

# Fast Spherical Harmonic Transform Algorithm based on Generalized Fast Multipole Method

Reiji Suda  
the University of Tokyo / CREST, JST

## 1 Introduction

Spherical harmonic transform is the most important orthogonal function transform only except Fourier transform, and is used not only for climate simulation and signal processing but also for a base of several numerical algorithms. Fast Fourier Transform (FFT), which runs in time  $O(N \log N)$  is quite well known, but, for spherical harmonic transform, there is no fast algorithm which is as simple as FFT.

We have proposed a fast transform algorithm for spherical harmonic transform using Fast Multipole Method (FMM)[17, 25], and its program is publicly available as FLTSS[24, 20]. Our algorithm computes the transform using polynomial interpolations, and to make it possible, we have introduced *split Legendre functions*[25], which nearly double the computational costs, and also affect the numerical stability[21].

To reduce computational costs, we tried fast transform without using split Legendre functions[22, 23] with the help of generalized FMM[29, 28]. This algorithm (we call this *new algorithm* and the former *old algorithm*) runs faster than the old algorithm with improved numerical stability.

This paper discusses some details of the new algorithm and its implementation.

## 2 Spherical Harmonic Transform

### 2.1 Spherical harmonic transform

A spherical harmonic function  $Y_n^m(\lambda, \mu)$  can be decomposed into an associated Legendre function and a trigonometric function as

$$Y_n^m(\lambda, \mu) = P_n^m(\mu)e^{im\lambda}.$$

Thus evaluation of a spherical harmonic expansion

$$g(\lambda, \mu) = \sum_{m=0}^T \sum_{n=m}^T g_n^m Y_n^m(\lambda, \mu)$$

can also be decomposed into associated Legendre function transform and Fourier transform as

$$g^m(\mu) = \sum_{n=m}^T g_n^m P_n^m(\mu), \quad (1)$$

$$g(\lambda, \mu) = \sum_{m=0}^T g^m(\mu)e^{im\lambda}. \quad (2)$$

The inverse spherical harmonic transform is a computation of  $g(\lambda_j, \mu_k)$  from  $g_n^m$ , while the spherical harmonic transform is a computation of  $g_n^m$  from  $g(\lambda_j, \mu_k)$ , where  $1 \leq j \leq J$ ,  $1 \leq k \leq K$ , and  $J$  and  $K$  are usually determined by the so-called alias-free condition

$$J \geq 3T + 1, \quad K \geq (3T + 1)/2. \quad (3)$$

For asymptotic complexity analysis, we can simply assume that  $J = O(T)$  and  $K = O(T)$ . The inverse transform consists of the associated Legendre function transform and Fourier transform, and the Fourier transform (2) can be computed in time  $O(T^2 \log T)$  by using FFT for each  $\mu_k$ . Conventional method of the associated Legendre function transform is the direct summation of the form of (1), thus requires time  $O(T^3)$ . Let us call this method *direct computation*. By using direct computation for associated Legendre function transform, the spherical harmonic transform requires time  $O(T^3)$ . Thus, by accelerating associated Legendre function transform, we can reduce the computational complexity of spherical harmonic transform up to  $O(T^2 \log T)$ , which is the complexity of FFT. It is well known that the forward transform algorithm is easily derived from the inverse transform algorithm. So we focus on the inverse associated Legendre function transform (1) in the following discussion.

## 2.2 Related works

Some related research works are reviewed.

Driscoll and Healy[6] proposed a fast spherical harmonic transform using FFT. Their algorithm runs in time  $O(T^2 \log^2 T)$ , and is precise (under infinite precision arithmetic operations). Their algorithm is quite well known, but unfortunately it is numerically unstable. Several modified versions[7, 14] were proposed to remedy the instability, but they increase computational costs. The research is still continuing[10, 9] but a tight complexity bound for the stabilized algorithm is not found yet.

Mohlenkamp[11] proposed two stable and fast algorithms for the spherical harmonic transform using fast wavelet transform. His algorithms run in time  $O(T^{5/2} \log T)$  and  $O(T^2 \log^2 T)$ , but the former runs faster in practical sizes.

Rokhlin and Tygert[15] proposed another fast spherical harmonic transform algorithm by using FMM. The complexity of their algorithm is  $O(T^2 \log T)$ , but their implementation in that paper is not faster than the direct computations for  $T \leq 4096$ , so it requires a better implementation.

The following works are of the Legendre polynomial transform, which is a part of spherical harmonic transform (only for  $m = 0$ ). To clarify the difference from the spherical harmonic transform, we will use  $N$  instead of  $T$ .

Orszag[13] invented an algorithm of Legendre polynomial transform that uses the WKB approximation and FFT. Orszag's algorithm runs in time  $O(N \log^2 N / \log \log N)$ . We have improved his algorithm using the asymptotic expansion by Stieltjes[16], with the resulting computational complexity of  $O(N \log N)$ [12].

Alpert and Rokhlin[1] proposed a fast algorithm for Legendre polynomial transform and runs in time  $O(N \log N)$ . They provide a pair of transforms between the coefficients of a Legendre polynomial expansion and those of a Chebyshev expansion that represents the same function. Accelerating those transforms as the same manner as the FMM, they can show their complexity as  $O(N)$ . Then by using FFT, one can compute the Legendre polynomial transforms (forward and backward) in time  $O(N \log N)$ .

Beylkin, Coifman and Rokhlin showed that the transform between a Legendre polynomial expansion and the corresponding Chebyshev expansion can be computed in time  $O(N)$  by using fast wavelet transform. Thus their algorithm also runs in time  $O(N \log N)$ .

The followings are not of spherical harmonic transforms, but our old algorithm owes them the key idea.

Boyd[3] showed that a function represented by an associated Legendre function expansion can be interpolated in linear time with a help of FMM.

Jakob-Chien and Alpert[8] proposed a fast algorithm for the spherical filtering, which removes the higher frequency components than a given cut-off frequency, in time  $O(T^2 \log T)$ , also using FMM.

### 3 The new fast transform algorithm

#### 3.1 Fast interpolation of associated Legendre function expansions

Legendre function can be represented with ultraspherical polynomial  $q_{n-m}^m(\mu)$  as

$$P_n^m(\mu) = c_n^m P_m^m(\mu) q_{n-m}^m(\mu), \quad (4)$$

where  $c_n^m$  is a constant.  $P_n^m(\mu)/P_m^m(\mu)$  is a polynomial of degree  $n-m$  and thus  $g^m(\mu)/P_m^m(\mu)$  (where  $g^m(\mu)$  is defined in (1)) is also a polynomial of degree  $T-m$ . Now we have Lagrange interpolation formula for polynomial

$$p(\mu) = \omega(\mu) \sum_{i=1}^D \frac{1}{\mu - \mu_i} \frac{p(\mu_i)}{\omega_i(\mu_i)}$$

with  $D = \deg(p) + 1$  and

$$\omega(\mu) = \prod_{i=1}^D (\mu - \mu_i), \quad \omega_i(\mu) = \frac{\omega(\mu)}{\mu - \mu_i}.$$

Thus letting  $M = T - m + 1$  we can evaluate  $g^m(\mu_k)$  on a set of arbitrary points  $\{\mu_k\}$  from  $M$  sampling points  $\{g^m(\mu_i)\}_{i=1}^M$  as

$$g^m(\mu_k) = P_m^m(\mu_k) \omega(\mu_k) \sum_{i=1}^M \frac{1}{\mu_k - \mu_i} \frac{g^m(\mu_i)}{P_m^m(\mu_i) \omega_i(\mu_i)}. \quad (5)$$

The computation (5) can be computed in three steps.

1. Point-wise multiplication  $\bar{g}_i = g^m(\mu_i)/P_m^m(\mu_i)\omega_i(\mu_i)$  for  $1 \leq i \leq M$ .
2. Summation  $\bar{g}_k = \sum_{i=1}^M \bar{g}_i/(\mu_k - \mu_i)$  for  $1 \leq k \leq K$ .
3. Point-wise multiplication  $g^m(\mu_k) = \bar{g}_k P_m^m(\mu_k) \omega(\mu_k)$  for  $1 \leq k \leq K$ .

If we have fixed the sets  $\{\mu_i\}$  and  $\{\mu_k\}$  and precompute  $P_m^m(\mu_i)\omega_i(\mu_i)$  and  $P_m^m(\mu_k)\omega(\mu_k)$ , then steps 1 and 3 are computed in time  $O(M)$  and  $O(K)$ . By applying FMM[5], we can compute step 2 in time  $O(M + K)$ , as is pointed out by Boyd[3]. Thus the interpolation (5) can be computed in time linear to the input size (i.e.,  $O(M + K)$ ).

#### 3.2 Base scheme of fast associated Legendre function transform

Using the fast interpolation of associated Legendre function expansion discussed in the previous section, we can accelerate the associated Legendre function transform in the following way.

1. Compute values on the sampling points  $g^m(\mu_i) = \sum_{n=m}^T g_n^m P_n^m(\mu_i)$  for  $1 \leq i \leq M$ .
2. Compute values on the target points  $g^m(\mu_k)$  for  $1 \leq k \leq K$  by interpolation from  $g^m(\mu_i)$ .

The computational complexity of step 1 is clearly  $O(M^2)$ . By choosing the sampling points  $\{\mu_i\}_{i=1}^M$  from the target points  $\{\mu_k\}_{k=1}^K$  and using fast interpolation, the computational complexity of step 2 is  $O(K)$ . Remembering  $M = T - m + 1$  and the alias-free condition (3), one can see that the asymptotic computational costs of the above scheme is 4/9 of the direct computation.

#### 3.3 The divide-and-conquer scheme

To reduce the computational costs furthermore, we apply the above scheme to step 1 recursively. Let us restate step 1:

$$g^m(\mu_i) = \sum_{n=m}^T g_n^m P_n^m(\mu_i) \quad \text{for } 1 \leq i \leq M.$$

We divide the summation into two roughly equal parts.

$$g^m(\mu_i) = g_0^m(\mu_i) + g_1^m(\mu_i), \quad (6)$$

$$g_0^m(\mu_i) = \sum_{n=m}^{n_0-1} g_n^m P_n^m(\mu_i), \quad (7)$$

$$g_1^m(\mu_i) = \sum_{n=n_0}^T g_n^m P_n^m(\mu_i). \quad (8)$$

where  $n_0$  is a nearest integer of  $m + (T - m)/2$ .

Now the fast interpolation scheme is applicable. To accelerate computation of (7), we choose an appropriate set of  $n_0 - m$  points  $\mathcal{M}_0 = \{\mu_{i_0}\}$  from the set of  $M$  points  $\mathcal{M} = \{\mu_i\}$ , compute values  $g_0^m(\mu_{i_0})$  for  $\mu_{i_0} \in \mathcal{M}_0$ , and interpolate those values onto the other points  $\mu_i \in \mathcal{M}_0 - \mathcal{M}$ . This is accomplished exactly as stated in the previous section. The computational complexity is  $O((n_0 - m)^2 + M)$ .

We want to do the same thing for (8). Let us choose  $T - n_0 + 1$  points  $\mathcal{M}_1 = \{\mu_{i_1}\}$  from  $\mathcal{M}$ . (Note that  $\mathcal{M}_0$  and  $\mathcal{M}_1$  can be different sets of points.) Because the degree of freedom of  $g_1^m(\mu)$  is  $T - n_0 + 1$ , the values  $g_1^m(\mu_i)$  on the remaining points  $\mu_i \in \mathcal{M} - \mathcal{M}_1$  can be determined from those on  $\mathcal{M}_1$ . However,  $g_1^m(\mu)/P_m^m(\mu)$  is a polynomial of degree  $M = T - m + 1$ , not  $T - n_0 + 1$ . For this reason, we cannot use the polynomial-based interpolation (5).

In the old algorithm[17, 25], we solved this problem by introducing split Legendre functions:

$$\begin{aligned} P_n^m(\mu) &= P_{n,\nu}^{m,0}(\mu) + P_{n,\nu}^{m,1}(\mu), \\ P_{n,\nu}^{m,l}(\mu) &= q_{n,\nu}^{m,l}(\mu) P_{\nu+l}^m(\mu) \quad l = 0, 1. \end{aligned}$$

By splitting the expansion  $g_1^m(\mu)$  accordingly

$$\begin{aligned} g_1^m(\mu) &= g_{1,\nu}^{m,0}(\mu) + g_{1,\nu}^{m,1}(\mu) \\ g_{1,\nu}^{m,l}(\mu) &= \sum_{n=n_0}^T g_n^m P_{n,\nu}^{m,l}(\mu) \quad l = 0, 1, \end{aligned}$$

$g_{1,\nu}^{m,l}(\mu)/P_{\nu+l}^m(\mu)$  becomes a polynomial of degree  $|T - \nu + l - 1|$ , and thus interpolation scheme (5) becomes applicable. This method requires two interpolations for  $l = 0$  and 1, thus the computational costs are doubled. Also we have additional computational costs to change the split point  $\nu$ , and the numerical stability is affected[21].

In the new algorithm[22], we use generalized FMM[29, 28]. Let  $P_1$  be an  $M \times (T - n_0 + 1)$  matrix whose  $(i, j)$  element is  $P_{n_0+j}^m(\mu_i)$ , and  $g_1$  be a column vector  $(g_{n_0}^m, g_{n_0+1}^m, \dots, g_T^m)^T$ . By letting  $v_1$  as a column vector with  $g_1^m(\mu_i)$  as its elements, we have matrix-vector representation for (8) as

$$v_1 = P_1 g_1.$$

Let us consider the set of sampling points  $\mathcal{M}_1$  which has  $T - n_0 + 1$  points. Collect the set of rows from  $P_1$  and  $v_1$  that correspond to  $\mathcal{M}_1$  and represent them as  $\bar{P}_1$  and  $\bar{v}_1$ . Then we have

$$\bar{v}_1 = \bar{P}_1 g_1.$$

Also collect the remaining rows and represent them as  $\hat{P}_1$  and  $\hat{v}_1$ , then we have

$$\hat{v}_1 = \hat{P}_1 g_1.$$

Since each element of  $v_1$  is either in  $\bar{v}_1$  or in  $\hat{v}_1$ , we can obtain  $v_1$  by aggregating  $\bar{v}_1$  and  $\hat{v}_1$  by using an appropriate permutation matrix  $\Pi_1$  as

$$v_1 = \Pi_1 \begin{pmatrix} \bar{v}_1 \\ \hat{v}_1 \end{pmatrix} = \Pi_1 \begin{pmatrix} \bar{P}_1 \\ \hat{P}_1 \end{pmatrix} g_1.$$

Note that  $\bar{P}_1$  is a square matrix, and assume that it is non-singular. Then we have

$$\begin{aligned} v_1 &= \Pi_1 \begin{pmatrix} I \\ Q_1 \end{pmatrix} \bar{P}_1 g_1, \\ Q_1 &= \hat{P}_1 \bar{P}_1^{-1}. \end{aligned} \quad (9)$$

The matrix  $Q_1$  represents the interpolations of the elements of  $\hat{v}_1$  from  $\bar{v}_1$ . We apply generalized FMM (details described in [28]) to  $Q_1$ . Then the computation  $Q_1 \bar{v}_1$  is accelerated.

For the problems to which the analytical FMM is applicable, generalized FMM works no worse than the analytical FMM. Thus if  $Q_1$  represents a polynomial interpolation, then the complexity of computing  $\hat{v}_1 = Q_1 \bar{v}_1$  is linear to the sum (instead of the product) of the numbers of the rows and the columns of  $Q_1$ . But  $Q_1$  is surely not a polynomial interpolation. So we lose the complexity bound. However, we can avoid the split Legendre functions that doubled the computational costs in the old algorithm. Thus we may have some gain. Let us evaluate it experimentally in section 4.

Before closing this subsection, we evaluate the computational complexity *on the assumption that generalized FMM runs in linear time*. We will discuss about the preprocessing phase somewhat later, and here let us consider only transform computations assuming that appropriate preprocessing has been done. Let us compute the computational costs in a backward manner. The last step of the computation is the polynomial interpolation (5). As is stated in the previous section, its complexity is  $O(K)$ . The second last step is the summation (6). The complexity is clearly  $O(M)$ . The third last step is the interpolation  $\hat{v}_1 = Q_1 \bar{v}_1$  and the corresponding interpolation of the other half. From the assumption, their computational costs are summed up as  $O(M)$ . The step before that is summations similar to (6). They are two summations of half sizes, thus their costs are summed up as  $O(M)$ . Then we have four interpolations of half sizes, which also amounts  $O(M)$ . After  $O(\log M)$  steps of recursion, we have  $O(M)$  problems of a constant size. Summing up them, we have the computational complexity  $O(K + M \log M)$ . This complexity is for each  $m$ . Note that  $M = T - m + 1$  and  $K = O(T)$ . Summing up for all  $m$ , the computational complexity is  $O(T^2 \log T)$ .

### 3.4 Error control and sampling point selection

As stated in the previous subsection, the interpolation matrix  $Q_1$  in (9) is accelerated by using generalized FMM. Because the FMM is an approximate algorithm, we have to consider the errors at the output ( $v_1$  in (9)) incurred by the approximation. The following methodology of error analysis and control is introduced in [26, 27] and used also for generalized FMM[28].

Before discussion, note that (9) does not reach the end of the computation. The resulting  $v_1$  are interpolated into the remaining points by the interpolation (5). Let us represent the interpolation (5) by a matrix  $Q$ , and consider the computation

$$w_1 = Q \Pi_1 \begin{pmatrix} I \\ Q_1 \end{pmatrix} \bar{P}_1 g_1.$$

Let  $\tilde{Q}_1$  be the matrix representation of the approximate interpolation accelerated by generalized FMM. Then the actual computation is

$$\tilde{w}_1 = Q \Pi_1 \begin{pmatrix} I \\ \tilde{Q}_1 \end{pmatrix} \bar{P}_1 g_1.$$

Thus the error is

$$\tilde{w}_1 - w_1 = Q \Pi_1 \begin{pmatrix} 0 \\ \tilde{Q}_1 - Q_1 \end{pmatrix} \bar{P}_1 g_1.$$

Let us ignore 0 above  $\tilde{Q}_1 - Q_1$ , and we want to control

$$e_1 = Q \hat{\Pi}_1 (\tilde{Q}_1 - Q_1) \bar{P}_1 g_1,$$

where  $\hat{\Pi}_1$  represents the part of  $\Pi_1$  that corresponds to  $Q_1$ .

Let  $\|\cdot\|$  represent some norm both for vectors and matrices that satisfies the consistency inequality:

$$\|AB\| \leq \|A\|\|B\|, \quad \|Av\| \leq \|A\|\|v\| \quad (10)$$

for any matrices  $A$  and  $B$  and any vector  $v$  (but of course the number of columns of  $A$  and the number of rows of  $B$  and  $v$  should be the same).

Then we have

$$\|e_1\| \leq \|Q\hat{\Pi}_1\|\|\tilde{Q}_1 - Q_1\|\|\bar{P}_1\|\|g_1\|. \quad (11)$$

Note that if we use double precision, then we cannot expect approximation of higher precision than the round-off error  $\varepsilon \approx 10^{-16}$ :

$$\frac{\|\tilde{Q}_1 - Q_1\|}{\|Q_1\|} \geq \varepsilon.$$

Thus the right-hand side of (11) has a lower bound

$$\|Q\hat{\Pi}_1\|\|\tilde{Q}_1 - Q_1\|\|\bar{P}_1\| \geq \|Q\hat{\Pi}_1\|\|Q_1\|\|\bar{P}_1\|\varepsilon,$$

and so  $\kappa = \|Q\hat{\Pi}_1\|\|Q_1\|\|\bar{P}_1\|$  works as a kind of condition number.

However, in many cases the above *condition number*  $\kappa$  can be very large even when the output error  $\|e_1\|$  or  $\|\tilde{v}_1 - v_1\|$  is quite reasonable. This happens when the consistency inequality (10) is very loose. Then the evaluation (11) does not give a good estimate.

A simple solution is proposed in [26, 27]. Let us introduce diagonal matrices  $S_A$  and  $S_B$  and insert them as:

$$e_1 = (Q\hat{\Pi}_1 S_A^{-1})(S_A(\tilde{Q}_1 - Q_1)S_B)(S_B^{-1}\bar{P}_1)g_1$$

so that we have

$$\|e_1\| \leq \|Q\hat{\Pi}_1 S_A^{-1}\|\|S_A(\tilde{Q}_1 - Q_1)S_B\|\|S_B^{-1}\bar{P}_1\|\|g_1\|.$$

By letting  $Z_1 = S_A Q_1 S_B$  and  $\tilde{Z}_1 = S_A \tilde{Q}_1 S_B$  we have

$$\|e_1\| \leq \|Q\hat{\Pi}_1 S_A^{-1}\|\|\tilde{Z}_1 - Z_1\|\|S_B^{-1}\bar{P}_1\|\|g_1\|.$$

Now the ‘‘condition number’’ becomes

$$\kappa_S = \|Q\hat{\Pi}_1 S_A^{-1}\|\|Z_1\|\|S_B^{-1}\bar{P}_1\|,$$

which can be controlled by appropriately choosing  $S_A$  and  $S_B$ . Our previous works [26, 27, 28] proposed to choose  $S_A$  and  $S_B$  so that the columns of  $Q\hat{\Pi}_1 S_A^{-1}$  and the rows of  $S_B^{-1}\bar{P}_1$  have the unit norm. This method is quite well known as a simple device to improve the condition numbers of matrices  $Q\hat{\Pi}_1$  and  $\bar{P}_1$ . The effects of  $S_A$  and  $S_B$  depend on the matrices, and we have no theoretical guarantee of their goodness. We only know it works very well in the experiments.

Now if we want to limit the ‘‘relative error’’

$$\frac{\|e_1\|}{\|g_1\|} \leq \delta,$$

then it is enough to approximate  $Z_1 = S_A Q_1 S_B$  as

$$\frac{\|\tilde{Z}_1 - Z_1\|}{\|Z_1\|} \leq \frac{\delta}{\kappa_S}.$$

Practically it is observed that this requirement is still too pessimistic. So in our implementation the required precision for  $\tilde{Z}_1$  is mitigated as follows.

Before discussing it, note that we can bound some norms by using the proposed  $S_A$  and  $S_B$ . Let us use 2-norm which is bounded by the Frobenius norm as  $\|A\|_2 \leq \|A\|_F$ . Remember that the Frobenius norm of a matrix is the positive square root of the sum of the 2-norms of the rows or the columns of the matrix. Reviewing the previous subsection, we can count the number of rows and columns of the matrices. We have

$$\|Q\hat{\Pi}_1 S_A^{-1}\| \leq \sqrt{n_0 - m}, \quad \|S_B^{-1}\bar{P}_1\| \leq \sqrt{T - n_0 + 1}.$$

Thus  $\kappa_S \leq \sqrt{(n_0 - m)(T - n_0 + 1)} \|Z_1\|$ .

In our implementation  $\kappa_S$  is replaced by  $\kappa_0 \|Z_1\|$  with a constant  $\kappa_0$ . We start computation (of preprocessing) with  $\kappa_0 = 1$ , and if the resulting error is too large, then we reduce  $\kappa_0$  accordingly. After a few iterations, the resulting error meets the requirement  $\delta$ . If the required precision  $\delta$  is too small, then  $\delta/(\kappa_0 \|Z_1\|)$  may be smaller than the round-off error level. Then our program terminates with a message that the precision  $\delta$  cannot be achieved.

Note that the attainable precision  $\delta$  is determined mostly by  $\|Z_1\| = \|S_A Q_1 S_B\|$ . Smaller  $\|Z_1\|$  is preferable. Remember that  $Z_1 = S_A Q_1 S_B$  and  $Q_1 = \hat{P}_1 \bar{P}_1^{-1}$ .  $\bar{P}_1$  and  $\hat{P}_1$  comes from one common matrix  $P_1$ , and  $\bar{P}_1$  corresponds to the sampling points of the interpolation and  $\hat{P}_1$  corresponds to the other points. Thus we know that  $\|Z_1\|$  depends on the selection of the sampling points. Since the number of choices of the set of the sampling points is finite, we can minimize  $\|Z_1\|$  by trying all possibilities, but it is computationally impractical. So we do not minimize  $\|Z_1\|$  directly. What we do is just an LQ (transposed QR) decomposition with pivoting for  $S_A P_1$ :

$$S_A P_1 = \begin{pmatrix} \bar{L}_1 \\ \hat{L}_1 \end{pmatrix} U_1 = \begin{pmatrix} I \\ \hat{L}_1 \bar{L}_1^{-1} \end{pmatrix} \bar{L}_1 U_1.$$

(Note that we use  $U$  instead of the usual  $Q$ .) The pivoted LQ decomposition approximately minimizes  $\hat{L}_1 \bar{L}_1^{-1}$ , but it does not completely equal to  $Z_1$ , rather

$$Z_1 = \hat{L}_1 \bar{L}_1^{-1} S_A S_B.$$

We don't consider  $S_A S_B$ , so it's suboptimal. Also we ignore the effects of the recursion. But it works well, as is reported in the next section.

Last, let us discuss about the numerical stability. The Driscoll-Healy algorithm [6], which is numerically unstable, also uses exactly the same divide-and-conquer scheme as our algorithm. However, our algorithm is numerically stable, as the next section reports. The numerical instability of the Driscoll-Healy algorithm comes from the use of Fourier-Chebyshev expansion using ultraspherical polynomial (4). It is successful for low values for  $m$ , but for higher  $m$ , we meet difficulties at representing the associated Legendre functions using trigonometric functions. Our algorithm uses the same relation (4) to derive the fast interpolation (5), but it has enough degrees of freedom to attain numerical stability, that is, choice of the sampling points. The use of Fourier-Chebyshev expansion in the Driscoll-Healy algorithm effectively limits the sampling points to be the Chebyshev knots, and that makes  $\bar{L}_1^{-1}$  large for large  $m$ . In our new algorithm it is effectively solved by the pivoted LQ decomposition. In the old algorithm we have another method of sampling point selection for numerical stability [18].

As is discussed in [21], the split Legendre functions in our old algorithm considerably increase numerical instability. Our new algorithm removes the split Legendre functions, so it may improve the numerical stability compared to the old algorithm. The experimental results in the next section confirm this.

### 3.5 Preprocessing algorithm

Our algorithm uses divide-and-conquer strategy. It recursively divide the problem into two sub-problems. At some level of recursion, the problem becomes so small that the direct computation is faster. At that point the recursion should be stopped. To minimize the computational cost, our implementation uses the branch-and-bound technique in the preprocessing phase. Figure 1 shows a pseudo-code of our preprocessing algorithm. It is almost same as that given in [27].

The function `fast_pp`( $m, n_0, n_1, \mathcal{M}, c$ ) computes the preprocessing of fast transform on the set of the evaluation points  $\mathcal{M}$  for specified  $m$  and  $n_0 \leq n < n_1$ . Parameter  $c$  is the limit of the computational costs for the prescribed transform, and the return value of `fast_pp` is the obtained computational costs. Function `dir_pp` takes similar parameters and returns the computational costs of the direct computation.

This algorithm compares three methods: (1) compute the prescribed transform by the direct computation without interpolation, (2) compute the values on the set of sampling points (hereafter

```

1  function fast_pp( $m, n_0, n_1, \mathcal{M}, c$ )
2       $c_D = \text{dir\_pp}(m, n_0, n_1, \mathcal{M})$ 
3       $c_n = \min\{c, c_D\}$  // new cost limit
4      compute interpolation matrix and its FMM approximation
5      let  $\mathcal{X} \subset \mathcal{M}$  be the set of the sampling points
6      let  $c_i$  be the computational costs of the fast interpolation
7      if ( $c_i > c_n$ )
8          prepare direct computation without interpolation (method 1)
9          return  $c_D$ 
10     else // try use of interpolation
11          $c_d = \text{dir\_pp}(m, n_0, n_1, \mathcal{X})$ 
12          $c_t = \min\{c_n - c_i, c_d\}$  // new cost limit
13          $\nu = (n_0 + n_1)/2$  // divide the problem into two halves
14          $c_0 = \text{fast\_pp}(m, n_0, \nu, \mathcal{X}, c_t)$ 
15          $c_1 = \text{fast\_pp}(m, \nu, n_1, \mathcal{X}, c_t - c_0)$ 
16         if ( $c_0 + c_1 < c_d$  and  $c_i + c_0 + c_1 < c_n$ )
17             prepare interpolation with divide-and-conquer (method 3)
18             return  $c_i + c_0 + c_1$ 
19         else if ( $c_i + c_d < c_n$ )
20             prepare interpolation with direct computation (method 2)
21             return  $c_i + c_d$ 
22         else
23             prepare direct computation without interpolation (method 1)
24             return  $c_D$ 

```

Figure 1: Pseudo-code of preprocessing algorithm

$\mathcal{X}$ ) by the direct computation and then interpolate them on  $\mathcal{M}$ , and (3) compute the values on  $\mathcal{X}$  using divide-and-conquer and then interpolate them on  $\mathcal{M}$ .

At lines 4 to 6 it generates the interpolation matrix and its FMM approximation, and evaluates its computational costs  $c_i$ . If the costs of interpolation  $c_i$  is larger than  $c$  (given limit as parameter) or  $c_D$  (costs of direct computation), then we must abandon methods (2) and (3) that use interpolation. This check is done at lines 7 to 9. Otherwise, we will try methods (2) and (3). At line 11 the costs of the direct method for computing values on  $\mathcal{X}$  is evaluated as  $c_d$ . Lines 14 and 15 call `fast_pp` recursively, with appropriate cost limits. The costs of the method (2) are now given as  $c_i + c_d$ , and the costs of the method (3) are  $c_i + c_0 + c_1$ . Comparing them with the costs of the direct method without interpolation  $c_D$ , the method of the lowest costs is chosen at lines 16 to 24. The function does not abort even if the obtained computational costs are over the given limit  $c$ , because the result must be discarded in the caller.

Let us consider the computational complexity of the preprocessing algorithm given in Figure 1. Let the number of points  $M = |\mathcal{M}|$  and  $N = n_1 - n_0 + 1$ . Note that usually we have  $M \geq N$ . Using Suda-Kuriyama algorithm[28] for generalized FMM, the preprocessing costs for the FMM in line 4 is  $O(MN)$ . The computation of the interpolation matrix in line 4 uses pivoted LQ decomposition, which requires computations of  $O(MN^2)$ , which is the major costs at this recursion level. So the computational costs are 1/8 of this level for each of the halves at the next recursion level. Therefore the computational costs of the lower recursion levels are less than half of those at this level. Thus the computational costs at the root of the recursion tree determine the complexity order, which is  $O(K(T - m + 1)^2)$ . Summing up for every  $m$ , and assuming that  $K = O(T)$ , the total computational costs for the preprocessing is computed as  $O(T^4)$ . Those heavy costs solely come from the LQ decomposition to obtain the interpolation matrix  $Q_1$ . If we had any method to compute the interpolation matrix faster, then the computational complexity of the preprocessing phase would be reduced.

## 4 Evaluation of the new algorithm

This section reports the performance of the new algorithm. The following data are taken from [23].

### 4.1 Evaluation in spherical harmonic transform

Table 1: Comparison of speedup rates of spherical harmonic transform by old and new algorithms

$T$	old speedup	new speedup	improvements
255	1.46	1.46	1.00
511	1.76	1.78	1.01
1023	2.14	2.32	1.08
2047	2.76	3.17	1.15

Table 1 compares the speedup rates of our old and new algorithms. The required precision is set to  $10^{-10}$ . Here *speedup* is computed with the number of floating point number operations relative to that of the direct computation, and it is not of the CPU times.

Table 1 reports considerable speedup even for  $T = 255$ . This comes mostly from the dropping[27], and the effects of the FMM are small in this size. For larger sizes, the new algorithm performs slightly better than the old one.

### 4.2 Evaluation in Legendre polynomial transform

Next the new algorithm is examined further by limiting to  $m = 0$ , that is, to the Legendre polynomial transform. The computational costs are the higher for the smaller  $m$ , and also the FMM has the more effects on the performance. Also the recursion is the deepest for  $m = 0$ . The number of points  $M$  is now set as  $M = T + 1$ .

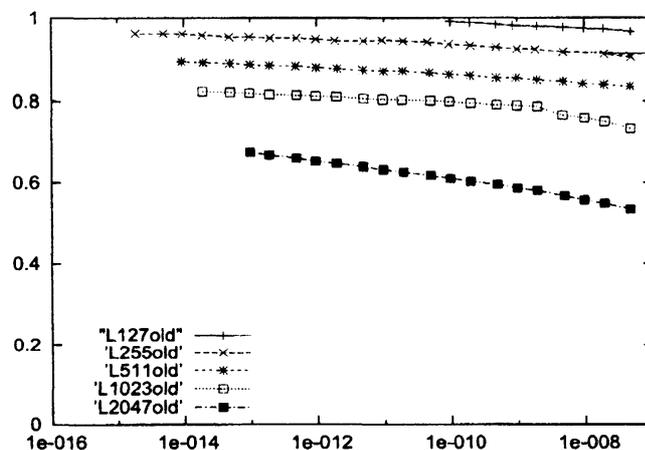


Figure 2: Relative computational costs of Legendre polynomial transforms by the old algorithm

Figures 2 and 3 show the floating point operation counts relative to the direct computation. The  $x$ -axis shows the attained precision. “L127old” and “L127new” give the performances for  $T = 127$ , etc.

Comparing those figures, we can see that  $T = 127$  gives small difference between the old and the new algorithms, but for  $T = 2047$  the old algorithm takes about 1.5 times more computations than the new algorithm.

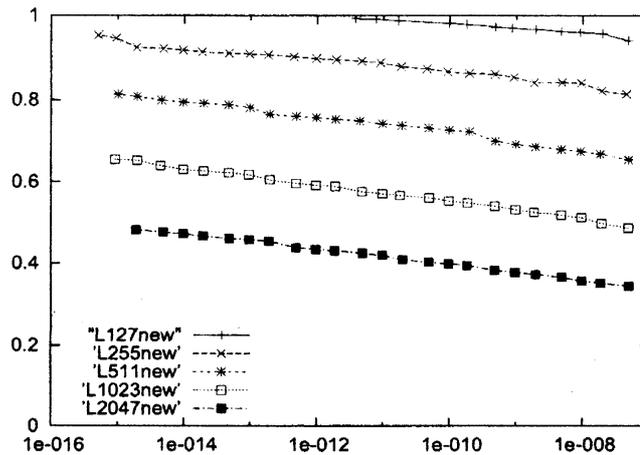


Figure 3: Relative computational costs of Legendre polynomial transforms by the new algorithm

For the old algorithm, the gradient is different for  $T = 1023$  with lower precision and for  $T = 2047$  than the other cases. This is because the split Legendre functions are used only for those conditions. However for the new algorithm, we found no such phase change.

The plots for the old algorithm stop at precision of  $10^{-15}$  to  $10^{-11}$ . This is because higher precision was not attained because of the numerical stability. For the new algorithm higher precisions were attained. Those results imply that the numerical stability is much improved in the new algorithm.

Further discussion on the performance will be found in [23].

## 5 Summary and future works

This paper discusses our fast algorithm for associated Legendre function transforms with generalized FMM. This algorithm is formerly reported in our previous reports [22, 23], but the details of the algorithm were not presented. In this paper, the transform algorithm and the preprocessing algorithm are discussed. The performance is reported in English for the first time.

There are several remaining works to do.

The new algorithm performs better than the old algorithm, but its computational complexity is not bounded theoretically. To bound it, the complexity bound for the interpolation with generalized FMM must be obtained, and to that purpose, we must have some analytical expression for the interpolation matrix entries.

Any specific property of the associated Legendre functions is utilized in the new algorithm. The new algorithm just compresses the transform matrix. The only exception is that the forward transform can be obtained by transposition of the inverse transform matrix, but this assumption can be removed if we compute the forward transform matrix directly. Thus the new algorithm can be applied to *any transform* at least conceptually.

Last, the implementation should be improved and released to the public use. Before doing that, we should improve the implementation of generalized FMM, because its current performance is somewhat lower than the previous one (but with dramatically reduced preprocessing time).

## Acknowledgments

This research is partly supported by Grant-in-Aid for Scientific Research on Priority Areas, MEXT and CREST, JST.

## References

- [1] B. K. Alpert and V. Rokhlin: A Fast Algorithm for the Evaluation of Legendre Expansions, *SIAM J. Sci. Stat. Comput.*, Vol. 12, No. 1, pp. 158-179, 1991.
- [2] G. Beylkin, R. R. Coifman, and V. Rokhlin: Fast Wavelet Transforms and Numerical Algorithms I, *Comm. Pure Appl. Math.*, No. 44, pp. 141-183, 1991.
- [3] J. P. Boyd: Multipole expansions and pseudospectral cardinal functions: A new generation of the fast Fourier transform, *J. Comput. Phys.*, Vol. 103, pp. 184-186, 1992.
- [4] A. Dutt, M. Gu, and V. Rokhlin: Fast algorithms for polynomial interpolation, integration, and differentiation, *SIAM J. Numer. Anal.*, Vol. 33, No. 5, pp. 1689-1711, 1996.
- [5] L. Greengard and V. Rokhlin: A Fast algorithm for particle simulations, *J. Comput. Phys.*, Vol. 73, pp. 325-, 1987.
- [6] J. R. Driscoll and D. Healy, Asymptotically Fast Algorithms for Spherical and Related Transforms, *Proc. 30th IEEE FOCS*, pp. 344-349 (1989).
- [7] D. M. Healy Jr., D. Rockmore, P. J. Kostelec, and S. S. B. Moore, FFTs for 2-Sphere — Improvements and Variations, *Tech. Rep. PCS-TR96-292*, Dartmouth Univ., 1996.
- [8] R. Jakob-Chien and B. K. Alpert, A Fast Spherical Filter with Uniform Resolution, *J. Comput. Phys.*, Vol. 136, pp. 580-584, 1997
- [9] J. Keiner and D. Potts, Fast Evaluation of Quadrature Formulae on the Sphere, *Math. Comp.*, Vol. 77, No. 261, pp. 397-419 (2008).
- [10] S. Kunis and D. Potts, Fast spherical Fourier algorithms, *J. Comp. Appl. Math.*, Vol. 161, pp. 75-98 (2003).
- [11] M. J. Mohlenkamp, A Fast Transform for Spherical Harmonics, *J. Fourier Anal. Appl.*, Vol. 2, pp. 159-184, 1999.
- [12] A. Mori, R. Suda and M. Sugihara, An improvement on Orszag's Fast Algorithm for Legendre Polynomial Transform, *Trans. Inform. Process. Soc. Japan*, Vol. 40, No. 9, pp. 3612-3615 (1999).
- [13] S. A. Orszag, Fast Eigenfunction Transforms, *Science and Computers, Adv. Math. Supplementary Studies*, Vol. 10, pp. 23-30, Academic Press, 1986.
- [14] D. Potts, G. Steidl, and M. Tasche, Fast and stable algorithms for discrete spherical Fourier transforms, *Linear Algebra Appl.*, pp. 433 - 450, 1998.
- [15] V. Rokhlin and M. Tygert: Fast algorithms for spherical harmonic expansions, *SIAM J. Sci. Comp.* Vol. 27, No. 6, pp. 1903-1928 (2006).
- [16] G. Szegö, *Orthogonal Polynomials*, p. 193. AMS, 1959.
- [17] R. Suda, A Fast Spherical Harmonics Transform Algorithm, *Infor. Proc. Soc. Japan SIG Notes*, No. 98-HPC-73, pp. 37-42, 1998.
- [18] R. Suda: Stability Control of Fast Spherical Harmonics Transform, *Info. Proc. Soc. Japan SIG Notes*, No. 2001-HPC-88, pp. 43-48 (2001).
- [19] R. Suda: Fast spherical harmonics transform of FLTSS and its evaluation, *The 2002 Workshop on the Solution of Partial Differential Equations on the Sphere*, Fields Inst., U. Toronto (2002).
- [20] R. Suda: Fast spherical harmonic transform routine FLTSS applied to the shallow water test set, *Mon. Wea. Rev.*, Vol. 133, No. 3, pp. 634-648 (2005).
- [21] R. Suda: Stability analysis of the fast Legendre transform algorithm based on the fast multipole method, *Proc. Estonian Acad. Sci. Phys. Math.*, Vol. 53, No. 2, pp. 107-115 (2004).

- [22] R. Suda: Fast Spherical Harmonic Transform with generalized Fast Multipole Method, *2005 International Conference on Scientific Computing and Differential Equations (SciCADE05)*, p. 86 (2005)
- [23] R. Suda: High Performance Implementation and Evaluation of the Spherical Harmonic Transforms in the Fast Orthogonal Function Transform Routine FXTPACK, *Info. Proc. Soc. Japan SIG Technical Report*, Vol. 2005, No. 97 (2005-HPC-104), pp. 31–36 (2005).
- [24] R. Suda: FLTSS home page:  
<http://www.na.cse.nagoya-u.ac.jp/~reiji/fltss/>
- [25] R. Suda and M. Takami: A Fast Spherical Harmonics Transform Algorithm, *Math. Comp.*, Vol. 71, No. 238, pp. 703–715 (2002).
- [26] R. Suda and M. Takami: Error Analysis and Control of Fast Spherical Harmonics Transform, *Info. Proc. Soc. Japan SIG Notes*, No. 2000-HPC-84, pp. 7–12 (2000).
- [27] R. Suda and M. Takami: Error Analysis and Control of Fast Spherical Harmonics Transform, *Trans. Info. Proc. Soc. Japan HPS*, Vol. 42, No. SIG12 (HPS4), pp.49–59 (2001).
- [28] R. Suda and S. Kuriyama: Another Preprocessing Algorithm for Generalized One-Dimensional Fast Multipole Method, *J. Comp. Phys.*, Vol.195, pp. 790–803 (2004).
- [29] N. Yarvin and V. Rokhlin: A Generalized One-Dimensional Fast Multipole Method with Application to Filtering of Spherical Harmonics, *J. Comp. Phys.*, Vol. 147, pp. 594–609 (1998).