which generate the shifted Legendre polynomials grow smaller the further L is from m. By applying the semi-naive technique to compute the lower degree coefficients, the hybrid algorithm avoids precisely those shifted Legendre polynomials which contribute most to error.



(a) DEC Alpha

(b) SGI Origin

(c) Linux Pentium

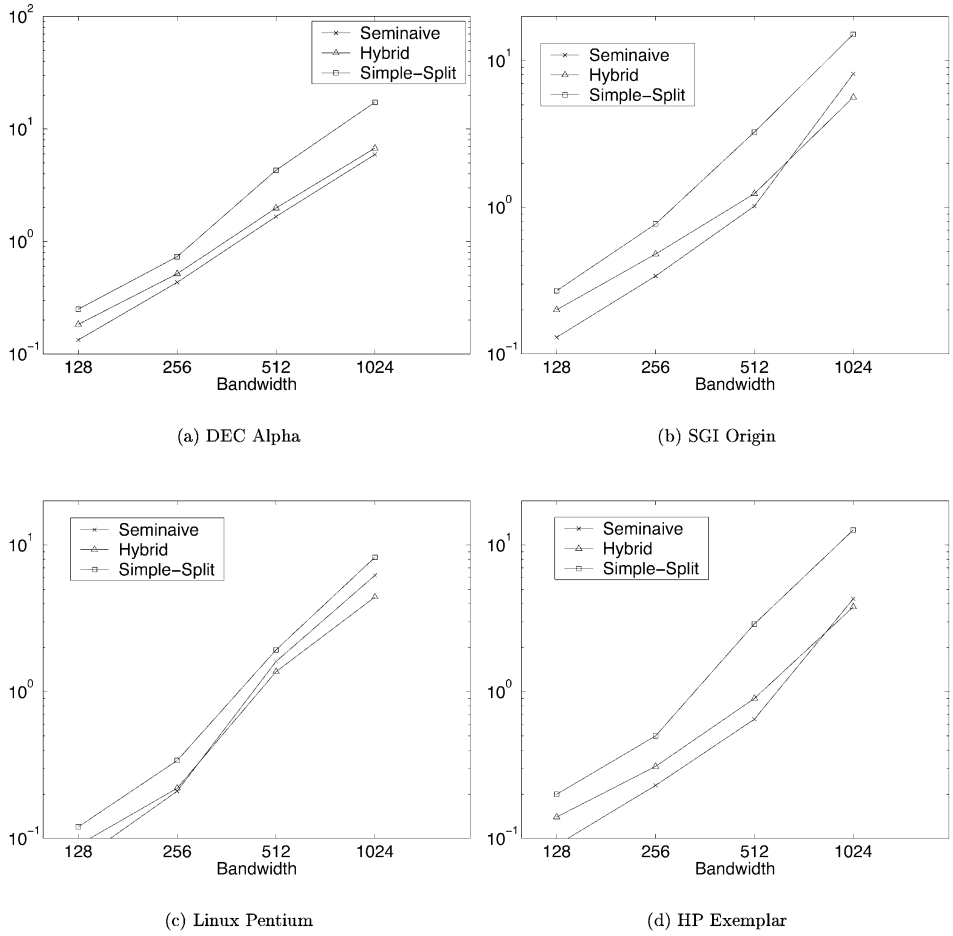
(d) HP Exemplar

FIGURE 10 DLT runtime ratios vs. Semi-naive, Order $m = B/2$.

Since, on the Linux, HP and SGI platforms, the hybrid algorithm is both faster than the semi-naive algorithm and stable for orders $m = 0$ through $m = 512$ at $B = 1024$, a significant savings can be achieved when performing a forward spherical transform. We implemented a hybrid spherical transform which works as follows. For orders $m = 0$ through $m = B/2$, the hybrid algorithm is used in performing the discrete Legendre transforms. For the remaining orders, the semi-naive algorithm is used.

We now present two sets of timing results. First, in Table 2.3 we give running times, on three different platforms, for the semi-naive and hybrid spherical transform algorithms, and the freely available software package SPHEREPACK [62]. The semi-naive and hybrid spherical transform algorithms in Table 2.3 reflect use of FFTPACK. We should note that, unlike all the algorithms discussed up to this point, SPHEREPACK assumes the function is sampled (in θ) on the Gaussian points (i. e., Legendre points) and not the Chebyshev points (i. e., $\cos\left(\frac{\pi(k+1/2)}{2B}\right)$). Hence, when doing a forward spherical transform of size B, SPHEREPACK expects the function to be sampled on a $2B \times B$ grid, while the semi-naive and hybrid spherical transform algorithms require the sampling to be on a $2B \times 2B$ grid.

As the bandwidth increases, the difference between the SPHEREPACK and other timings grows larger. At bandwidth $B = 512$, the semi-naive and hybrid algorithms are



(a) DEC Alpha

(b) SGI Origin

(c) Linux Pentium

(d) HP Exemplar

FIGURE 11 DLT runtimes (in CPU seconds) for 1000 iterations, Order $m = B/2$.

TABLE 2.3

Time (in CPU Seconds) for 10 Iterations of a Forward Spherical Transform on the DEC, HP and SGI; *: Walltime

		DEC Alpha		SGI Origin		HP Exemplar	
B	SPHEREPACK	Seminaive	Hybrid	Seminaive	Hybrid	Seminaive	Hybrid
128	8.33e−01	6.83e−01	7.67e−01	6.00e−01	6.60e−01	6.30e−01	6.40e−01
256	6.21e+00	4.25e+00	4.20e+00	3.67e+00	3.76e+00	3.52e+00	3.46e+00
512	4.77e+01	2.87e+01	2.79e+01	2.23e+01	2.25e+01	1.99e+01	1.94e+01
1024	NA	NA	NA	3.21e+02	2.41e+02	2.72e+02	1.77e+02
1024	NA	NA	NA	3.30e+02*	2.52e+02*	5.16e+02*	3.19e+02*

nearly 40% faster than SPHEREPACK. Although we were not able to test SPHEREPACK on the Linux, HP or SGI, given the results obtained on the DEC, we believe we could reasonably expect that SPHEREPACK would run slower than the semi-naive and hybrid algorithms on these platforms as well.

In Table 2.4, partially as a comparison with Table 2.3, we give results of running the semi-naive and hybrid spherical transform algorithms on the Linux and SGI, in this case *without* taking advantage of FFTPACK in either algorithm. When comparing Tables 2.3 and 2.4, the sensitivity of the SGI to the use (or not) of FFTPACK is quite pronounced.

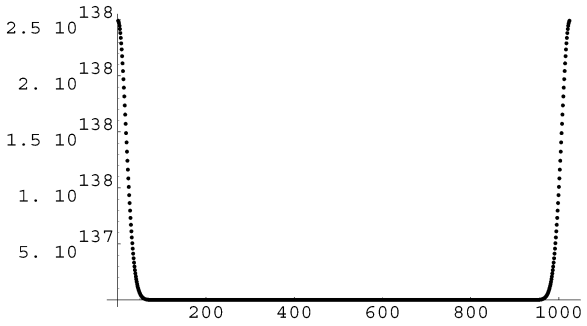


FIGURE 12 $\mathbf{A}_{256}^{512}$: sampled 1024 times in the interval $[-1, 1]$.

Errors are given in Table 2.5.

TABLE 2.4

Time (CPU Seconds) for 10 Iterations of a Forward Spherical Transform, Without the use of FFTPACK; *: Walltime

	Linux Pentium		SGI Origin	
bw	Seminaive	Hybrid	Seminaive	Hybrid
128	6.80e−01	7.00e−01	1.49e+00	1.67e+00
256	3.46e+00	3.51e+00	7.76e+00	9.38e+00
512	2.44e+01	2.30e+01	4.33e+01	5.91e+01
1024	2.36e+02	1.74e+02	5.85e+02	4.88e+02
1024	5.86e+02*	3.29e+02*	5.66e+02	5.43e+02*

Due to local hardware limitations (the source of which will be apparent in the following sentence), we did not run the $B = 1024$ forward spherical transform on the DEC. At $B = 1024$, the semi-naive spherical transform requires a precomputed data structure on the order of 1.3 gigabytes. The hybrid spherical transform requires under 0.9 gigabytes. Both these data structures are far too large to be entirely contained in memory. Therefore we had to read the precomputed data off disk, and this is why we also report the walltime. We believe that this is a legitimate quantity to mention because, in practice, we would expect *not to* precompute the data structures every time prior to performing a spherical transform.

For the record, the HP walltime given in Table 2.3 was obtained when the precomputed data was saved on a disk specially configured to allow 30% faster disk i/o than on a normally configured disk. Experiments confirmed this when the data was saved on such a "normal" disk. We note that the CPU time measured in both experiments was virtually identical, as to be expected.

We emphasize that although the asymptotic optimality of the basic algorithm is a theoretical result, it establishes the utility of shifted Legendre polynomials in developing fast algorithms.

TABLE 2.5

Errors, Over 10 Iterations, on the Linux Pentium for the Semi-Naive and Hybrid Forward Spherical Transforms, for Bandwidth $B = 512, 1024$

	$B = 512$		$B = 1024$	
	Seminaive	Hybrid	Seminaive	Hybrid
Average Error	2.2077e-09	2.1003e-09	7.8214e-09	1.8142e-08
Std Dev	1.4125e-10	1.0797e-10	6.5612e-10	2.2814e-09
Relative Error	1.7037e-07	1.2718e-07	6.9960e-06	1.5200e-06
Std Dev	1.0059e-07	5.1994e-08	1.2784e-05	1.1063e-06

7. Two Applications

As stated in Section 1, a fast Fourier transform for S^2, as well as a fast convolution algorithm have many ready-made applications in applied science (see e. g., [52, 38, 59, 12, 48, 29, 28, 43].) In this section we examine in a little more detail two of these, one for the efficient computation of the bispectrum, potentially of use for image processing insensitive to rotations, and the other to matched filtering on S^2.

7.1 Computation of the Bispectrum and Triple Correlation

The techniques of multiple correlations and higher order spectra have been developed for nonabelian Lie groups and their homogeneous spaces by R. Kakarala [37]. Of particular interest is the triple correlation and its associated Fourier transform, the bispectrum.

For functions on the line, the triple correlation is the integral of the product of the function with two independently shifted copies of itself. The resulting function on $\mathbf{R}^2$ determines the original function up to translation. The usefulness of computing the triple correlation derives from the fact that it is (1) insensitive to additive Gaussian noise; (2) retains most of the phase information of the underlying signal and (3) is invariant under translation of the underlying signal. This makes it useful in recovering a signal from multiple observations in situations in which the signal may be translating on a noisy background.

Kakarala has been able to generalize many of the results for functions on the line to arbitrary locally compact groups and their homogeneous spaces. For particular examples of interest such as the sphere, a suitably defined triple correlation of a band-limited function is again unique up to translation, (assuming that the Fourier coefficients are nonsingular) and insensitive to additive Gaussian noise. This suggests possible applications for global rotational motion compensation and Kakarala goes on to suggest possible applications to imaging the heart [12].

The techniques which we have developed for fast, reliable spherical convolution admit almost immediate application to fast, reliable computation of the triple correlation or bispectrum on the sphere. A detailed explanation of this is beyond the intended scope of this article. However, the following abbreviated discussion should give some indication of our ideas.

To get to the bispectrum on the sphere we must go through the bispectrum for functions on its cover $SO(3)$. If $f \in L^2(SO(3))$, then the triple correlation of f is the function on

$SO(3) \times SO(3)$ given by

$$a_{3,f}(s,t) = \int_{SO(3)} f(gs) f(gt) f(g)\, dg$$

where dg denotes Haar measure on $SO(3)$. Assuming f is integrable on $SO(3)$, then $a_{3,f}$ is integrable on $SO(3) \times SO(3)$.

The irreducible representations of $SO(3)$ are naturally indexed by nonnegative integers, one irreducible of dimension $2l+1$ for each $l \geq 0$, which we denote as ρ_l. Consequently, the irreducible representations of $SO(3) \times SO(3)$ are given by all possible tensor products $\rho_l \otimes \rho_{l'}$, so are indexed by all pairs $\{l, l'\}$ with $l \geq l' \geq 0$.

The Fourier transform of f at ρ_l, denoted as $\hat{f}(l)$ is the integral

$$\hat{f}(l) = \int_{SO(3)} f(g) \rho_l(g)^\dagger\, dg$$

where $\dagger$ indicates conjugate transpose. The Fourier transform of f is the collection $\{\hat{f}(l)\}_{l \geq 0}$.

Similarly, the Fourier transform of a function on $SO(3) \times SO(3)$ will be the analogously defined collection $\{\hat{f}(l,l')\}_{l' \geq l \geq 0}$. The bispectrum of f is Fourier transform of $a_{3,f}$. In [37] Kakarala shows how the bispectrum may be computed from the Fourier transform of $f \in L^2(SO(3))$. For this we need to introduce one more piece of notation. Notice that $SO(3)$ has a natural embedding in $SO(3) \times SO(3)$ as the diagonal subgroup. Considered as such, each representation $\rho_{l',l}$ when restricted to the diagonal will be equivalent to a direct sum of appropriate ρ_j. Thus, there exists an invertible matrix $C_{l,l'}$ such that

$$\rho_{l,l'}(s,s) = C_{l,l'} \left[\rho_{j_1(l,l')}(s) \oplus \rho_{j_2(l,l')}(s) \oplus \cdots \oplus \rho_{j_m(l,l')}(s) \right] C_{l,l'}^\dagger$$

for suitable indices $j_i(l,l')$.

Theorem ([37], Lemma 3.2.3).
With the notation as above,

$$\widehat{a}_{3,f}(l,l') = \hat{f}(l) \otimes \hat{f}(l')\, C_{l,l'} \left[\hat{f}\left(j_1(l,l')\right)^\dagger \oplus \hat{f}\left(j_2(l,l')\right)^\dagger \oplus \cdots \oplus \hat{f}\left(j_m(l,l')\right)^\dagger \right] C_{l,l'}^\dagger \quad (7.1)$$

When $f \in L^2(SO(3))$ comes from a function on the sphere, (i. e., $f \in L^2(SO(3))$ is right $SO(2)$-invariant) then the matrix $\hat{f}(l)$ will have entries all 0 except, possibly, for a single column which (up to a normalization constant) will contain the associated Legendre transforms

$$\left\{ \hat{f}(l,-l), \ldots, \hat{f}(l,0), \ldots, \hat{f}(l,l) \right\}.$$

Thus, if f is band-limited, then the bispectrum will only involve a finite number of Fourier transforms. For each l, l', $\widehat{a}_{3,f}(l,l')$ can then be computed directly as follows. Compute first the spherical harmonic expansion as described in Section 3. This precomputes all possible Fourier transforms $\hat{f}(l)$ for any $f \in L^2(S^2)$. The inner direct sum of the matrices

$$\left[\hat{f}\left(j_1(l,l')\right)^\dagger \oplus \hat{f}\left(j_2(l,l')\right)^\dagger \oplus \cdots \oplus \hat{f}\left(j_m(l,l')\right)^\dagger \right]$$

is then constructed by retrieving the appropriate associated Legendre transforms and organizing them together into a single sparse block diagonal matrix. This is then conjugated by the precomputed change of basis matrices $C_{l,l'}$. Finally the lefthand factor

$$\hat{f}(l) \otimes \hat{f}\left(l'\right)$$

simply requires the computation of all possible pointwise products $\hat{f}(l,m)\hat{f}(l',m')$ organized as the appropriate single nonzero column in some suitably defined matrix. These matrices are then all multiplied together, giving the relevant component of the bispectrum.

7.2 Matched Filters

One simple application of the techniques of this article may be found in certain problems of detection, estimation, and pattern matching for data defined on the sphere. This sort of data arises in geophysics, computer vision, or quality assurance for computer designed and manufactured parts.

A simple problem arising in this area may be stated as follows: Suppose we are considering a known signal or pattern in the directional data setting, described by a function, $f(\omega)$ on the sphere. In many situations, we are interested in determining the presence or absence of this signal in data coming from measurements of some real world phenomenon. This is often made more difficult by the presence of some random interference, or noise, in the measurements. In the simplest cases, we assume that one of two hypotheses obtains for the measured data, $\mathbf{y}(\omega)$:

- H_0: $\mathbf{y}(\omega) = \mathbf{n}(\omega)$
- H_1: $\mathbf{y}(\omega) = f(\omega) + \mathbf{n}(\omega)$,

where $\mathbf{n}(\omega)$ is a random process on the sphere representing the noise. Our task is then to devise an algorithm which takes a particular instance of the measured data and returns an assessment of whether or not the signal is present in the data.

A more interesting version of this problem occurs when, in addition to the additive noise, the pattern signal f may have undergone a rotation which is unknown to us. That is, when f is present, the measured data has the form

$$\mathbf{y}(\omega) = \Lambda(g) f(\omega) + \mathbf{n}(\omega) \ ,$$

where g is an unknown element of $SO(3)$, and $\Lambda(g)$ is the associated operator, $\Lambda(g) f(\omega) = f(g^{-1}\omega)$. In this case we have the more involved detection and estimation problem; determine if a rotated version of the pattern is present, and if so, estimate the value of the rotation parameter g. We will concern ourselves with this question.

The intuitive approach to this involves a template matching operation. That is, one correlates the data with the pattern one is looking for, which amounts to forming the inner product of the data with a large number of shifted versions of the pattern. Those shifts which produce a large inner product, or correlation, between the pattern and the data are regarded as good indicators that there really is a copy of the pattern shifted to the corresponding location and buried in the noise. This correlation process is known as matched filtering; it amounts to computing the function

$$\chi(g) = \int_{S^2} y(\omega)\Lambda(g) f(\omega)\, d\omega \ .$$

This matched filter is also indicated by a standard statistical analysis for these sorts of problems. This analysis yields optimal detection and estimation solutions involving a computation of the appropriate *likelihood function*, [39, 71, 69]. Basically, for a given measurement of data, the value of this function gives the likelihood of having made that

particular observation given a certain hypothesis (signal present or signal absent) or a given value of the unknown parameter.

For example, suppose we know that a rotated version of the signal is present, in the data process $\mathbf{y}(\omega)$, and we wish to know where it is. This is the same as estimating the rotation parameter g. We assume that the additive noise $\mathbf{n}(\omega)$ is Gaussian and white. The latter term refers to the covariance structure of the noise, implying first that the covariance $R(\omega_1, \omega_2) = \mathbf{E}[\mathbf{n}(\omega_1)\mathbf{n}(\omega_2)]$ is actually rotation independent, so that $R(\omega_1, \omega_2) = R(g\omega_1, g\omega_2)$ for any rotation $g \in SO(3)$. This property is sometimes referred to as "stationarity." A consequence of stationarity is that R is determined by the values $R(\omega) = R(\omega, \eta)$, for η the north pole of the sphere. White noise is a particular stationary noise with point mass covariance; $R(\omega) = \sigma^2\delta_\eta(\omega)$. Strictly speaking, this requires the usual sorts of mathematical temporizing required when dealing with distributions; we'll assume that is familiar.

The likelihood $L(g)$ of a particular value of the parameter g given the data y is the probability density for the random variable $\mathbf{y} = \Lambda(g)f + \mathbf{n}$ evaluated at the particular observed measurement values y_o; this is the same as $p_{\mathbf{y}}(y_o; g) = p_{\mathbf{n}}(y_o - \Lambda(g)f)$. Using our assumptions on $\mathbf{n}$ and some limiting arguments, we obtain for our likelihood:

$$\begin{aligned} L(g) &\propto e^{-\frac{||y_o - \Lambda(g)f||_2^2}{2\sigma^2}} \\ &\propto e^{\frac{<y_o, \Lambda(g)f>}{\sigma^2}}, \end{aligned}$$

as the other terms which come from expanding the norm, $e^{-\frac{||y_o||_2^2}{2\sigma^2}}$ and $e^{-\frac{||\Lambda(g)f||_2^2}{2\sigma^2}} = e^{-\frac{||f||_2^2}{2\sigma^2}}$ are constant, independent of g. Thus the maximum likelihood estimate of g is

$$\text{Arg Max}_{g\in SO(3)} \int_{S^2} y(\omega)\Lambda(g)f\omega\, d\omega\,.$$

Let us consider now a simple case in which the pattern signal is rotationally symmetric. In fact, we take f to be the analog of the normal density on the sphere, the Fisher–von Mises density $C_\kappa \exp(\kappa\cos\theta)$. Here, κ is a concentration parameter, C_κ a normalizing factor. In this case, the matched filter expression actually reduces to a function defined on the sphere, rather than the entire group, due to rotation invariance. Below we show the results of some experiments in which f is rotated and buried in white noise, and then passed through a matched filter. The results are shown in Figure 13.

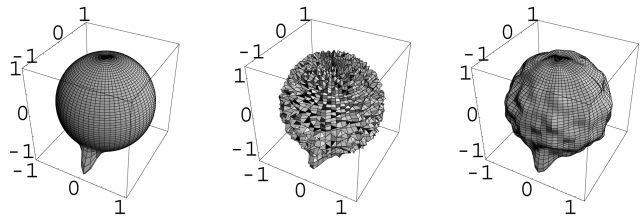


FIGURE 13 (a) The pattern signal $f(\omega)$ is the Fisher–von Mises density $C_\kappa \exp(\kappa\cos\theta)$, with concentration parameter $\kappa = 64$, and rotated by an arbitrary rotation g on the sphere; (b) $\Lambda(g)f(\omega)$ has been buried in additive white noise $n(\omega)$ to simulate noisy measured data $y(\omega)$; (c) The likelihood function $L(g)$ computed by using the fast convolution algorithm to convolve $y(\omega)$ with a matched filter. The position $g_{\max}$ of the maximum value of $L(g)$ indicates the maximum likelihood estimate of the position of the pattern signal $f(\omega)$.

8. Summary and Future Directions

We have presented a divide-and-conquer algorithm for the efficient and exact computation of the forward and inverse Fourier transform of a band-limited function on the 2-sphere, which in addition provides a fast algorithm to compute the exact convolution of two such functions. We give evidence that by fine-tuning different variations of the basic algorithm, highly efficient and numerically reliable implementations can be obtained. These algorithms have a wide range of applicability in a great range of scientific disciplines.

We view this work as another step in the still nascent development of algorithms and applications for efficient nonabelian harmonic analysis. Given the small ratio of nonabelian to abelian articles, it seems that there are still many potentially fruitful directions to pursue (see also [47, 36]).

1. **Fine-tuning.** It is clear that there are many parameters to vary over a range of "simple-split-like" algorithms. The forthcoming article [32] explores some of these variations and gives strong indications that highly stable and efficient implementations of this recurrence-based exact approach are feasible. Beyond that it is also clear that the execution time is architecture dependent. It would be of interest to pursue in this setting the sort of architecture-optimization analysis shown to be so effective in the abelian FFT case [4].

2. **Vector and tensor harmonic expansions.** For various applications in meteorology, it is also important to compute the expansions of vector and tensor fields in terms of vector and tensor harmonics. With an appropriate definition, this may be reduced to the computation of several individual Fourier transforms on the 2-sphere, so that our algorithms may be applied [42].

3. **Parallelizabilty.** The divide-and-conquer nature of the basic algorithm indicates that efficient parallel implementations may be possible. The technical report [30] was a first step in this direction. Recently, Inda et al. develop and discuss an efficient parallel implementation of the basic algorithm based on polynomial arithmetic [36].

4. **Improved complexity.** Our algorithms only use one of the recurrences satisfied by the $\mathbf{P}_\ell^m$. Perhaps through the use of other recurrences the overall complexity can be reduced to $O(n \log n)$.

5. **Other compact groups and their quotients.** Due to its applicability, we have concentrated our efforts on developing algorithms for the 2-sphere. The basic ideas shown here work (in theory) for any compact group and its quotients (cf. [44, 45, 47]). Recent work explores implementations for the full orthogonal group as well as higher orthogonal groups [41]. Identification of new applications in this setting would probably dictate the priorities of related software development. The articles [33, 31, 58] give some indication of the wide variety of applications being found for these generalized FFTs.

6. **Noncompact groups.** G. Chirikjian and his collaborators (esp. A. Kyatkin) have been pursuing an active program developing FFTs for noncompact nonabelian groups, especially the Euclidean motion groups in two and three dimensions. Algorithms and implementations have been developed for use in areas such as motion planning for robotics, pattern recognition. This is an exciting new direction of work with many opportunities for both theoretical and practical development. See the book [13] for both a development of the theory as well as an encyclopedic collection of references.

7. **Quantum analogues.** Our algorithm is effectively a sparse structured unitary matrix

factorization. As such, it could permit an efficient "quantum" implementation (see e. g., [35] for descriptions of quantum FFTs for finite noncommutative groups). It would be of interest to pursue this.

Acknowledgment

We have benefited greatly from close contact with the National Center for Atmospheric Research (NCAR). We thank NCAR's Scientific Computing Division for their hospitality on several occasions and access to their HP Exemplar, a machine named Sioux. In particular, we thank Mark Taylor for helping to educate us in the ways of scientific computing for climate modeling and helping us to run our comparisons against SPHEREPACK. Thanks also to Paul Swarztrauber for helpful discussions and encouragement.

References

[1] Ahmed, N., Natarajan, T., and Rao, K.R. (1974). Discrete cosine transforms, *IEEE Transactions on Computers*, **23**, 90–93.

[2] Aho, A., Hopcroft, J., and Ullman, J. (1976). *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading.

[3] Alpert, B. and Rokhlin, V. (1991). A fast algorithm for the evaluation of Legendre transforms, *SIAM J. Sci. Statist. Comput.*, **12**, 158–179.

[4] Auslander, L., Johnson, J.R., and Johnson, R.W. (1996). Multidimensional Cooley–Tukey algorithms revisited, *Adv. in Appl. Math.*, **17**(4), 477–519.

[5] Auslander, L. and Tolmieri, R. (1979). Is computing with the fast Fourier transform pure or applied mathematics?, *Bulletin of the American Mathematical Society*, (N. S.), **1**, 847–897.

[6] Barrucand, P. and Dickinson, D. (1968). On the associated Legendre polynomials, in *Orthogonal Expansions and their Continuous Analogues*, Southern Illinois University Press, Carbondale.

[7] Biedenharn, L.C. and Louck, J.D. (1981). *Angular Momentum in Quantum Mechanics*, Addison-Wesley, Reading.

[8] Borodin, A. and Munro, I. (1975). *The Computational Complexity of Algebraic and Numeric Problems*, Elsevier, New York.

[9] Boyd, J. (1989). *Chebyshev and Fourier Spectral Methods*, Lecture Notes in Engineering, Vol. 49, Springer-Verlag, NY.

[10] Bshouty, N., Kaminski, M., and Kirkpatrick, D. (1988). Addition requirements for matrix and transposed matrix products, *J. of Algorithms*, **9**, 354–364.

[11] Calvetti, D. (1991). A stochastic roundoff error analysis for the fast Fourier transform, *Math. Comput.*, **56**(194), 755–774.

[12] Chen, C.W. and Huang, T.S. (1990). Epicardial motion and deformation estimation from coronary artery bifurcation points, in *Proc. of Third. Int. Conf. on Comp. Vision, Dec. 4-7, 1990*, IEEE Press, 456–460.

[13] Chirikjian, G.S. and Kyatkin, A.B. (2000). *Engineering Applications of Noncommutative Harmonic Analysis: With Emphasis on Rotation and Motion Groups*, CRC Press.

[14] Cooley, J.W. and Tukei, J.W. (1965). An algorithm for machine calculation of complex Fourier series, *Math. Comput.*, **19**, 297–301.

[15] Cormen, T.H., Leiserson, C.E., and Rivest, R.L. (1990). *Introduction to Algorithms*, MIT Press.

[16] Diaconis, P. (1980). Average running time of the fast Fourier transform, *J. Algorithms*, **1**, 187–208.

[17] Dilts, G.A. (1985). Computation of spherical harmonic expansion coefficients via FFTs, *J. of Computational Physics*, **57**(3), 439–453.

[18] Driscoll, J.R. and Healy, D. (1989/1994). Computing Fourier transforms and convolutions on the 2-sphere, (extended abstract) in *Proc. 34th IEEE FOCS*, 344–349; *Adv. in Appl. Math.*, **15**, 202–250.

[19] Driscoll, J.R., Healy, D., and Rockmore, D. (1997). Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs, *SIAM J. Comput.*, **26**(4), 1066–1099.

[20] Duncan, B.S. and Olson, A.J. (1993). Approximation and characterization of molecular surfaces, *Biopolymers*, **33**, 219–229.

[21] Elliot, D.F. and Rao, K.R. (1982). *Fast Transforms: Algorithms, Analyses, and Applications*, Academic Press, New York.

[22] FFTPACK is a freely available collection of FORTRAN programs for computing one-dimensional fast Fourier transforms developed at NCAR by Paul Swarztrauber.

[23] FFTW is a C subroutine library for computing the Discrete Fourier Transform (DFT) in one or more dimensions, of both real and complex data, and of arbitrary input size. FFTW is a freely available at `http://www.fftw.org`.

[24] Freeden, W. (1980). On integral formulas of the (unit) sphere and their application to numerical computation of integrals, *Computing*, **25**, 131–146.

[25] Gallagher, N.C., Wise, G.L., and Allen, J.W. (1978). A novel approach for the computation of Legendre polynomial expansions, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-**26**(1), 105–106.

[26] Gilbert, F.A. (1971). Inverse problems for the earth's normal modes, in *Mathematical Problems in the Geophysical Sciences*, Vol. I, American Mathematical Society, Providence.

[27] Greengard, L. (1994). Fast algorithms for classical physics, *Science*, **265**, 909–914.

[28] Healy, D. and Kim, P. (1996). An empirical Bayes approach to directional data and efficient computation on the sphere, *Ann. Stat.*, **24**(1), 232–254.

[29] Healy, D., Hendriks, H., and Kim, P. (1993). Spherical deconvolution with application to geometric quality assurance, Technical Report, Department of Mathematics and Computer Science, Dartmouth College.

[30] Healy, D., Moore, S., and Rockmore, D. (1994). Efficiency and stability issues in the numerical convolution of Fourier transforms and convolutions on the 2-sphere, Technical Report PCS-TR94-222, Department of Computer Science, Dartmouth College.

[31] Healy, D., Olson, T., Rockmore, D., and Mirchandani, G. Wreath products for image processing, in *Proceedings of 1996 ICASSP*, Vol. 6, 3582–3585.

[32] Healy, D., Rockmore, D. and Kostelec, P. Stabilized algebraic FFTs on the 2-sphere, preprint.

[33] Healy, D., Rockmore, D., and Moore, S. An FFT for the 2-sphere and applications, in *Proceedings of 1996 ICASSP*, Vol. 3, 1323–1326.

[34] Heideman, M.T., Johnson, D.H., and Burrus, C.S. (1985). Gauss and the history of the fast Fourier transform, *Arch. for History of Exact Sciences*, **34**(3), 265–277.

[35] Høyer, P. (1997). Efficient quantum transforms, Los Alamos preprint quant-ph/9702028, February.

[36] Inda, M.A., Bisseling, R.H., and Maslen, D.K. (2001). On the efficient parallel computation of Legendre transforms, *SIAM J. on Scientific Computing*, **23**(1), 271–303.

[37] Kakarala, R. (1992). Triple correlation on groups, Ph.D. thesis, Dept. of Math., University of California, Irivine.

[38] Kanatani, K. (1990). *Group-Theoretical Methods in Image Understanding*, Springer-Verlag, NY.

[39] Kay, S. (1993). *Fundamentals of Statistical Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ.

[40] Kobayashi, K. (1985). Solution of multi-dimensional neutron transport equation of the spherical harmonics method using the finite Fourier transformation and quadrature formula, *Transport Theory and Stat. Phys.*, **14**, 63–85.

[41] Kostelec, P., Rockmore, D., Maslen, D., and Healy Jr., D.M. FFTs on the rotation group, in preparation.

[42] Kostelec, P., Maslen, D., Rockmore, D., and Healy Jr., D.M. (2000). Computational harmonic analysis for tensor fields on the two-sphere, *J. Comp. Physics*, **162**, 514–535.

[43] Martyna, G.J. and Berne, B.J. (1989). Structure and energies of Xe_n^-: Many body polarization effects, *J. Chem. Phys.*, **90**(7), 3744–3755.

[44] Maslen, D. (1993). Fast transforms and sampling for compact roups, Ph.D. thesis, Department of Mathematics, Harvard University.

[45] Maslen, D. (1998). Efficient computation of Fourier transforms on compact groups, *J. Fourier Anal. Appl.*, **4**(1), 19–52.

[46] Maslen, D. and Rockmore, D. (2001). The Cooley–Tukey FFT and group theory, *Notices of the AMS*, **48**(10), 1151–1160.

[47] Maslen, D. and Rockmore, D. (1997). Generalized FFTs, in *Groups and Computation II*, DIMACS Series in Discrete Math. and Computer Science, Vol. 28, Finkelstein, L. and Kantor, W., Eds., 183–237.

[48] Miller, M., Joshi, S., Maffitt, D., McNally, J., and Grenander, U. (1994). Membranes, mitochondria and amoebae: shape models, in *Advances in Applied Statistics*, 137–159.

[49] Mohlenkamp, M.P. (1999). A fast transform for spherical harmonics, *J. Fourier Anal. Appl.*, **5**(2-3), 159–184.

[50] Moore, S. (1994). Efficient stabilization methods for fast polynomial transforms, Ph.D. thesis, Department of Mathematics and Computer Science, Dartmouth College.

[51] Moore, S., Healy, D., and Rockmore, D. (1993). Symmetry stabilization for polynomial evaluation and interpolation, *Lin. Alg. Appl.*, **192**, 249–299.

[52] Peng Oh, S., Spergel, D.N., and Hinshaw, G. (1998). An efficient technique to determine the power spectrum from cosmic microwave background maps, preprint.

[53] Oppenheim, A. and Schafer, R. (1989). *Discrete-Time Signal Processing*, Prentice-Hall, NJ.

[54] Orzag, S.A. (1986). Fast eigenfunction transforms, in *Science and Computers*, Academic Press, Orlando.

[55] Potts, D., Steidl, G., and Tasche, M. (1998). Fast and stable algorithms for discrete spherical Fourier transforms, *Linear Algebra Appl.*, **275/276**, 433–450.

[56] Potts, D., Steidl, G., and Tasche, M. (1998). Fast algorithms for discrete polynomial transforms, *Math. Comp.*, **67**, 1577–1590.

[57] Ramos, G.U. (1971). Roundoff error analysis of the fast Fourier transform, *Math. Comp.*, **25**, 757–768.

[58] Rockmore, D. (1997). Some applications of generalized FFTs, (Appendix with D. Healy), in *Groups and Computation II*, DIMACS Series in Discrete Math and Computer Science, Vol. 28, Finkelstein, L. and Kantor, W., Eds., 329–369.

[59] Schwartz, J.T. (1985). Mathematics addresses problems in computer vision for advanced robotics, *SIAM News*, **18**(3).

[60] Swarztrauber, P. (1993). The vector harmonic transform method for solving partial differential equations in spherical geometry, *Monthly Weather Review*, **121**(12), 3415–3437.

[61] 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, it is available at `www.cs.dartmouth.edu/~geelong/sphere/`.

[62] SPHEREPACK is a freely available collection of FORTRAN programs that facilitates computer modeling of geophysical processes developed at NCAR by John Adams and Paul Swarztrauber.

[63] Spotz, W.F. and Swarztrauber, P.N. (2001). A performance comparison of associated Legendre projections, *J. Comp. Physics*, **168**(2), 339–355.

[64] Steidl, G. and Tasche, M. (1991). A polynomial approach to fast algorithms for discrete Fourier-cosine and Fourier-sine transforms, *Math. Comp.*, **56**, 281–296.

[65] Swarztrauber, P.N. and Spotz, W.F. (2000). Generalized discrete spherical harmonic transforms, *J. Comp. Physics*, **159**(2), 213–230.

[66] Temperton, C. (1991). On scalar and vector transform methods for global spectral models, *Mon. Wea. Rev.*, **119**, 1303–1307.

[67] Vaidyanathan, P.P. (1993). *Multirate Systems and Filter Banks*, Prentice-Hall, Englewood Cliffs, NJ.

[68] Van Loan, C. (1992). Computational framework for the fast Fourier transform, *SIAM*, Philadelphia.

[69] Van Trees, H. (1968). *Detection, Estimation and Modulation Theory*, Vol. I, Wiley, New York.

[70] Vilenkin, N.J. (1968). *Special Functions and the Theory of Group Representations*, American Mathematical Society, Providence.

[71] Woodward, P.M. (1980). *Probability and Information Theory, with Applications to Radar*, Artech House, Dedham, MA.

Received October 18, 2001

Revision received February 04, 2002

Mathematics Department, University of Maryland, College Park, MD 20742-4015

Department of Mathematics, Dartmouth College, Hanover, NH 03755
e-mail: rockmore@tahoe.cs.dartmouth.edu

Department of Mathematics, Dartmouth College, Hanover, NH 03755

Cetacean Networks, Inc, 100 Arboretum Drive, Suite 301, Pease International Tradeport,
Portsmouth, NH 03801-6815