

A FAST SPHERICAL HARMONICS TRANSFORM ALGORITHM

REIJI SUDA AND MASAYASU TAKAMI

ABSTRACT. The spectral method with discrete spherical harmonics transform plays an important role in many applications. In spite of its advantages, the spherical harmonics transform has a drawback of high computational complexity, which is determined by that of the associated Legendre transform, and the direct computation requires time of $O(N^3)$ for cut-off frequency N . In this paper, we propose a fast approximate algorithm for the associated Legendre transform. Our algorithm evaluates the transform by means of polynomial interpolation accelerated by the Fast Multipole Method (FMM). The divide-and-conquer approach with split Legendre functions gives computational complexity $O(N^2 \log N)$. Experimental results show that our algorithm is stable and is faster than the direct computation for $N \geq 511$.

1. INTRODUCTION

The discrete spherical harmonics transform plays an essential role in many applications, such as those in computational physics/chemistry/astronomy. The resolution of the spherical harmonics expansion is uniform on a sphere, thus stable and reliable numerical computation is possible. The Laplace equation is easy to solve because the spherical harmonics are the eigenfunctions of the Laplace operator on a sphere. Those advantages make numerical computations with the spherical harmonics transform highly reliable. However, the spherical harmonics transform lacks a fast transform algorithm. The computational complexity of the direct computation of the spherical harmonics transform is $O(N^3)$ for cut-off frequency N . There are some propositions of fast spherical harmonics transform algorithms, but they need further developments for practical use.

In this paper, we propose a novel fast spherical harmonics transform algorithm that runs in time $O(N^2 \log N)$. Our algorithm is based on fast polynomial interpolation accelerated by the FMM (Fast Multipole Method). We introduce split Legendre functions, which enables the numerically stable divide-and-conquer approach. Experimental results show that our algorithm is faster than the direct computation for $N \geq 511$ and is numerically stable.

This paper is organized as follows. The rest of this section provides some basic properties of the spherical harmonics transform and surveys other research on

Received by the editor January 24, 2000 and, in revised form, July 10, 2000.

2000 *Mathematics Subject Classification.* Primary 65T99, 42C10.

Key words and phrases. Spherical harmonics transform, associated Legendre transform, fast transform algorithm, computational complexity.

This research is partly supported by the Japan Society for Promotion of Science (Computational Science and Engineering for Global Scale Flow Systems Project), Grant-in-Aid #11450038 of the Ministry of Education, and the Toyota Physical and Chemical Research Institute.

fast spherical harmonics transform algorithms. Our algorithm is explained in Section 2, where split Legendre functions, the linear-time polynomial interpolation algorithm, stabilization of the interpolation, and the divide-and-conquer approach are discussed, and experimental results are presented. Section 3 presents a brief summary and direction for future work.

1.1. The spherical harmonics transform. We consider the discrete spherical harmonics transform *evaluation* on a set of *evaluation points* $\{(\lambda_j, \mu_k) : 1 \leq j \leq J, 1 \leq k \leq K\}$ ($-1 \leq \mu_k \leq 1, 0 \leq \lambda_j < 2\pi$)

$$(1) \quad g(\lambda_j, \mu_k) = \sum_{m=0}^N \sum_{n=m}^N g_n^m Y_n^m(\lambda_j, \mu_k).$$

The *expansion* that is to compute g_n^m from $g(\lambda, \mu)$ can be done by the Gauss integral scheme

$$g_n^m = \sum_{j=1}^J \sum_{k=1}^K w_{jk} g(\lambda_j, \mu_k) Y_n^m(\lambda_j, \mu_k),$$

where λ_j and μ_k are the integral nodes and w_{jk} are the weights. For ease of the expansion, the evaluation points are usually chosen as the integral nodes. In many cases, the number of points J and K are determined by the so-called *alias-free condition*

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

We assume $J \approx 3N$ and $K \approx 3N/2$ in the following discussion.

A spherical harmonic $Y_n^m(\lambda, \mu)$ is the product of a trigonometric function and an associated Legendre function P_n^m

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

Thus, the spherical harmonics transform can be evaluated by successive computations of the associated Legendre transform evaluation and of the inverse Fourier transform

$$(3) \quad \begin{aligned} g^m(\mu_k) &= \sum_{n=m}^N g_n^m P_n^m(\mu_k), \\ g(\lambda_j, \mu_k) &= \sum_{m=0}^N g^m(\mu_k) e^{im\lambda_j}. \end{aligned}$$

Similarly, the spherical harmonics expansion becomes

$$(4) \quad \begin{aligned} g^m(\mu_k) &= \frac{1}{J} \sum_{j=1}^J g(\lambda_j, \mu_k) e^{-im\lambda_j}, \\ g_n^m &= \sum_{k=1}^K w_k g^m(\mu_k) P_n^m(\mu_k). \end{aligned}$$

In both transforms, the Fourier transforms can be done by the FFT and consume time $O(N^2 \log N)$, because $J = O(N)$ and $K = O(N)$. Usually, the associated Legendre transforms are directly computed and require time $O(N^3)$. Therefore, a fast associated Legendre transform algorithm is required to accelerate the spherical harmonics transform.

The associated Legendre transforms (3) and (4) are matrix-vector products, and the matrix for the expansion is the transposed matrix for the evaluation. Thus, it is enough to consider only the evaluation problem (3).

1.2. Other research for the fast transform. This subsection surveys some algorithms for the fast associated Legendre transform. An associated Legendre function $P_n^m(x)$ is related to an *ultraspherical polynomial* $q_{n-m}^m(x)$ with some constant c_n^m as

$$(5) \quad P_n^m(x) = c_n^m P_m^m(x) q_{n-m}^m(x).$$

Using that relation, the associated Legendre transform can be done through polynomial computations. Some researchers [6, 12] proposed algorithms of the associated Legendre transform based on fast polynomial computation algorithms with FFT or FCT (Fast Cosine Transform). Those algorithms are precise (i.e., not approximate) and run in time $O(N^2 \log^2 N)$, but tend to lack numerical stability. Some devices to remedy the instability are proposed [12], but the computational complexity under those devices is unknown.

Mohlenkamp [8] reported that a wavelet approach attains fast transform algorithms that run in time $O(N^{5/2} \log N)$ and $O(N^2 \log^2 N)$. The performance of the algorithm of complexity $O(N^{5/2} \log N)$ was much improved [9], but performances are reported only when N 's are powers of two.

There are some algorithms for the fast Legendre polynomial transform (i.e., only for $m = 0$). Alpert and Rokhlin [1] showed that the Legendre polynomial expansion can be transformed into the Chebyshev polynomial expansion in time $O(N)$. Beylkin et al. [2] showed that the same transform can be done in linear time using a wavelet approach. The Chebyshev polynomial expansion can be evaluated in time $O(N \log N)$ using FFT. Their approaches are based on the similarity of the Legendre polynomials and the Chebyshev polynomials, so the performance will be worse for large m .

Orszag [11] proposed a fast evaluation scheme based on the WKB approximation for Sturm-Liouville eigenfunction transforms including the associated Legendre transform. His algorithm can be improved to have the computational complexity $O(N \log N)$ for $m = 0$ [10]. For higher m , although his scheme is applicable, the precision of asymptotic approximations becomes worse, and the computational costs will increase.

The following research is on a fast algorithm for related computations. Boyd [3] pointed out that the FMM enables linear-time interpolation of the functions expanded by the associated Legendre functions. That fact is important in our context in two ways. First, transform algorithms are free from restrictions on the evaluation points. The value at any point can be computed in linear time from the values on other points. Second, one can accelerate the associated Legendre transform by interpolation. We can evaluate the transform on some $N - m + 1$ points, then interpolate the values on the other required points. Assuming the alias-free condition (2), the asymptotic computational costs become 4/9 of that without fast interpolation, but are still $O(N^3)$. However, considerations on the stability of the interpolation are lacking in his discussion. We enhance his idea with split Legendre functions and stabilization of the interpolation.

Jakob-Chien and Alpert [7] proposed a fast spherical filter algorithm that runs in time $O(N^2 \log N)$. Their algorithm also uses FMM, and their experiments show high stability and high speed.

2. OUR FAST TRANSFORM ALGORITHM

In this section, we propose a novel fast associated Legendre transform algorithm. First, split Legendre functions are introduced, and some of their properties are discussed. Second, the linear-time interpolation algorithm with the FMM, which is pointed out by Boyd [3], is reviewed, and a stabilization scheme is proposed. Third, the divide-and-conquer approach in our algorithm is explained, and the computational complexity of our algorithm is discussed. Last, some experimental results on speed and precision are reported.

2.1. Split Legendre functions. Associated Legendre functions satisfy the recurrence formula

$$(2n+1)xP_n^m(x) = (n-m+1)P_{n+1}^m(x) + (n+m)P_{n-1}^m(x).$$

From that formula, we can easily see that the following theorem holds.

Theorem 1. *An associated Legendre function can be split into the sum of two functions as*

$$(6) \quad P_n^m(x) = P_{n,\nu}^{m,0}(x) + P_{n,\nu}^{m,1}(x),$$

where each split Legendre function $P_{n,\nu}^{m,l}(x)$ ($l = 0, 1$) is the product of a polynomial and an associated Legendre function as

$$(7) \quad P_{n,\nu}^{m,l}(x) = q_{n,\nu}^{m,l}(x)P_{\nu+l}^m(x).$$

That polynomial becomes $q_{n,\nu}^{m,l}(x) \equiv 0$ if $n - \nu + l - 1 = 0$, otherwise $\deg(q_{n,\nu}^{m,l}(x)) = |n - \nu + l - 1| - 1$.

The split is trivial for $\nu = n$ or $\nu = n - 1$ and coincides with the recurrence formula for $\nu = n + 1$ and $\nu = n - 2$. For the other values of ν , we can easily prove it by induction. The polynomial $q_{n,\nu}^{m,l}(x)$ is called the *shifted Legendre polynomial* by some researchers [6].

We call the parameter ν the *split point*. Splits at different split points

$$\begin{aligned} P_n^m(x) &= P_{n,\nu_0}^{m,0}(x) + P_{n,\nu_0}^{m,1}(x) \\ &= P_{n,\nu_1}^{m,0}(x) + P_{n,\nu_1}^{m,1}(x) \end{aligned}$$

are related as

$$(8) \quad \begin{aligned} P_{n,\nu_1}^{m,0}(x) &= T_{\nu_0,\nu_1}^{m,00}(x)P_{n,\nu_0}^{m,0}(x) + T_{\nu_0,\nu_1}^{m,10}(x)P_{n,\nu_0}^{m,1}(x), \\ P_{n,\nu_1}^{m,1}(x) &= T_{\nu_0,\nu_1}^{m,01}(x)P_{n,\nu_0}^{m,0}(x) + T_{\nu_0,\nu_1}^{m,11}(x)P_{n,\nu_0}^{m,1}(x), \end{aligned}$$

where the coefficients are

$$T_{\nu_0,\nu_1}^{m,l_0l_1}(x) = P_{\nu_0+l_0,\nu_1}^{m,l_1}(x)/P_{\nu_0+l_0}^m(x).$$

Note that $T_{\nu_0,\nu_1}^{m,l_0l_1}(x)$ is independent of n . The equation (8) provides a method to compute the split at ν_1 from the split at ν_0 . We call that process the *shift of split points*.

The split (6) and the shift (8) are numerically stable for $\nu \leq n$ and $\nu_1 \leq \nu_0$. Their stability is directly connected with that of the recurrence formula, and they are

found in the direct computation (known as Horner’s rule or Clenshaw algorithm). In the direct computation, the associated Legendre transform is computed as

$$g^m(\mu_k) = c_0^1(\mu_k)P_{m+1}^m(\mu_k) + c_0^0(\mu_k)P_m^m(\mu_k),$$

where the $c_n^l(\mu_k)$ are computed through the dual recurrence

$$\begin{aligned} c_n^1(\mu_k) &= \frac{(2n+1)\mu_k}{n-m+1}c_{n+1}^1(\mu_k) + c_{n+1}^0(\mu_k) \\ c_n^0(\mu_k) &= \frac{n+m}{n-m+1}c_{n+1}^1(\mu_k) + g_n^m \end{aligned}$$

with initial values $c_{N-1}^1 = g_N^m$ and $c_{N-1}^0 = g_{N-1}^m$. Defining

$$s_n^l(\mu_k) = c_n^l(\mu_k)P_{n+l}^m(\mu_k),$$

we can see that these values are the split partial sum

$$s_n^l(\mu_k) = \sum_{n'=n}^N g_{n'}^m P_{n',n}^{m,l}(\mu_k),$$

and that the dual recurrence is a shift of the split point from $n+1$ to n .

However, it is known that the backward recurrence formula

$$P_{n-1}^m(x) = \frac{2n+1}{n+m}xP_n^m(x) - \frac{n-m+1}{n+m}P_{n+1}^m(x)$$

is unstable for $m \leq n < 2m$. For $n \geq 2m$ it behaves better, but the stability is not as good as the forward recurrence.

2.2. The linear-time interpolation algorithm. Next, we remind the reader of the linear-time interpolation algorithm [3]. Assume that the values of a polynomial $p(x)$ are known on a set of points (*sampling points*) $\{x_i\}_{i=1}^N$. If the number of points $N > \text{deg}(p)$, then the values of the polynomial on another set of points (*target points*) $\{y_j\}_{j=1}^M$ can be computed by interpolation:

$$p(y_j) = \omega(y_j) \sum_{i=1}^N \frac{1}{y_j - x_i} \frac{p(x_i)}{\omega_i(x_i)},$$

where $\omega(x) = \prod_{i=1}^N (x - x_i)$ and $\omega_i(x) = \omega(x)/(x - x_i)$. If the sets $\{x_i\}$ and $\{y_j\}$ are fixed, then we can pre-compute $\omega(y_j)$ and $\omega_i(x_i)$ and do the interpolation in three steps:

$$\begin{aligned} \xi_i &= \frac{p(x_i)}{\omega_i(x_i)} && \text{for } i = 1, \dots, N, \\ \eta_j &= \sum_{i=1}^N \frac{\xi_i}{y_j - x_i} && \text{for } j = 1, \dots, M, \\ p(y_j) &= \omega(y_j)\eta_j && \text{for } j = 1, \dots, M. \end{aligned}$$

It is clear that the first and the third steps are computed in time $O(N)$ and $O(M)$, respectively. The second step can be computed approximately in time $O((N+M)\log(1/\epsilon))$ by the Fast Multipole Method (FMM) [5], where ϵ is a parameter that determines the precision. The original paper on the FMM [5] assumes uniform distribution of the points, but it is proven that the FMM runs in linear time for any distribution of the points [13].

The fast interpolation algorithm is applicable to associated Legendre functions. Because an associated Legendre function is factorized as (5), it can be interpolated as

$$P_n^m(y_j) = P_m^m(y_j)\omega(y_j) \sum_{i=1}^N \frac{1}{y_j - x_i} \frac{P_n^m(x_i)}{P_m^m(x_i)\omega_i(x_i)},$$

where $N > n - m$. The split Legendre functions can be interpolated similarly. The interpolation algorithm itself was pointed out by Boyd [3] and others [4].

2.3. Choosing sampling points for stable interpolation. Numerical stability of interpolation is mostly dependent on the set of sampling points $\{x_i\}$. Defining the interpolation matrix Θ as

$$\Theta_{ji} = \frac{P_m^m(y_j)\omega_i(y_j)}{P_m^m(x_i)\omega_i(x_i)},$$

the interpolation is unstable if $\max\{|\Theta_{ji}|\}$ is large. Because the resulting value $P_n^m(y_j)$ is independent of the sampling points, large Θ_{ji} entries mean canceling, which incurs loss of precision. Thus, it is necessary to keep $|\Theta_{ji}|$ small.

Θ_{ji} can be defined as the solution of the linear equation with coefficient matrix Ξ , whose entries are $\Xi_{ni} = P_n^m(x_i)$. Cramer's rule gives $\Theta_{ji} = \det \Xi^{(ji)} / \det \Xi$, where $\Xi^{(ji)}$ is the same as Ξ but the i -th column is replaced by $P_n^m(y_j)$. Thus, if the set of sampling points $\{x_i\}$ is chosen so as to maximize $|\det \Xi|$, then $|\Theta_{ji}| \leq 1$. That fact proves that stable interpolation is attainable by an appropriate choice of the sampling points.

However, we do not know of an efficient algorithm for choosing the best set of sampling points. It is a kind of combinatorial optimization, and a naïve algorithm takes an impractical amount of time for large problems.

We propose the following algorithm for choosing the sampling points for stable interpolation. Given the number of sampling points N and the set of evaluation points \mathcal{M} , the algorithm chooses sampling points $\{x_i\}_{i=1}^N$ from \mathcal{M} .

- (1) Initialize as $w_1(\mu) = P_m^m(\mu)$ for each $\mu \in \mathcal{M}$.
- (2) Set $i = 1$.
- (3) Choose the i -th sampling point $x_i \in \mathcal{M}$ such that $|w_i(x_i)| = \max |w_i(\mu)|$.
- (4) Let $w_{i+1}(\mu) = (\mu - x_i)w_i(\mu)$ for each $\mu \in \mathcal{M}$.
- (5) If $i < N$, then increment i and go to step 3.

$w_i(\mu)$ is the partially evaluated $P_m^m(\mu)\omega(\mu)$ for the first $i - 1$ sampling points. The algorithm chooses the point that gives the largest $|w_i(\mu)|$ as the next sampling point, so as to make denominators larger and numerators smaller in Θ_{ji} .

For sampling points for split Legendre functions, the sum of the sets of the sampling points for P_ν^m and $P_{\nu+1}^m$ will work. Although it does not change the asymptotic computational complexity, such a strategy doubles the number of sampling points and the computational costs. We propose the following algorithm for restricting the number of sampling points without affecting the numerical stability much.

- (1) Initialize as $w_1^0(\mu) = P_\nu^m(\mu)$ and $w_1^1(\mu) = P_{\nu+1}^m(\mu)$ for each $\mu \in \mathcal{M}$.
- (2) Set $i = 1$.
- (3) Choose the i -th sampling point $x_i \in \mathcal{M}$ so that $\min\{|w_i^0(x_i)| / \max\{|w_i^0(\mu)|\}, |w_i^1(x_i)| / \max\{|w_i^1(\mu)|\}\}$ is maximized.
- (4) Let $w_{i+1}^0(\mu) = (\mu - x_i)w_i^0(\mu)$ and $w_{i+1}^1(\mu) = (\mu - x_i)w_i^1(\mu)$ for each $\mu \in \mathcal{M}$.
- (5) If $i < N$, then increment i and go to step 3.

In the case of split functions, it might not be able to maximize $w^0(\mu)$ and $w^1(\mu)$ at the same time, so we use a max-min strategy. Although these algorithms are simple greedy optimizations, interpolations using those sampling points were quite stable in our experiments.

The linear-time interpolation algorithm with the above algorithms for choosing sampling points can be used to accelerate the associated Legendre transform by dividing it into two steps of computations:

- (1) Evaluate transforms on $N - m + 1 = \deg(q_N^m) + 1$ chosen points.
- (2) Interpolate the values on the other evaluation points.

The second step can be computed in $O(N^2)$ time using the linear-time interpolation algorithm. The first step takes $O(N^3)$ time, but the constant coefficient is $4/9$ of the direct computation, assuming $K = 3N/2$ from (2). To attain lower asymptotic computational complexity, acceleration of the first step is required.

2.4. The divide-and-conquer approach. Our algorithm uses the divide-and-conquer approach to accelerate the first step discussed in the previous subsection. To evaluate the transform on $N - m + 1$ points, our algorithm chooses the *division point* $n_0 \approx (m + N)/2$ and divides the sum into two partial sums of roughly equal size as

$$\begin{aligned} g^m(\mu_k) &= g_0^m(\mu_k) + g_1^m(\mu_k), \\ g_0^m(\mu_k) &= \sum_{n=m}^{n_0-1} g_n^m P_n^m(\mu_k), \\ g_1^m(\mu_k) &= \sum_{n=n_0}^N g_n^m P_n^m(\mu_k). \end{aligned}$$

Since $g_0^m(x)$ is the product of a polynomial of degree $n_0 - m$ and $P_m^m(x)$, it can be evaluated in two steps: evaluation on $n_0 - m$ sampling points and interpolation.

Evaluation of $g_1^m(\mu_k)$ can be done by interpolation as well. For efficient interpolation, the partial sum $g_1^m(\mu_k)$ should be split as

$$\begin{aligned} g_1^m(\mu_k) &= g_{1,\nu}^{m,0}(\mu_k) + g_{1,\nu}^{m,1}(\mu_k), \\ g_{1,\nu}^{m,0}(\mu_k) &= \sum_{n=n_0}^N g_n^m P_{n,\nu}^{m,0}(\mu_k), \\ g_{1,\nu}^{m,1}(\mu_k) &= \sum_{n=n_0}^N g_n^m P_{n,\nu}^{m,1}(\mu_k). \end{aligned}$$

Choosing $\nu = n_0$, each split partial sum becomes the product of a polynomial of degree $\leq N - n_0 + 1$ and an associated Legendre function. Thus we can efficiently evaluate $g_{1,\nu}^{m,l}(\mu_k)$ ($l = 0, 1$) by interpolation. If the numerical stability allows us to choose $\nu \approx (n_0 + N)/2$, the interpolation becomes more efficient because the degree of the polynomial is nearly halved.

The computations of $g_0^m(\mu_k)$ and $g_{1,\nu}^{m,l}(\mu_k)$ on the sampling points are computed recursively using polynomial interpolations. Each $g_{1,\nu}^{m,l}(\mu_k)$ is divided as

$$\begin{aligned} g_{1,\nu}^{m,l}(\mu_k) &= g_{10,\nu}^{m,l}(\mu_k) + g_{11,\nu}^{m,l}(\mu_k), \\ g_{10,\nu}^{m,l}(\mu_k) &= \sum_{n=n_0}^{n_{11}-1} g_n^m P_{n,\nu}^{m,l}(\mu_k), \\ g_{11,\nu}^{m,l}(\mu_k) &= \sum_{n=n_{11}}^N g_n^m P_{n,\nu}^{m,l}(\mu_k), \end{aligned}$$

and the $g_{1p,\nu}^{m,l}(\mu_k)$ ($p = 0, 1$) are again computed by interpolation. Here, the split point ν is not the best for them, so a better split point ν_p should be used for each p , and after interpolation, $g_{1p,\nu}^{m,l}(\mu_k)$ will be obtained by the shift of split points as

$$g_{1p,\nu}^{m,l}(\mu_k) = T_{\nu_p,\nu}^{m,0l} g_{1p,\nu_p}^{m,0}(\mu_k) + T_{\nu_p,\nu}^{m,1l} g_{1p,\nu_p}^{m,1}(\mu_k).$$

The recursion stops when the evaluation-interpolation scheme needs more computational costs than the direct computation.

The following pseudo-code describes the proposed algorithm. Here, a split pair of vectors $\langle g^0, g^1 \rangle$ is represented using a bar as \bar{g} .

```

getsingle( $m, \gamma, n_0, n_1, \mathcal{M}$ )
// COMPUTING  $g(\mu) = \sum_{n=n_0}^{n_1} \gamma_n^m P_n^m(\mu)$  FOR  $\mu \in \mathcal{M}$ 
if (interpolation is inefficient)
  Directly evaluate  $g$  on  $\mathcal{M}$ 
else // EVALUATE AND INTERPOLATE
  if ( $n_0 = m$ ) // INTERPOLATE WITHOUT SPLIT
    Choose the sampling points  $\mathcal{M}' \subseteq \mathcal{M}$ 
    if (divide-and-conquer is inefficient)
      Directly evaluate  $g$  on  $\mathcal{M}'$ 
    else
      Choose the division point  $n_m$ 
       $g_0 = \text{getsingle}(m, \gamma, n_0, n_m - 1, \mathcal{M}')$ 
       $g_1 = \text{getsingle}(m, \gamma, n_m, n_1, \mathcal{M}')$ 
       $g = g_0 + g_1$ 
    Interpolate  $g$  onto  $\mathcal{M}$ 
  else // INTERPOLATE WITH SPLIT
    Choose the split point  $\nu$  and the sampling points  $\mathcal{M}' \subseteq \mathcal{M}$ 
    if (divide-and-conquer is inefficient)
      Directly evaluate  $\bar{g}$  on  $\mathcal{M}'$ 
    else
      Choose the division point  $n_m$ 
       $\bar{g}_0 = \text{getdual}(m, \gamma, n_0, n_m - 1, \nu, \mathcal{M}')$ 
       $\bar{g}_1 = \text{getdual}(m, \gamma, n_m, n_1, \nu, \mathcal{M}')$ 
       $\bar{g} = \bar{g}_0 + \bar{g}_1$ 
    Interpolate  $\bar{g}$  onto  $\mathcal{M}$ 
     $g = g^0 + g^1$ 
  return  $g$ 
getdual( $m, \gamma, n_0, n_1, \nu, \mathcal{M}$ )
// COMPUTING  $\bar{g}: g^l(\mu) = \sum_{n=n_0}^{n_1} \gamma_n^m P_{n,\nu}^{m,l}(\mu)$  FOR  $l = 0, 1$  AND  $\mu \in \mathcal{M}$ 

```

```

if (interpolation is inefficient)
  Directly evaluate  $\bar{g}$  on  $\mathcal{M}$ 
else // EVALUATE AND INTERPOLATE
  Choose the split point  $\nu'$  and the sampling points  $\mathcal{M}' \subseteq \mathcal{M}$ 
  if (divide-and-conquer is inefficient)
    Directly evaluate  $\bar{g}$  on  $\mathcal{M}'$ 
  else
    Choose the division point  $n_m$ 
     $\bar{g}_0 = \text{getdual}(m, g, n_0, n_m - 1, \nu', \mathcal{M}')$ 
     $\bar{g}_1 = \text{getdual}(m, g, n_m, n_1, \nu', \mathcal{M}')$ 
     $\bar{g} = \bar{g}_0 + \bar{g}_1$ 
    Interpolate  $\bar{g}$  onto  $\mathcal{M}$ 
    Shift split point of  $\bar{g}$  to  $\nu$ 
return  $\bar{g}$ 

```

2.5. Computational complexity of the algorithm. In this subsection, we consider the computational complexity of our algorithm. In the analysis, the evaluation points are assumed to be fixed, and the following data are assumed to be pre-computed.

- (1) The division points n_m .
- (2) The split points ν and ν' .
- (3) The sampling points \mathcal{M}' .
- (4) The functions $P_{n_0}^m(\mu)$, $P_\nu^m(\mu)$ and $P_{\nu+1}^m(\mu)$ for direct computation.
- (5) The factors $P_{\nu+l}^m(y_j)\omega(y_j)$ and $P_{\nu+l}^m(x_i)\omega_i(x_i)$ for interpolations.
- (6) The coefficients $T_{\nu_0, \nu_1}^{m, l_1 l_0}(\mu)$ for shifts of the split points.
- (7) The trees for FMM.

Our algorithm uses the divide-and-conquer approach, and the major computational costs are of the interpolations and of the shifts of split points. The costs of both computations are linear to the number of points to be evaluated, which is linear to the number of the summand of the partial sum. Thus, the total computational costs at each recursion level is $O(N)$. Because the linear combinations can be divided into two roughly equal partial sums, the recursion stops in $O(\log N)$ levels. Therefore, the computational complexity to evaluate an associated Legendre expansion on $O(N)$ points is $O(N \log N)$. Since there are $O(N)$ transforms ($m = 0, \dots, N$) to evaluate, the computational complexity for the spherical harmonics transform is $O(N^2 \log N)$.

The above analysis assumes that the precision of FMM is fixed. Complexity of FMM is proportional to $\log(1/\epsilon)$, where ϵ is the precision of FMM. Experimental results show that our algorithm is quite stable, but we do not yet have any analytical bounds on the approximation errors of our algorithm.

The preprocessing steps 3–6 require time $O(N^3)$. We will not discuss how to choose the division points and the split points in this paper. However, $n_m = (n_0 + n_1)/2$ and $\nu = \nu' = n_0$ work and do not change the complexity of the transform.

2.6. Experimental results. This section reports some results of our implementation of the proposed algorithm. The program is written in C and compiled by `gcc` with option `-O2`. It is run on SUN Enterprise 450 (300MHz). The evaluation points are the nodes of the Gauss-Legendre integral. The associated Legendre functions

TABLE 1. The precision and the speed of the proposed algorithm. K : The order of expansions in FMM; N : the cut-off frequency; error: relative error of the results in the max norm; T_f : time for our algorithm; T_d : time for lazy direct computation; T_0 : estimated time for full direct computation; T_d/T_f : the speed-up rate against the lazy direct computation; T_0/T_f : the speed-up rate against the full direct computation. The times are measured in seconds.

K	N	error	T_f	T_d	T_0	T_d/T_f	T_0/T_f
10	341	5.13E-06	0.832	0.840	1.143	1.01	1.37
14	341	1.59E-08	0.848	0.881	1.168	1.04	1.38
18	341	1.59E-10	0.862	0.887	1.166	1.03	1.35
22	341	2.74E-13	0.896	0.875	1.133	0.98	1.27
10	511	6.14E-06	2.506	2.843	3.968	1.13	1.58
14	511	1.52E-08	2.734	3.004	4.079	1.10	1.49
18	511	8.61E-11	2.861	3.037	4.076	1.06	1.42
22	511	4.48E-13	3.052	3.035	4.061	0.99	1.33
10	682	6.53E-06	5.379	7.098	9.889	1.32	1.84
14	682	7.17E-08	6.198	7.050	9.796	1.14	1.58
18	682	1.03E-09	6.489	7.660	10.389	1.18	1.60
22	682	1.91E-12	6.477	6.928	9.449	1.07	1.46
10	1023	1.50E-05	16.063	25.033	35.329	1.56	2.20
14	1023	1.47E-07	18.440	26.637	36.883	1.44	2.00
18	1023	1.57E-09	19.661	25.941	35.863	1.32	1.82
22	1023	1.36E-11	21.583	26.827	36.499	1.24	1.69
10	1365	8.89E-06	33.921	62.418	88.677	1.84	2.61
14	1365	7.29E-08	38.810	64.743	91.548	1.67	2.36
18	1365	1.01E-09	43.691	63.689	88.830	1.46	2.03
22	1365	1.59E-11	47.668	63.979	88.343	1.34	1.85
10	2047	3.37E-05	98.062	233.450	333.790	2.38	3.40
14	2047	8.38E-07	115.169	235.493	333.215	2.04	2.89
18	2047	1.25E-09	124.836	232.700	325.763	1.86	2.61
22	2047	2.54E-11	139.115	237.069	333.282	1.70	2.40
10	2730	1.24E-05	202.077	567.975	816.783	2.81	4.04
14	2730	1.48E-07	242.160	566.347	808.492	2.34	3.34
18	2730	1.60E-09	271.347	573.762	817.067	2.11	3.01
22	2730	2.57E-11	296.666	587.312	825.074	1.98	2.78
10	4095	2.92E-05	567.141	2078.464	2983.363	3.66	5.26
14	4095	1.75E-07	675.425	2131.294	3051.445	3.16	4.52
18	4095	2.25E-09	774.313	2231.551	3146.774	2.88	4.06
22	4095	7.39E-11	869.345	2069.444	2957.820	2.38	3.40

are scaled as

$$\tilde{P}_n^m(x) = \sqrt{(2n+1) \frac{(n-m)!}{(n+m)!}} P_n^m(x),$$

which gives $\int_{-1}^1 |\tilde{P}_n^m(x)|^2 dx = 2$.

For both the direct computation and the fast transform, the computational costs are halved by symmetry. Furthermore, we drop computations for $|P_n^m(x)| \leq \theta$ (the threshold θ is chosen according to the required precision) in the direct computation,

including that used in the proposed algorithm. We call the direct computation with that dropping the “lazy direct computation” and that without the dropping the “full direct computation”. Such a device improves not only the performance of the direct computation without much affecting the precision, but also the numerical stability because it avoids overflows in computing the dual recurrence.

In our algorithm, the split point ν is chosen as the lower bound ($\nu = n_0$ in the pseudo-code in the previous section), because we found that the error is quite sensitive to the stability of the split. The division point n_m is chosen as $(n_0 + n_1)/2$. That is a natural choice, but is not necessarily the best for performance. The implementations, especially that of the FMM, are rather rough, and they can be refined and tuned for higher performance.

Table 1 reports the precision and the performance of our algorithm. K and N are the order of the expansions in the FMM and the cut-off frequency, respectively. We chose 20 values for $m = 0, N/20, \dots, 19N/20$, then the time for each m is measured, and the sum of them is multiplied by $N/20$. (We cannot help using sampling-based evaluations for large N , because the preprocessing takes a considerable amount of time to avoid underflows for large m . To estimate the influence of the sampling, we have run the program for all m for some parameter sets. The difference on the relative performance is less than 5%; the total time consumption is 5–10% less than that of sampling-based estimation; and the errors are 2–10 times larger than those in the table.) The estimated transform times obtained in that way are shown in the table: T_f is the time for our algorithm; T_d is the time for the lazy direct computation; and T_0 is the time for the full direct computation, which is estimated from T_d by counting the number of the floating-point operations. T_d/T_f and T_0/T_f are the speed-up rates of our algorithm against the lazy direct computation and the full direct computation, respectively. To estimate the precision of our algorithm, the transform is computed for several random vectors, and the maximum of the relative difference in the max norm of the vectors computed with our algorithm and with the direct computation is shown in the “error” column of the table.

The computational costs of the lazy direct computation is about $2/3$ of that of the full direct computation, and slightly decreases for smaller K , because we set larger thresholds for the dropping. Our algorithm in the current implementation does not accelerate the transform for $N = 341$, but the speed-up rate gradually increases for larger N . The times for $N = 4095$ is about 2.8 times larger than the times for $N = 2730$. The coefficient 2.8 is a little larger than $1.5^2 = 2.25$, but clearly smaller than $1.5^3 = 3.375$.

Our algorithm is quite stable, and the relative error is only a few orders larger than the precision of the FMM. The errors for the same K seem to increase for larger N . The main reason for the small error for small N is that the approximation is restricted to small m (our scheme uses the direct computation if interpolation is inefficient), where the relative error tends to be small.

3. SUMMARY

In this paper, we proposed a novel fast stable algorithm for the spherical harmonics transform. Our algorithm is a fast approximate evaluation of the associated Legendre transform. The computational complexity is reduced using the linear-time polynomial interpolation algorithm based on the FMM (Fast Multipole Method)

and the divide-and-conquer approach with split Legendre functions. In experimental implementation, our algorithm was faster than the direct computation for $N \geq 511$.

However, current implementation of the FMM is rather primitive, and the Chebyshev approximation or the singular value decomposition must be introduced to attain higher performance. Because the distribution of the points is far from uniform, it requires some devices to cope with non-uniformity in the FMM. The effects of the choices on split points, sampling points, and division points on the approximation error and on the computational costs must be clarified. Then the trade-off between the approximation error and the computational costs should be solved. The storage requirements for the transform are also to be considered. The time and the storage for preprocessing must be reduced.

Our algorithm was successful in its stability, while some earlier algorithms [6, 12] had difficulty at this point. They share the divide-and-conquer approach with us, where a partial sum is represented by a linear combination $\sum c_j \psi_j(x)$. The speed of the algorithms comes from the efficiency of the algorithm to add two partial sums in that representation. The difference of the stability comes from the stability of the representation, that is, how much the linear combination is distorted if the coefficients c_j are rounded. In our algorithm, the stability of the representation is indirectly controlled by choosing the sampling points so that the entries in the interpolation matrix become as small as possible. The sampling points are chosen for each partial sum, so the variety of the characteristics of the associated Legendre functions is not a problem.

However, complete analysis on the stability and the precision is not done. Analysis, prediction, and control of the approximation errors will be necessary for higher reliability. It will require more effort to obtain practical analytical bounds on the approximation errors.

ACKNOWLEDGMENTS

The authors would like to express their sincere gratitude to Professor Sugihara and Professor Kaneda for their support, encouragement, and suggestions. We are thankful to Professor Yoden, Dr. Akahori and Dr. Yoshida for very valuable discussion. In addition, we thank the reviewer, because of whose comments our paper had essential improvements.

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. MR **91i**:65042
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. MR **92c**:65061
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. MR **93h**:65175
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. MR **97h**:65015
5. L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations", *J. Comput. Phys.*, Vol. 73, pp. 325-348, 1987. MR **88k**:82007
6. 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.
7. R. Jakob-Chien and B. K. Alpert, "A Fast Spherical Filter with Uniform Resolution", *J. Comput. Phys.*, Vol. 136, pp. 580-584, 1997.

8. M. J. Mohlenkamp, "A Fast Transform for Spherical Harmonics", PhD dissertation, Yale University, 1997.
9. _____, "A Fast Transform for Spherical Harmonics", *J. Fourier Anal. Appl.*, Vol. 2, pp. 159–184, 1999. MR **2000b**:65247
10. 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). CMP 2000:04
11. S. A. Orszag, "Fast Eigenfunction Transforms", Academic Press: Science and Computers, *Adv. Math. Supplementary Studies*, Vol. 10, pp. 23-30, 1986.
12. D. Potts, G. Steidl, and M. Tasche, "Fast and stable algorithms for discrete spherical Fourier transforms", *Linear Algebra Appl.* pp. 433 – 450, 1998. MR **99h**:65229
13. J. H. Reif and S. R. Tate, "N-body Simulation I: Fast Algorithms for Potential Field Evaluation and Trummer's Problem", Tech. Rep. N-96-002, Univ. of North Texas, Dept. of Computer Science (1996).

DEPARTMENT OF COMPUTATIONAL SCIENCE AND ENGINEERING, NAGOYA UNIVERSITY, FURO-CHO, CHIKUSA-KU, NAGOYA, 464-8603, JAPAN

E-mail address: reiji@na.cse.nagoya-u.ac.jp

DEPARTMENT OF COMPUTATIONAL SCIENCE AND ENGINEERING, NAGOYA UNIVERSITY, FURO-CHO, CHIKUSA-KU, NAGOYA, 464-8603, JAPAN

E-mail address: m-takami@kubota.co.jp