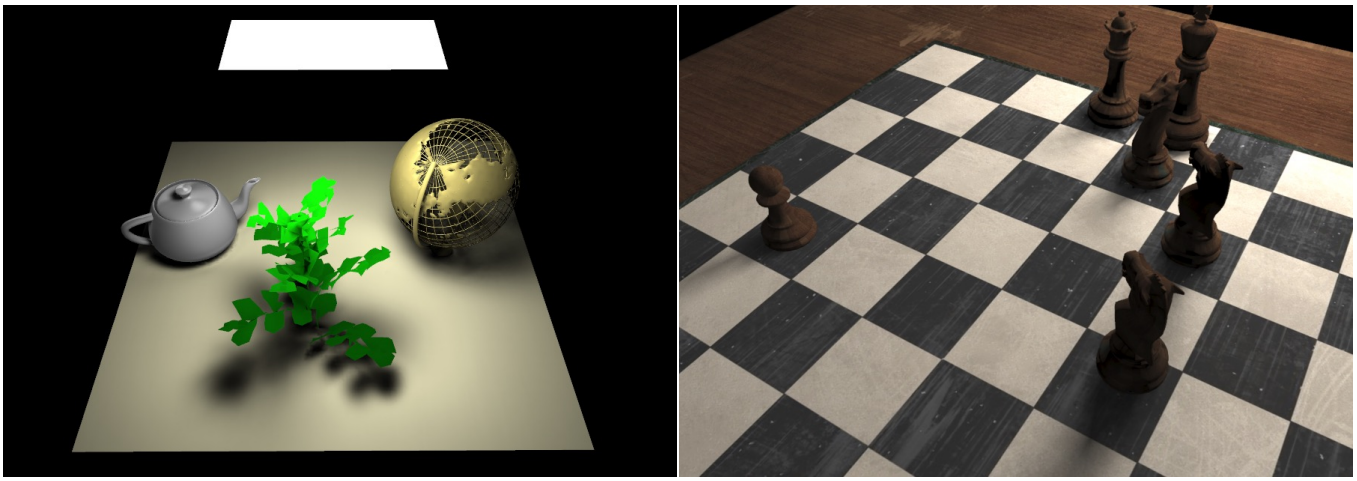
Fast and Accurate Spherical Harmonics Products

HANGGAO XIN, BNRist, Department of CS&T, Tsinghua University, China
ZHIQIAN ZHOU, BNRist, Department of CS&T, Tsinghua University, China
DI AN, BNRist, Department of CS&T, Tsinghua University, China
LING-QI YAN, University of California, Santa Barbara, United States of America
KUN XU*, BNRist, Department of CS&T, Tsinghua University, China
SHI-MIN HU, BNRist, Department of CS&T, Tsinghua University, China
SHING-TUNG YAU, Harvard University, United States of America



(a) PRT. 3.2 FPS (Trad. method) / 26 FPS (Ours)

(b) Shadow fields. 2.6 FPS (Trad. method) / 24 FPS (Ours)

Fig. 1. Performance comparison between our method and traditional method in rendering applications of glossy relighting using PRT (a) and dynamic scene rendering using shadow fields (b). Thanks to the efficiency in computing triple and multiple products, our method achieves a speedup of more than 8×. SH order $n = 15$ is used.

Spherical Harmonics (SH) have been proven as a powerful tool for rendering, especially in real-time applications such as Precomputed Radiance Transfer (PRT). Spherical harmonics are orthonormal basis functions and are efficient in computing dot products. However, computations of triple product and multiple product operations are often the bottlenecks that prevent moderately high-frequency use of spherical harmonics. Specifically, state-of-the-art methods for accurate SH triple products of order n have a time complexity of $O(n^5)$, which is a heavy burden for most real-time applications. Even worse, a brute-force way to compute k-multiple products would take $O(n^{2k})$ time. In this paper, we propose a fast and accurate method for spherical harmonics triple products with the time complexity of only $O(n^3)$, and further extend it for computing k-multiple products with the time complexity of $O(kn^3 + k^2n^2 \log(kn))$. Our key insight is to conduct the triple and multiple products in the Fourier space, in which the multiplications can be performed much more efficiently. To our knowledge, our method is theoretically the fastest for accurate spherical harmonics triple and multiple products. And in practice, we demonstrate the efficiency of our method in rendering applications including mid-frequency relighting and shadow fields.

CCS Concepts: • **Computing methodologies** → **Rendering**.

Additional Key Words and Phrases: spherical harmonics, Fourier transform, triple product, multiple product, precomputed radiance transfer

ACM Reference Format:
Hanggao Xin, Zhiqian Zhou, Di An, Ling-Qi Yan, Kun Xu, Shi-Min Hu, and Shing-Tung Yau. 2021. Fast and Accurate Spherical Harmonics Products.

*Kun Xu is the corresponding author.

Authors' addresses: Hanggao Xin, BNRist, Department of CS&T, Tsinghua University, Beijing, China, xhg18@mails.tsinghua.edu.cn; Zhiqian Zhou, BNRist, Department of CS&T, Tsinghua University, Beijing, China, zhouzq18@mails.tsinghua.edu.cn; Di An, BNRist, Department of CS&T, Tsinghua University, Beijing, China, and17@mails.tsinghua.edu.cn; Ling-Qi Yan, University of California, Santa Barbara, Santa Barbara, United States of America, lingqi@cs.ucsb.edu; Kun Xu, BNRist, Department of CS&T, Tsinghua University, Beijing, China, xukun@tsinghua.edu.cn; Shi-Min Hu, BNRist, Department of CS&T, Tsinghua University, Beijing, China, shimin@tsinghua.edu.cn; Shing-Tung Yau, Harvard University, Boston, United States of America, yau@math.harvard.edu.

0730-0301/2021/12-ART1 $15.00
https://doi.org/10.1145/3478513.3480563

ACM Trans. Graph. 40, 6, Article 1 (December 2021), 14 pages. https://doi.org/10.1145/3478513.3480563

1 INTRODUCTION

Spherical Harmonics (SH) have been proven as a powerful tool for rendering. As a set of 2D basis functions, spherical harmonics are able to represent and recover low frequency functions faithfully with only a few coefficients (9 or 16). Therefore, spherical harmonics are pervasively used in real-time applications such as Precomputed Radiance Transfer (PRT) [Sloan et al. 2002].

SH possess nice properties such as orthogonality, which immediately enables dot products at a computational cost linear to the number of SH functions, or equivalently, $O(n^2)$ where n is the order of SH. In PRT, when the lighting (usually from an environment map) and light transport (including diffuse BRDFs and visibilities) are both represented as vectors of SH coefficients, at each vertex on an object, the rendering equation, as a double product integral of lighting and light transport, will simply reduce to a dot product.

However, in order to enable more flexible and more general rendering tasks, it is often required to perform triple product and multiple product operations of spherical harmonics. For example, relighting [Ng et al. 2004; Ren et al. 2006] a static scene with dynamic lighting and BRDFs at a fixed view would require fast integration of lighting, BRDF, as well as the visibility per point. Another example is the shadow fields [Zhou et al. 2005] method that extends PRT to support moving objects. This is achieved by separately precomputing each object's Object Occlusion Field (OOF). At runtime, the OOFs will be multiplied to resolve the final visibilities at different vertices, which immediately gives rise to the multiple product problem.

Unfortunately, triple and multiple products are much more complex than dot products for spherical harmonics. The orthogonality property is no longer useful here, and there is no obvious sparsity to exploit as in the wavelet case [Ng et al. 2004]. Accurate solutions do exist [Ng et al. 2004; Sloan et al. 2002], but are extremely heavy – $O(n^5)$ for triple products, and $O(n^{2k})$ for k-multiple products [Sun and Mukherjee 2006]. Fast approximate methods also exist [Ren et al. 2006; Zhou et al. 2005], but reduce quality or introduce artifacts due to accumulated truncation errors.

In this paper, we challenge this long-standing problem of how to compute spherical harmonics triple and multiple products both quickly and accurately. We are inspired by the observation that SH basis functions are similar to Fourier basis functions. When we transform SH basis functions from the direction domain (namely the SH space) to the frequency domain (Fourier space), triple and multiple products can be performed much more efficiently. Besides, the conversion between the SH space and the Fourier space can be performed in an efficient way. Overall, we make the following contributions to the rendering community:

- *Theory*: we propose a new and complete theoretical framework of performing SH triple and multiple products in the Fourier space.
- *Performance*: we present a break down performance on each step of our proposed method, and achieve the overall time complexity of $O(n^3)$ instead of $O(n^5)$ for triple products, and $O(kn^3 + k^2n^2\log(kn))$ for k-multiple products, as opposed to $O(n^{2k})$ of a bruteforce approach, which is the best so far to our knowledge.
- *Analysis*: we perform a detailed analysis of our method on both CPUs and GPUs, with different orders of spherical harmonics, and different numbers of multiple product components. We also present thorough comparisons on performance and errors with different methods.
- *Applications*: we integrate our method into PRT applications, such as glossy relighting and shadow fields, to demonstrate that our method is in practice much faster than previous methods for real applications.

2 RELATED WORK

Precomputed Radiance Transfer (PRT), was proposed by Sloan et al. [2002]. They projected both environment lighting [Ramamoorthi and Hanrahan 2001] and light transport to a SH basis to enable dynamic lighting on diffuse and glossy materials. PRT and SHs have been further extended in many aspects, such as efficient SH computation [Lessig et al. 2014; Schaeffer 2013; Snyder 2006], efficient SH rotation [Nowrouzezahrai et al. 2012; Sloan et al. 2005], reduced precomputed storage [Sloan et al. 2003a], supporting translucent materials [Hao and Varshney 2004; Xu et al. 2007], glossy materials [Liu et al. 2004], dynamic geometry [Sloan et al. 2005; Wang et al. 2006], and bi-scale rendering effects [Sloan et al. 2003b].

Recently, Wang and Ramamoorthi [2018] derived an analytic solution to SH lighting of near-field polygonal area lights. Wu et al. [2020] further supported the near-field illumination of many lights by deriving closed-form SH gradients. Besides the aforementioned usage in real-time rendering, PRT has also been used in offline rendering [Pantaleoni et al. 2010]. We refer the readers to the surveys [Kautz et al. 2005; Lehtinen 2007; Ramamoorthi 2009] for more details. Note that, none of the above works aim at accurate and fast solutions to SH triple products or multiple products. Therefore, such operations would still be the bottleneck.

Another line of research uses other basis functions instead of SHs. Ng et al. [2003] used wavelet basis for all-frequency lighting and shadows, and Tsai and Shih [2006] chose to use Spherical Radial Basis Functions (SRBFs) for more accurate approximation. Later, more attention has been paid to Spherical Gaussians [Tsai and Shih 2006; Wang et al. 2009; Xu et al. 2011, 2013; Yamaguchi et al. 2020; Yan et al. 2012], a specific kind of SRBFs. These basis functions are able to provide all-frequency representation, but would often cost tens to hundreds times more memory than spherical harmonics basis. In addition, these basis functions often do not share the same nice properties with spherical harmonics. For example, the wavelets do not support fast rotations due to the non-linearity in its compression, and the SRBFs are not orthogonal. Therefore, the run-time computation can be more complex than with spherical harmonics.

Triple and multiple products. Accurate triple and multiple products for wavelet basis functions have been studied in depth. Ng et al. [2004] first introduced an accurate and fast way of computing Haar wavelet triple product integrals, with the performance linear to the number of non-zero wavelet coefficients. The work was later

extended by Sun and Ramamoorthi [2009] to consider affine transforms on the integrand, enabling near field lighting. A generalization from triple product integrals to multiple product integrals of Haar wavelets was also studied [Sun and Mukherjee 2006]. These methods successfully improved the practicality of wavelets as basis functions, and inspired us to develop a theory for accurate SH triple and multiple products.

Ng et al. [2004] also provided a general study of accurate SH triple products. The method described in their work has been the state-of-the-art for almost two decades, which has a complexity of $O(n^5)$ and requires the precomputation of Clebsch-Gordan coefficients [Inui et al. 1999; Thornber and Jacobs 2006]. However, methods for accurate SH multiple products have not been explicitly introduced yet. A naïve and bruteforce way to accurately compute a multiple product of $(k-1)$ functions will require $O(n^{2k})$ time [Sun and Mukherjee 2006], which is not affordable for most practical applications.

For SH multiple products, the most widely used approximation in previous works [Sun and Mukherjee 2006; Zhou et al. 2005] is to approximate a multiple product by continuously performing multiple times of triple products, which we refer to as *recursive triple product approximation*. However, the results can be significantly biased due to accumulated truncation error. Another approximation to multiple products is SH Exponentiation (SHEXP) [Ren et al. 2006], which proposes log and exp operators in SH space to turn multiplications into summations. However, SHEXP is not efficient for direct multiple product of arbitrary spherical functions due to its $O(n^6)$ log step, so it has to further approximate scene geometry with set of spheres and precompute log steps towards sphere primitives only. It is highly coupled with its scene geometry approximation and therefore loses generality.

It is worth noting that the Reproducing Kernel Hilbert Space (RKHS) method of Lessig et al. [2014] can also be used for computing SH triple and multiple products, although this was not explicitly discussed by the authors. However, it requires $O(n^4)$ time for accurate triple products and $O(k^3n^4)$ time for accurate k-multiple products, which is slower than our method.

3 BACKGROUND

To make our paper self-contained, in this section, we review the important operators used in general basis functions in Sec. 3.1, and review spherical harmonics and 2D Fourier series in Sec. 3.2.

3.1 Basis Function Operators

3.1.1 Projection and Reconstruction. Considering a set of orthonormal basis functions $\{B_i(\omega)\}$ defined over the unit sphere S^2, any spherical function $F(\omega)$ could be approximated as a linear combination of the basis functions:

$$F(\omega) \approx \sum_i f_i B_i(\omega), \tag{1}$$

where $\boldsymbol{f} = \{f_i\}$ are the coefficients of the corresponding basis functions, which are computed through *projection*:

$$f_i = \int_{S^2} F(\omega) B_i(\omega) \mathrm{d}\omega. \tag{2}$$

Note that approximately computing the spherical function $F(\omega)$ back from its coefficients $\boldsymbol{f}$ using Equ. 1 is called *reconstruction*.

In the following, we use N to denote the number of basis functions used for the linear approximation in Equ. 1. Note that order-n spherical harmonics has $N = n^2$ basis functions.

3.1.2 Dot Product. Given two spherical functions $F_1(\omega)$ and $F_2(\omega)$ represented by their basis function coefficients $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$, respectively (such that $F_1(\omega) \approx \sum_i f_{1,i} B_i(\omega)$ and $F_2(\omega) \approx \sum_i f_{2,i} B_i(\omega)$), the integral of their product could be simply computed as a dot product of the coefficients:

$$\int F_1(\omega) F_2(\omega) \mathrm{d}\omega \approx \sum_i f_{1,i} f_{2,i} = \boldsymbol{f_1} \cdot \boldsymbol{f_2}. \tag{3}$$

3.1.3 Triple Product. Considering three spherical functions $F_1(\omega)$, $F_2(\omega)$, and $F_3(\omega)$ represented by their basis function coefficients $\boldsymbol{f_1}, \boldsymbol{f_2}$ and $\boldsymbol{f_3}$, respectively, their triple product integral could be computed as [Ng et al. 2004]:

$$\int F_1(\omega) F_2(\omega) F_3(\omega) \mathrm{d}\omega \approx \sum_i \sum_j \sum_k \mathrm{C}_{ijk} f_{1,i} f_{2,j} f_{3,k}, \tag{4}$$

where C_{ijk} are the tripling coefficients defined as:

$$\mathrm{C}_{ijk} = \int B_i(\omega) B_j(\omega) B_k(\omega) \mathrm{d}\omega. \tag{5}$$

Multiplication of two spherical functions could be computed similarly. Suppose we have $G(\omega) = F_1(\omega) F_2(\omega)$. The coefficients of $G(\omega)$ could be computed as [Ng et al. 2004]:

$$g_k = \sum_i \sum_j \mathrm{C}_{ijk} f_{1,i} f_{2,j}. \tag{6}$$

The above multiplication process is also referred to as a *triple product*, and is usually denoted in the vector form: $\boldsymbol{g} = \boldsymbol{f_1} \otimes \boldsymbol{f_2}$, where $\otimes$ denotes the triple product operator.

For spherical harmonics, the tripling coefficients C_{ijk} are known as Clebsch-Gordan coefficients [Inui et al. 1999; Thornber and Jacobs 2006]. Among all n^6 tripling coefficients, only $O(n^5)$ of them are non-zero [Ng et al. 2004]. Hence, the time complexity to compute a triple product integral (Equ. 4) or a triple product (Equ. 6) is $O(n^5)$. Besides, a precomputation step is required to compute and store the tripling coefficients in advance.

3.1.4 Multiple product. The triple product could be extended to a more general case, i.e., multiple product. Specifically, a k-multiple product ($k \geq 3$) is referred to as the multiplication of $k-1$ spherical functions in the space of basis functions.

Suppose we would like to compute $G(\omega) = F_1(\omega) F_2(\omega) \cdots F_{k-1}(\omega)$. The k-multiple product in the vector form is given by:

$$\boldsymbol{g} = \otimes_k(\boldsymbol{f_1}, \boldsymbol{f_2}, \cdots, \boldsymbol{f_{k-1}}), \tag{7}$$

where $\boldsymbol{g}, \boldsymbol{f_1}, \boldsymbol{f_2}, \cdots, \boldsymbol{f_{k-1}}$ denote the coefficients of the corresponding functions, respectively, and $\otimes_k$ denotes the k-multiple product operator.

The *multiple product integral* of k spherical functions could be computed through a k-multiple product and a dot product:

$$\int F_1(\omega) f_2(\omega) \cdots F_k(\omega) \mathrm{d}\omega \approx \otimes_k(\boldsymbol{f_1}, \boldsymbol{f_2}, \cdots, \boldsymbol{f_{k-1}}) \cdot \boldsymbol{f_k}. \tag{8}$$

Note that triple product is could be viewed as a specialization of k-multiple product when $k = 3$.

Accurate brute-force approach. A brute force way to accurately computing the multiple product for general basis functions is given in [Ren et al. 2006; Sun and Mukherjee 2006] by extending the tripling coefficients to higher orders:

$$g_{j_k} = \sum_{j_1} \sum_{j_2} \cdots \sum_{j_{k-1}} [C^{(k)}_{j_1,j_2,\cdots,j_{k-1},j_k} \cdot f_{1,j_1} f_{2,j_2} \cdots f_{k-1,j_{k-1}}], \quad (9)$$

where the $k-$th order *integral coefficient* $C^{(k)}_{j_1,j_2,\cdots,j_{k-1},j_k}$ is computed through a integral:

$$C^{(k)}_{j_1,j_2,\cdots,j_{k-1},j_k} = \int B_{j_1}(\omega) B_{j_2}(\omega) \cdots B_{j_k}(\omega) \mathrm{d}\omega. \quad (10)$$

Such a brute-force approach is rather expensive. The time complexity for computing a k-multiple product and the space complexity to store the integral coefficients are both $O(N^k)$ for general basis functions. For spherical harmonics, the time complexity would be $O(n^{2k})$, which would be prohibitively expensive, especially when k is large. The time complexity might be slightly reduced by enumerating over only non-zero coefficients, as in triple products whose complexity is reduced from $O(n^6)$ to $O(n^5)$. However, we cannot find any existing researches that take use of such sparsity to reduce its time complexity.

3.1.5 Multiple product using recursive triple product approximation. A more practical method to compute the multiple product is by recursively applying triple products $(k-1)$ times [Sun and Mukherjee 2006; Zhou et al. 2005]:

$$\boldsymbol{g} \approx (\cdots((\boldsymbol{f_1} \otimes \boldsymbol{f_2}) \otimes \boldsymbol{f_3}) \otimes \cdots \otimes \boldsymbol{f_{k-1}}). \quad (11)$$

This is done by first computing $\boldsymbol{f_1} \otimes \boldsymbol{f_2}$, then computing $(\boldsymbol{f_1} \otimes \boldsymbol{f_2}) \otimes \boldsymbol{f_3}$, and so on. Its time complexity is $O(kn^5)$, which is usually acceptable in real-time applications for low-order spherical harmonics. However, this method only produces an approximation to the ground truth multiple product defined in Equ. 7. This is due to early and accumulated truncation: we need to truncate the intermediate result of each triple product within order n spherical harmonics. We will visualize and evaluate the truncation errors in Sec. 5.2.

3.2 Spherical Harmonics and 2D Fourier Series

3.2.1 Spherical harmonics. Spherical harmonics (SH) is a widely used series of orthonormal spherical basis functions. Order n spherical harmonics are defined as:

$$y^m_l(\omega) = y^m_l(\theta,\phi) = \begin{cases} K^0_l P^0_l(\cos\theta) & m = 0 \\ \sqrt{2} K^m_l \cos(m\phi) P^m_l(\cos\theta) & m > 0 \\ \sqrt{2} K^{|m|}_l \sin(|m|\phi) P^{|m|}_l(\cos\theta) & m < 0 \end{cases}, \quad (12)$$

where $\omega = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$, index m and band l satisfy that $-l \leqslant m \leqslant l, 0 \leqslant l < n$. K^m_l are normalization constants and $P^m_l(\cdot)$ are the associated Legendre polynomials defined as:

$$P^m_l(x) = (-1)^m \cdot 2^l \cdot (1-x^2)^{m/2} \cdot \sum_{k=m}^{l} \frac{k!}{(k-m)!} x^{k-m} \cdot \binom{l}{k} \cdot \binom{\frac{l+k-1}{2}}{l}.$$

Any spherical function F could be projected into its SH coefficients $\boldsymbol{f} = \{f_{l,m}\}$ such that:

$$F(\theta,\phi) \approx \sum_{l=0}^{n-1} \sum_{m=-l}^{l} f_{l,m} y^m_l(\theta,\phi). \quad (13)$$

3.2.2 2D Fourier series. Another common way to represent 2D functions is to use Fourier series. Any spherical function F could be represented as 2D Fourier series using (θ,ϕ) parameterization:

$$\begin{aligned} F(\theta,\phi) \approx \sum_{s,t} [&A_{s,t}\cos(s\theta)\cos(t\phi) + B_{s,t}\sin(s\theta)\cos(t\phi) \\ &+ C_{s,t}\cos(s\theta)\sin(t\phi) + D_{s,t}\sin(s\theta)\sin(t\phi)]. \end{aligned} \quad (14)$$

The above trigonometric form could be equivalently written in the complex form:

$$F(\theta,\phi) \approx \sum_{s,t} f^*_{s,t} e^{i(s\theta+t\phi)}. \quad (15)$$

Note that the Fourier coefficients $\boldsymbol{f^*} = \{f^*_{s,t}\}$ in the above equation are complex values. Its formula is slightly different from conventional 2D Fourier series (i.e., the exponential value is not multiplied by 2π), since it is defined for spherical functions. To be consistent with SH terminology, we call 2D Fourier series to have degree n if the values of s and t are both restricted in $[-(n-1), n-1]$.

It is easy to find that both spherical harmonics and 2D Fourier representations are essentially composed of bivariate polynomials of trigonometric functions. [Hofsommer and Potters 1960; Sneeuw and Bun 1996] provided methods to convert between the two representations in $O(n^3)$ time. The conversion is accurate and no approximations are involved.

3.2.3 Conversion from SH to Fourier series. Given a spherical function F represented by its SH coefficients $\boldsymbol{f} = \{f_{l,m}\}$ (Equ. 13), the conversion from SH to Fourier series is to find its Fourier coefficients $\boldsymbol{f^*} = \{f^*_{s,t}\}$ (Equ. 15).

There are two key observations. First, for both SH and Fourier representations, the two dimensions of θ and ϕ could be separably computed. Second, for spherical harmonics function $y^m_l(\theta,\phi)$ with band l and index m (See definition in Equ. 12), the ϕ part is composed of only $|m|$-th harmonics, i.e., $\cos(m\phi)$, $\sin(|m|\phi)$, and their combinations.

We could rewrite a spherical harmonics function $y^m_l(\theta,\phi)$ in 2D Fourier series:

$$y^m_l(\theta,\phi) = \sum_{\substack{-n<s<n \\ -n<t<n}} y^*_{l,m,s,t} e^{i(s\theta+t\phi)}, \quad (16)$$

where $\{y^*_{l,m,s,t}\}$ is a 4D tensor. By substituting the above equation to Equ. 13, the Fourier coefficients $\boldsymbol{f^*}$ could be obtained from SH coefficients $\boldsymbol{f}$ through:

$$f^*_{s,t} = \sum_{\substack{0 \leqslant l < n \\ -l \leqslant m \leqslant l}} f_{l,m} y^*_{l,m,s,t}. \quad (17)$$

Since the ϕ part of $y^m_l(\theta,\phi)$ is a $|m|$-th harmonics, it is easy to find that $y^*_{l,m,s,t}$ is non-zero only when $m = \pm t$. Hence, the 4D tensor $\{y^*_{l,m,s,t}\}$ could be reduced to 3D, and could be precomputed only once and stored. The time complexity of converting from SH coefficients to Fourier coefficients is $O(n^3)$. Please note that, 2D

Fourier series used to represent real functions must be conjugate symmetric, so $y^*_{l,m,s,t}$ is conjugate and such property could reduce a half computation of the conversion from SH to Fourier series.

3.2.4 Fourier series to SH. Given a spherical function F represented by its Fourier coefficients $\boldsymbol{f^*}$, its SH coefficients $\boldsymbol{f}$ could be computed similarly as in Sec. 3.2.3, but in an inverse way.

We could rewrite a Fourier term by a linear combination of SH basis functions:

$$e^{i(s\theta+t\phi)} = \sum_{\substack{0\leqslant l<n \\ -l\leqslant m\leqslant l}} h^*_{l,m,s,t} y^m_l(\theta,\phi), \tag{18}$$

where $\{h^*_{l,m,s,t}\}$ is a 4D tensor. Thus, we have:

$$f_{l,m} = \sum_{\substack{-n<s<n \\ -n<t<n}} h^*_{l,m,s,t} f^*_{s,t}. \tag{19}$$

Similarly, $h^*_{l,m,s,t}$ is non-zero only when $m = \pm t$. $\{h^*_{l,m,s,t}\}$ could be precomputed only once and stored as a 3D tensor. The time complexity of converting from Fourier coefficients to SH coefficients is $O(n^3)$.

4 OUR APPROACH FOR SH PRODUCTS

Given $(k-1)$ spherical functions represented by order-n SH coefficients: $\boldsymbol{f_1}, \boldsymbol{f_2}, \cdots, \boldsymbol{f_{k-1}}$, our goal is to compute the accurate order-n SH coefficients of their multiple product, i.e., $\boldsymbol{g} = \otimes_k(\boldsymbol{f_1}, \boldsymbol{f_2}, \cdots, \boldsymbol{f_{k-1}})$. Note that a multiple product is called a triple product when $k = 3$.

To avoid ambiguity, we would like to emphasize that our accurate method is targeted to accurately compute SH coefficients of the product within the order limit n. We do not consider the SH coefficients beyond the order limit for both the resulting product and the input spherical functions. Such design is reasonable since SH based representations usually use a fixed order in real applications.

4.1 Overview

Our key observation is, using the Fast Fourier Transform (FFT), multiplication can be performed more efficiently in Fourier space than in SH space. We design our method as follows:

(1) **Conversion from SH space to Fourier space.** First, we convert each spherical function from SH coefficients $\boldsymbol{f_i}$ to Fourier series $\boldsymbol{f^*_i}$ $(1 \leq i < k)$. See Sec. 3.2.3.
(2) **Convolution in Fourier space.** Then, we compute the multiplication of all converted Fourier series, which is essentially computing convolutions of Fourier coefficients and can be accelerated using FFT. The Fourier coefficients of the resulting product are denoted as $\boldsymbol{g^*}$. We describe it in detail in Sec. 4.2.
(3) **Conversion back to SH space**. Finally, we convert back the product from Fourier series $\boldsymbol{g^*}$ to SH coefficients $\boldsymbol{g}$. See Sec. 4.3.

The pipeline is visualized in Fig. 2. The whole computation is accurate and no approximations are involved.

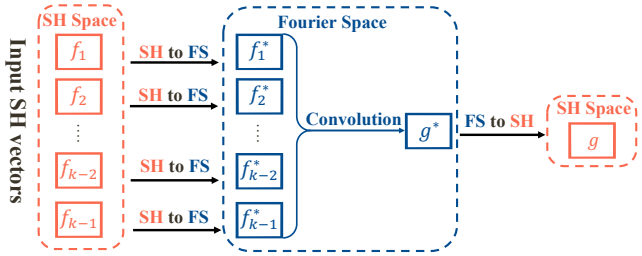


Fig. 2. The overview of our method for computing spherical harmonics triple products ($k = 3$) and multiple products ($k > 3$).

4.2 Convolution in Fourier space

Given $(k-1)$ spherical functions F_j $(1 \leq j < k)$ in Fourier series:

$$F_j(\theta,\phi) = \sum_{s,t} f^*_{j,s,t} e^{i(s\theta+t\phi)}, \tag{20}$$

where $\boldsymbol{f^*_j} = \{f^*_{j,s,t}\}$ are its Fourier coefficients, our goal is to compute the Fourier coefficients $\boldsymbol{g^*} = \{g^*_{s,t}\}$ of their product $G = F_1 F_2 \cdots F_{k-1}$, such that:

$$G(\theta,\phi) = \sum_{s,t} g^*_{s,t} e^{i(s\theta+t\phi)}. \tag{21}$$

4.2.1 Triple Product. Let us first discuss the simplest triple product case (i.e., $k = 3$), in which we would like to compute the multiplication of two 2D Fourier series $G = F_1 \cdot F_2$. Specifically, we have:

$$\begin{aligned} G(\theta,\phi) &= F_1(\theta,\phi) \cdot F_2(\theta,\phi) \\ &= \left(\sum_{s_1,t_1} f^*_{1,s_1,t_1} e^{i(s_1\theta+t_1\phi)}\right) \cdot \left(\sum_{s_2,t_2} f^*_{2,s_2,t_2} e^{i(s_2\theta+t_2\phi)}\right) \\ &= \sum_{s_1,t_1,s_2,t_2} (f^*_{1,s_1,t_1} f^*_{2,s_2,t_2}) e^{i[(s_1+s_2)\theta+(t_1+t_2)\phi]}. \end{aligned} \tag{22}$$

By comparing the coefficients in Equ. 21 and Equ. 22, it is easy to know that the Fourier coefficients of the product $g^*_{s,t}$ could be computed as:

$$g^*_{s,t} = \sum_{\substack{s_1+s_2=s \\ t_1+t_2=t}} f^*_{1,s_1,t_1} \cdot f^*_{2,s_2,t_2}. \tag{23}$$

Notice that the above equation is exactly a 2D convolution of the Fourier coefficients and we could further write it in vector form:

$$\boldsymbol{g^*} = \boldsymbol{f^*_1} * \boldsymbol{f^*_2}, \tag{24}$$

where $*$ denotes the 2D convolution operator.

It is well known that the 2D convolution could be efficiently computed through FFT [Brigham 1988], which has a time complexity of only $O(n^2 \log n)$. Specifically, first, we transform the Fourier coefficients $\boldsymbol{f^*_1}$ and $\boldsymbol{f^*_2}$ into the frequency domain using 2D Discrete Fourier Transform (DFT). Then, we compute element-wise multiplication in the frequency domain. Finally, we obtain the Fourier coefficients of the product $\boldsymbol{g^*}$ by transforming back from the frequency domain using 2D Inverse Discrete Fourier Transform (IDFT). Both DFT and IDFT can be computed using FFT. Hence, the 2D convolution in Equ. 24 has an overall time complexity of $O(n^2 \log n)$.

4.2.2 Multiple Product. Similar to the triple product case, for general multiple product cases (i.e., $k > 3$), the Fourier coefficients of the product $\boldsymbol{g}^*$ could also be formulated as 2D convolutions of coefficients, which is:

$$\boldsymbol{g}^* = \boldsymbol{f}_1^* * \boldsymbol{f}_2^* * \cdots * \boldsymbol{f}_{\boldsymbol{k-1}}^*. \tag{25}$$

We could still use FFT to accelerate the above computation. However, care must be taken since the degree of Fourier series will increase after multiplication. For example, the multiplication of two degree n Fourier series will result in a degree $(2n-1)$ Fourier series. We cannot simply restrict intermediate results to degree n since that will introduce truncation errors. To ensure an accurate multiple product, a straight-forward way to successively apply the 2D convolution operator:

$$\boldsymbol{g}^* = \underbrace{\underbrace{[\underbrace{(\boldsymbol{f}_1^* * \boldsymbol{f}_2^*)}_{\text{deg. } 2n-1} * \boldsymbol{f}_3^*]}_{\text{deg. } 3n-2} * \cdots * \boldsymbol{f}_{\boldsymbol{k-1}}^*}_{\text{deg. } (k-1)(n-1)+1}. \tag{26}$$

In Equ. 26, the first convolution $\boldsymbol{f}_1^* * \boldsymbol{f}_2^*$ takes $O(n^2 \log n)$ time. Since the degree increases during the process, later convolutions will take more time, i.e., the j-th convolution takes $O\big(j^2 n^2 \log(jn)\big)$ time. Overall, it takes $O\big(k^3 n^2 \log(kn)\big)$ time.

Divide-and-conquer strategy. For better efficiency, we utilize a divide-and-conquer strategy instead of successively computing convolutions. Our key observation is that the convolution operator is associative, i.e., $(\boldsymbol{f}_1^* * \boldsymbol{f}_2^*) * \boldsymbol{f}_3^* = \boldsymbol{f}_1^* * (\boldsymbol{f}_2^* * \boldsymbol{f}_3^*)$, hence, the order in applying convolutions will not affect the final results. Specifically, we compute the multiple product in this way:

$$\boldsymbol{g}^* = \underbrace{\underbrace{[\underbrace{(\boldsymbol{f}_1^* * \boldsymbol{f}_2^*)}_{\text{deg. } 2n-1} \cdots \underbrace{(\boldsymbol{f}_{\lfloor \frac{k}{2} \rfloor - 1}^* * \boldsymbol{f}_{\lfloor \frac{k}{2} \rfloor}^*)}_{\text{deg. } 2n-1}]}_{\text{deg. } \lfloor \frac{k}{2} \rfloor (n-1)+1} * \underbrace{[\underbrace{(\boldsymbol{f}_{\lfloor \frac{k}{2} \rfloor + 1}^* * \boldsymbol{f}_{\lfloor \frac{k}{2} \rfloor + 2}^*)}_{\text{deg. 2n - 1}} \cdots \boldsymbol{f}_{k-1}^*]}_{\text{deg. } \lfloor \frac{k-1}{2} \rfloor (n-1)+1}}_{\text{deg. } (k-1)(n-1)+1} \tag{27}$$

By using the divide-and-conquer strategy, the time complexity of multiplying $(k-1)$ Fourier series of degree n can be expressed as a recursive formula:

$$T(n,k) = O(k^2 n^2 \log(kn)) + 2T(n, \frac{k}{2}). \tag{28}$$

Applying the master theorem for divide-and-conquer recurrences shows that the above divide-and-conquer strategy (Equ. 27) has an overall time complexity of $O(k^2 n^2 \log(kn))$, which is smaller than that of successive computing convolutions (Equ. 26).

4.3 Conversion back to SH space

After convolution in Fourier space, the next step is to convert the Fourier coefficients back into SH space. As mentioned in Sec. 4.2, the Fourier coefficients of the resulted product $\boldsymbol{g}^*$ have a degree of $(k-1)(n-1)+1$, but we only need SH coefficients $\boldsymbol{g}$ within order n. A straight-forward way would be first converting $\boldsymbol{g}^*$ into SH coefficients with full order $(k-1)(n-1)+1$ using the Fourier-to-SH conversion procedure described in Sec. 3.2.4, then truncate them by only leaving the lowest n orders. However, it would require $O(k^3 n^3)$ time due to the increase of degrees/orders.

Instead, for better efficiency, we could slightly modify the conversion procedure, to support directly converting a Fourier series to SH coefficients with a lower order. This is achieved by modifying the index range for summation in Equ. 19, which yields:

$$f_{l,m} = \sum_{\substack{-n' < s < n' \\ t = \pm m}} h_{l,m,s,t}^* g_{s,t}^*, \tag{29}$$

where $n' = (k-1)(n-1)+1$. The time complexity is reduced to $O(kn^3)$.

4.4 Complexity Analysis

4.4.1 Time Complexity. The first step, conversion from SH space to Fourier space, has a time complexity $O(kn^3)$, since $(k-1)$ spherical functions need to be converted and each takes $O(n^3)$ time. The second step (convolution in Fourier space) has a time complexity of $O(k^2 n^2 \log(kn))$ by using the divide-and-conquer strategy. The third step, conversion back to SH space, requires an additional $O(kn^3)$ time. Overall, the time complexity of our multiple product method is $O(kn^3 + k^2 n^2 \log(kn))$. For triple products, the time complexity would be $O(n^3 + n^2 \log(n))$ or simply $O(n^3)$.

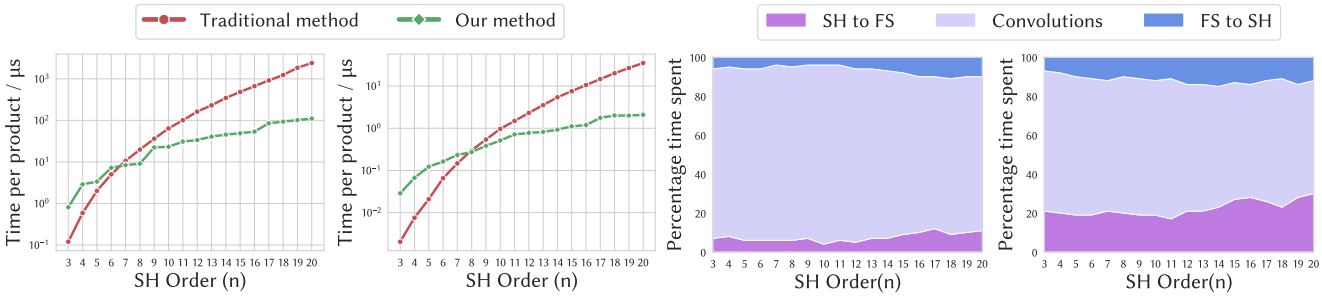
The above time complexity analysis provides a nice indication from a theoretical aspect. However, it may not be able to reflect the actual performance in practical applications. As we can see from the above analysis, the step of convolution in Fourier space has a relatively low time complexity, which is $O(k^2 n^2 \log(kn))$ (or $O(n^2 \log(n))$ for triple products), compared to the $O(kn^3)$ of the conversion steps. However, in practice, in both triple and multiple products, we still find that the convolution step is the performance bottleneck. There are two reasons. First, the constant factor involved in the convolution step is relatively large. Second, in practical applications of SH, the order n is usually small, i.e., $3 \le n \le 20$. We will further validate the efficiency of our method in Sec. 5.

4.4.2 Space Complexity. Our multiple product method requires $O(kn^3 + k^2 n^2)$ memory, including $O(kn^3)$ space for storing the precomputed tensors and $O(k^2 n^2)$ space for runtime storage. In contrast, the traditional brute-force method would require $O(n^{2k})$ memory to store the precomputed integral coefficients (Equ. 10), which would be prohibitively expensive. For triple products, we require $O(n^3)$ memory while the traditional method requires $O(n^5)$ memory to store the tripling coefficients (Equ. 5).

4.5 Recursive triple product approximation

Recall that traditional approaches use recursive triple products to approximately computing a multiple product (see Sec. 3.1.5 and Equ. 11). It has a time complexity of $O(kn^5)$, since it involves k times of triple products and each triple product costs $O(n^5)$ time. Its errors come from early truncation of intermediate results of triple products within SH order n.

Aside from our accurate solution of multiple product described in previous subsections, in applications where accuracy is not the top



(a) Performance comparison for triple products (CPU).

(b) Performance comparison for triple products (GPU).

(c) Profiling of our method for triple products (CPU).

(d) Profiling of our method for triple products (GPU).

Fig. 3. Comparisons between our method (Green Curve) and the traditional method (Red Curve) for triple products. On CPU, our method is faster than the traditional method when SH order $n \geq 7$. On GPU, our method is faster when SH order $n \geq 8$. By profiling, we found that the convolution in Fourier space is the bottleneck of our method on both CPU and GPU.

priority, we can also use such recursive triple product approximation to compute the multiple product, by only replacing the $O(n^5)$ traditional triple product method in the recursive approximation (Sec. 3.1.5) with our proposed $O(n^3)$ triple product method (Sec. 4.1). This reduces the overall time complexity from $O(kn^5)$ to $O(kn^3)$.

Note that our approximate method and the traditional approximate method have used the same recursive triple product approximation. They differ in computational efficiency, but are expected to produce the same results in theory.

5 EXPERIMENTS

We have performed extensive experiments to evaluate the effectiveness of our proposed method for triple and multiple products under various settings. We implement our method on both CPU and GPU. We implement the CPU version using C++14 and use the fftw 3.3.9 library for FFT computation. We implement the GPU version using CUDA 10.2 and use cuFFT library for FFT computation. The experiments are carried out on a PC with an Intel Xeon E5-2678 v3 CPU and an NVIDIA 2080 Ti GPU.

5.1 Performance comparison

5.1.1 Triple product. We compare the running performance of our proposed triple product method (Sec. 4.1) with traditional triple product method (Sec. 3.1.3), as shown in Fig. 3 (a-b). Our method starts to perform faster than the traditional method when $n \geq 7$ on CPU or $n \geq 8$ on GPU. For larger SH order n, our method runs significantly faster. For example, our method achieves a speedup of 3.0×, 9.9×, and 21.7× on CPU for $n = 10, 15$ and 20, respectively, and achieves a speedup of 1.9×, 8.9×, and 17.8× on GPU for $n = 10, 15$ and 20, respectively.

We further analyze the execution time of different steps in our method, including conversion from SH space to Fourier space (Sec. 3.2.3), convolution in Fourier space (Sec. 4.2), and conversion back to SH space (Sec. 4.3), as shown in Fig. 3 (c-d). While the convolution step has a lower time complexity (i.e., $O(n^2 \log n)$) than that of conversion steps (i.e., $O(n^3)$), it is still the bottleneck of the whole computation for all SH orders $3 \leq n \leq 20$. Nevertheless, its proportion in timing decreases when we increase SH order n, e.g., reduced from 87% ($n = 3$) to 79% ($n = 20$) on CPU or from 72% ($n = 3$) to 58% ($n = 20$) on GPU.

We also notice that our method achieves slightly less speedup on GPU than on CPU. This is because the bottleneck of our method–the convolution using FFT–is mainly dominated by random access instead of sequential access to the memory, which is less cache-friendly on GPU.

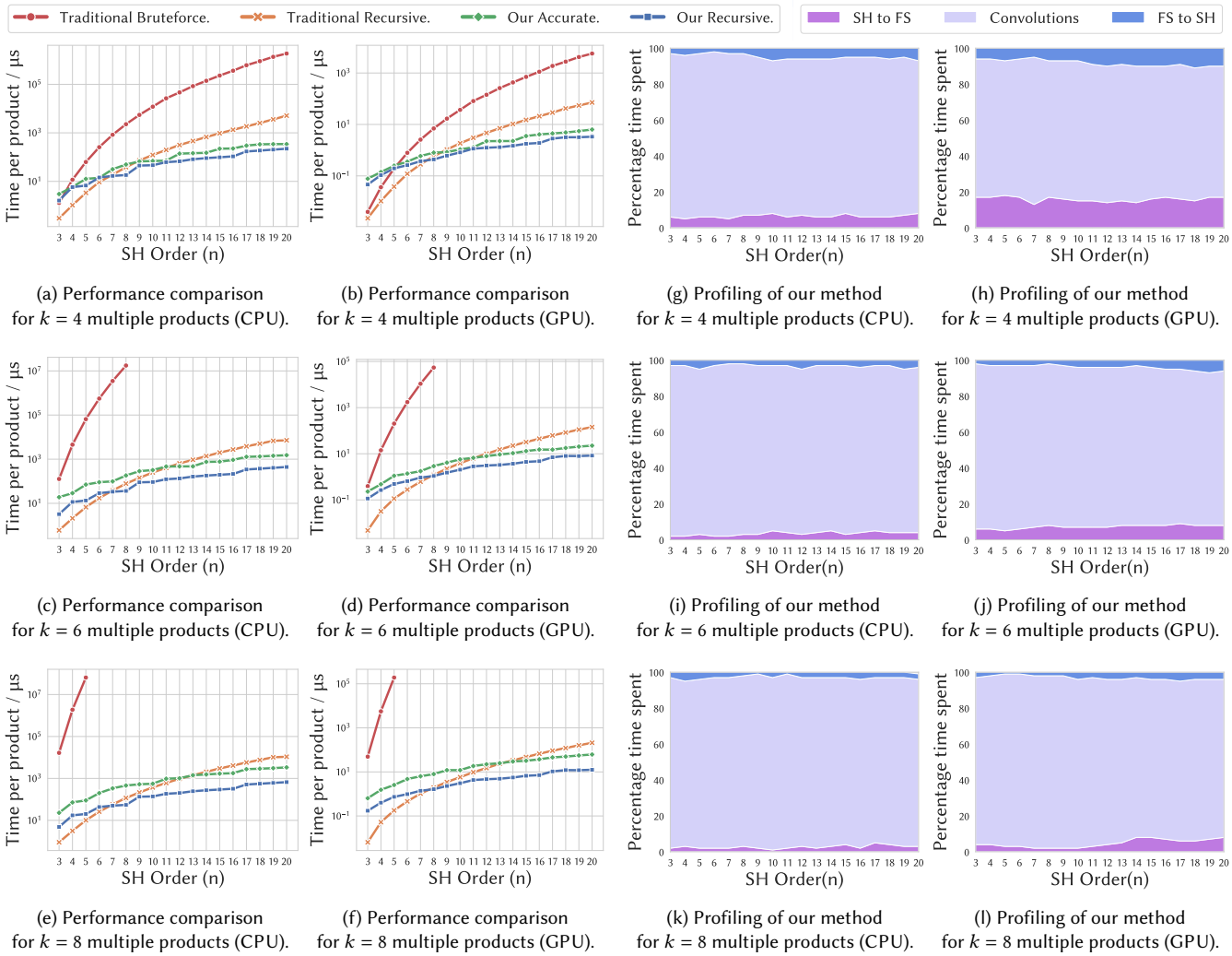
5.1.2 Multiple product. We compare the efficiency of four different methods in computing SH multiple products, including: our accurate method (Sec. 4.1), our approximate method using recursive triple products (Sec. 4.5), traditional bruteforce method (Sec. 3.1.4), and traditional approximate method using recursive triple products (Sec. 3.1.5), as shown in Fig. 4 (a-f). In our implementation of the the traditional bruteforce method, we only enumerate over the non-zero integral coefficients for better efficiency.

The traditional bruteforce method (red curve) is extremely inefficient, especially when the number of functions k is large (i.e., $k = 6$ or 8). This is not surprising since it has an extremely high time complexity, making it not applicable for practical use. Thanks to a much lower time complexity, our accurate method (green curve) runs orders of magnitude faster in the majority of settings, except the cases when n and k are both very small, e.g., $k = 4$ and $n \leq 5$. Notice that our accurate method scales well with both values of n and k: it is still able to run efficiently when n and k are very large.

As for the approximate methods, our approximate method (blue curve) starts to run faster than the traditional approximate method (orange curve) when SH order $n \geq 7$ on CPU or $n \geq 8$ on GPU for all values of k, and runs much faster when n is large. It is mostly consistent with the results of performance comparison conducted for triple products (see Sec. 5.1.1 and Fig. 3 (a-b)). This is because both methods use the same recursive triple product approximation.

It is important to note that our accurate method (green curve) runs even faster than the traditional approximate method (orange curve) when SH order n is large, e.g, when $n \geq 9$ for $k = 4$, $n \geq 12$ for $k = 6$, or $n \geq 14$ for $k = 8$.

We further analyze the time of the three different steps in our accurate multiple product method, as shown in Fig. 4 (g-l). The



(a) Performance comparison for $k = 4$ multiple products (CPU).
(b) Performance comparison for $k = 4$ multiple products (GPU).
(g) Profiling of our method for $k = 4$ multiple products (CPU).
(h) Profiling of our method for $k = 4$ multiple products (GPU).
(c) Performance comparison for $k = 6$ multiple products (CPU).
(d) Performance comparison for $k = 6$ multiple products (GPU).
(i) Profiling of our method for $k = 6$ multiple products (CPU).
(j) Profiling of our method for $k = 6$ multiple products (GPU).
(e) Performance comparison for $k = 8$ multiple products (CPU).
(f) Performance comparison for $k = 8$ multiple products (GPU).
(k) Profiling of our method for $k = 8$ multiple products (CPU).
(l) Profiling of our method for $k = 8$ multiple products (GPU).

Fig. 4. Performance comparisons between our accurate method (Green Curve), our recursive method (Blue Curve), traditional brute-force method (Red Curve) and traditional recursive method (Orange Curve) for multiple products. Our accurate method is orders of magnitude faster than the traditional brute-force method especially when k and n are large. Furthermore, our accurate method is even faster than the approximate method with high-order n. And our recursive method is faster than traditional recursive method for most cases. For each k and n, we profile our implementations on CPU and GPU. In all cases tested, the convolution in Fourier space step is still the bottleneck of our method.

convolution step is the bottleneck of the whole computation for all SH orders $3 \leq n \leq 20$ and for all $k = 4, 6$, and 8. Nevertheless, its proportion in time decreases when we increase SH order n. For example, for $k = 4$, it reduces from 91%/77% ($n = 3$) to 85%/73% ($n = 20$) on CPU/GPU, and for $k = 8$, it reduces from 95%/93% ($n = 3$) to 93%/88% ($n = 20$) on CPU/GPU.

5.2 Error analysis of approximate methods

In this subsection, we analyze and visualize the errors introduced in the approximate multiple product methods using recursive triple products. Note that our approximate method (Sec. 4.5) and the traditional approximate method (Sec. 3.1.5) only differ in computational efficiency, but will produce the same results.

5.2.1 Analysis on Band-limited Inputs. First, we measure the relative L2 error of the SH coefficients of the multiple product results. The SH coefficients of each input band-limited spherical function are randomly and uniformly sampled in $[-1, 1]$. For each value of k, we randomly generate 100,000 groups of band-limited inputs, and further compute the SH coefficients of their multiple products under each value of SH order n. The error values are given in Table 1. We can see that the error grows quickly with the number of functions k involved in multiplication, e.g., from about 20% for $k = 4$ to nearly 90% for $k = 8$. This is because the truncation errors in intermediate triple product will accumulate when k increases. We also find that

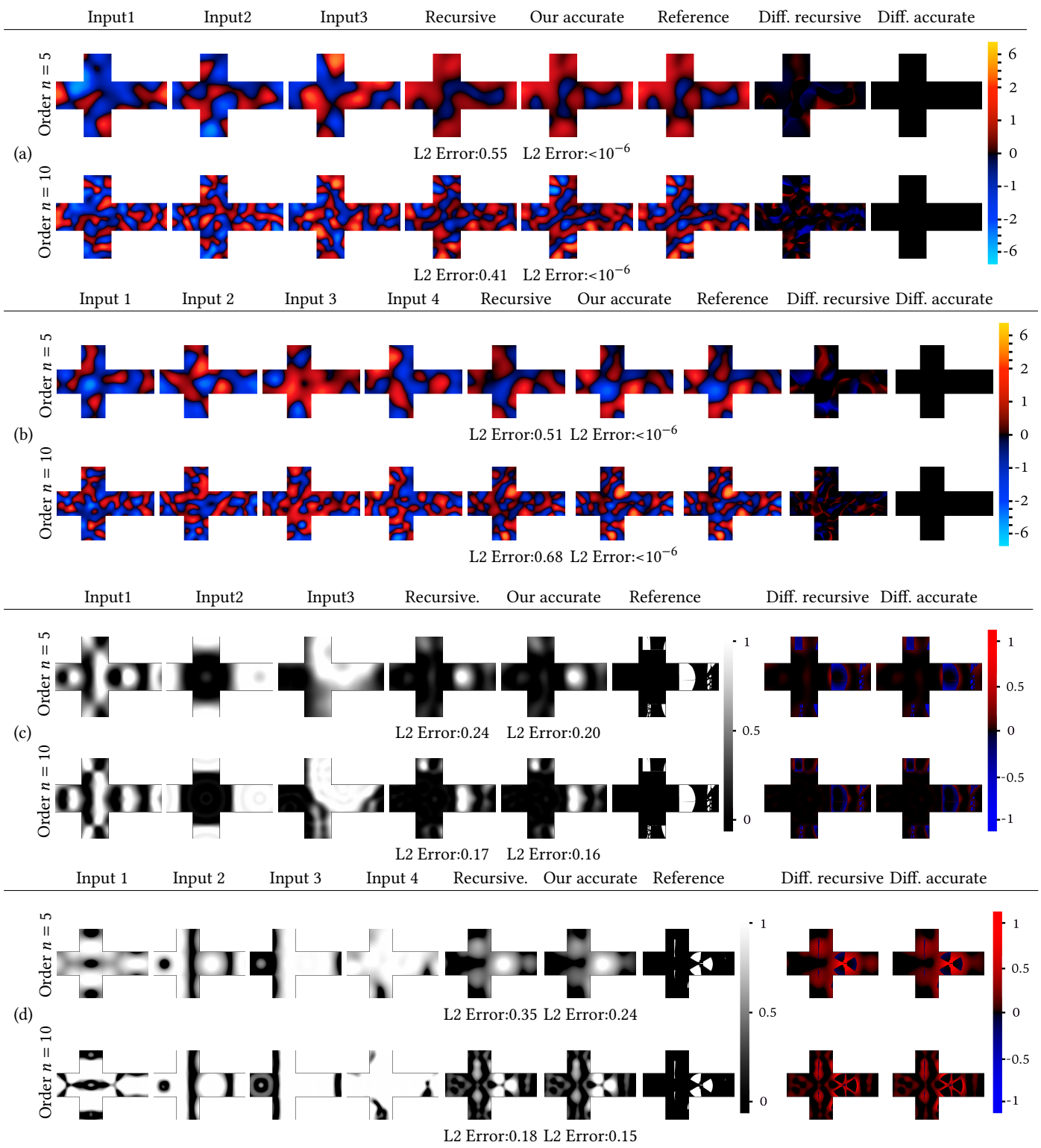
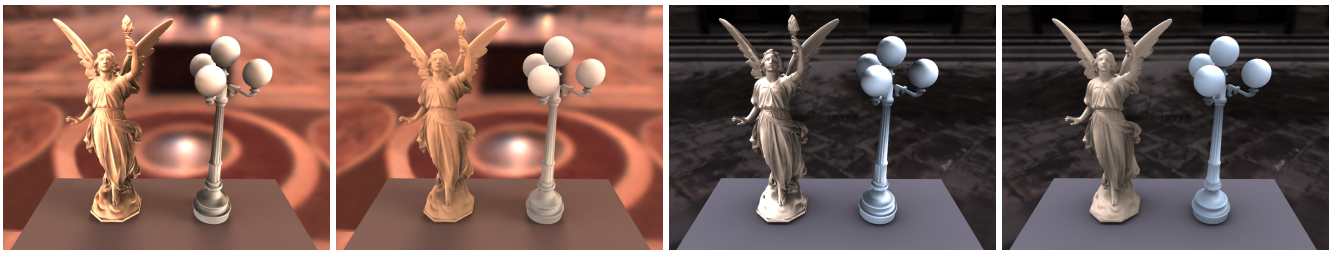
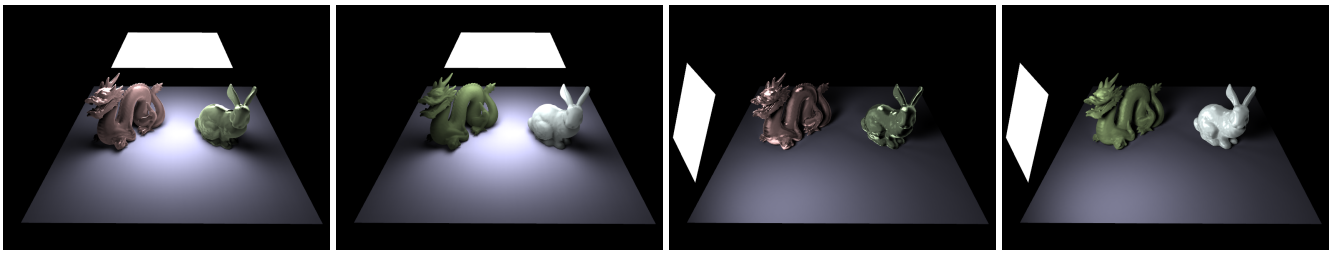


Fig. 5. Visualization of multiple product results. For each example, from left to right, we show the input band-limited spherical functions, the multiple product results generated by the recursive method and by our accurate method, respectively, the ground truth, and difference images.



(a) Rendered with SH order $n = 10$. Frame rates: traditional method 14.9 FPS / ours 26.4 FPS



(b) Rendered with SH order $n = 15$. Frame rates: traditional method 3.0 FPS / ours 25.5 FPS

Fig. 6. Results of glossy relighting using PRT.

the error values do not change a lot for varying SH order $3 \leq n \leq 20$ under the same value of k. Besides, we also measure the relative L2 errors of our accurate method for comparison, which are below 10^{-6} under all settings.

We randomly pick two groups of examples from the above experiments, and visualize the reconstructed input and the resulted spherical functions from their SH coefficients, as shown in Fig. 5 (a-b). Each group provides two examples under $n = 5$ and $n = 10$, respectively. We could find that the multiple product result generated the approximate method exhibit large visible differences from the ground truth. In comparison, our accurate method produces almost the same results as ground truth.

5.2.2 Analysis on Non-band-limited Inputs. In Fig. 5 (c-d), we further visualize examples of multiple product results from real scenes, where each input spherical function is a self-visibility map or a visibility map from object occlusion fields [Zhou et al. 2005]. We project each input spherical function into SH space and compute the multiple products using our approximate method and our accurate method, respectively. The relative L2 errors of the resulting multiple product are computed against the non-band-limited ground truth. Since our accurate method is designed to obtain accurate results only for band-limited cases, it is not surprising it still exhibits differences with the non-band-limited ground truth, however, its error is smaller than those of the recursive triple product approximation in all examples.

5.3 Rendering applications

To demonstrate the merits of our triple and multiple product methods in practice, we further implement two SH based rendering applications, including glossy relighting [Ng et al. 2004; Sloan et al. 2002] and shadow fields [Zhou et al. 2005]. Adapting our method into these existing rendering applications is very simple. By replacing the traditional triple/multiple product methods by our proposed ones, we can easily improve running performance.

5.3.1 Glossy Relighting using PRT. The direct illumination of a static scene could be formulated as:

$$L_o(x, \omega_o) = \int L(x, \omega_i) V(x, \omega_i) \rho(\omega_i, \omega_o) \mathrm{d}\omega_i, \tag{30}$$

where L_o is the exit radiance at position x, ω_o is the view direction, ω_i is the light direction and ρ is the BRDF baked with the cosine term. Please note that, we do not combine visibility and BRDF together in order to support dynamic lighting and materials.

Using the PRT framework, we represent each term in the above equation using SH basis functions. Specifically, the lighting and visibility function could be approximated as: $L(x, \omega_i) \simeq \sum_{j=1}^{n^2} l_j y_j(\omega_i)$, $V(x, \omega_i) \simeq \sum_{j=1}^{n^2} v_j y_j(\omega_i)$. By tabulatingthe BRDF in terms of view direction ω_o [Kautz et al. 2002; Lehtinen and Kautz 2003], BRDF could also be projected to SH space as: $\rho(\omega_i, \omega_o) \simeq \sum_{j=1}^{n^2} \rho_j y_j(\omega_i)$. After that, the rendering integral in Equ. 30 naturally becomes computing a triple product integral in SH space [Ng et al. 2004].

We have tested three scenes with dynamic lighting and dynamic BRDFs. Both environment lights and local area lights [Wang and Ramamoorthi 2018] are tested. Results are given Fig. 1 (a) and Fig. 6. The performance statistics are given in Table 2. Notice that our triple product method is able to achieve a speedup of about 1.8× using SH order $n = 10$, and a significant speedup over 8× using SH order $n = 15$.

5.3.2 Shadow fields. Shadow fields [Zhou et al. 2005] have extended PRT to handle dynamic scenes. It supports moving and rotating occluders by precomputing an object occlusion field (OOF) for each occluder. At run-time, the visibility map at each shading point x

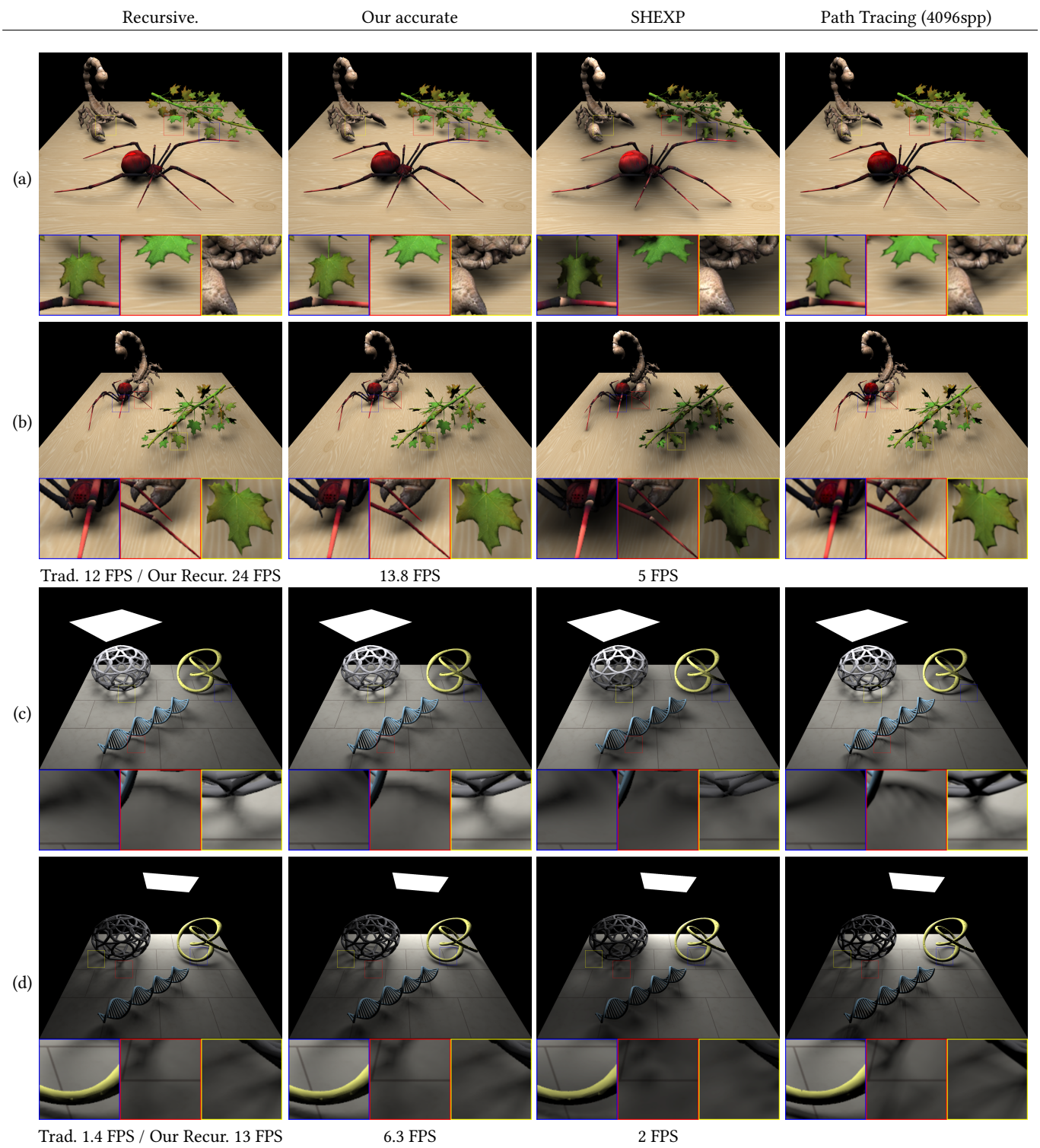


Fig. 7. Results of shadow fields compared with SHEXP and path tracing. (a-b) are rendered with SH order $n = 10$. (c-d) are rendered with SH order $n = 15$. Compared with SHEXP, our results are closer to the ground truth generated by path tracing.

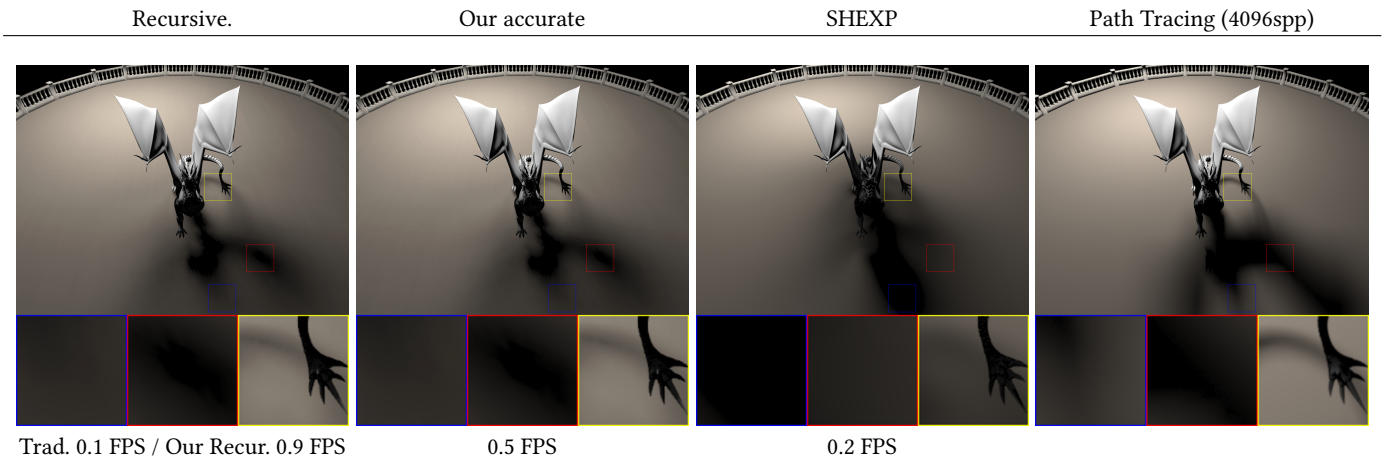


Fig. 8. Results of shadow fields compared with SHEXP and path tracing tested with SH order $n = 15$. Even for this complex scene with more than 30k vertices, our method is still more efficient than SHEXP and produces better results.

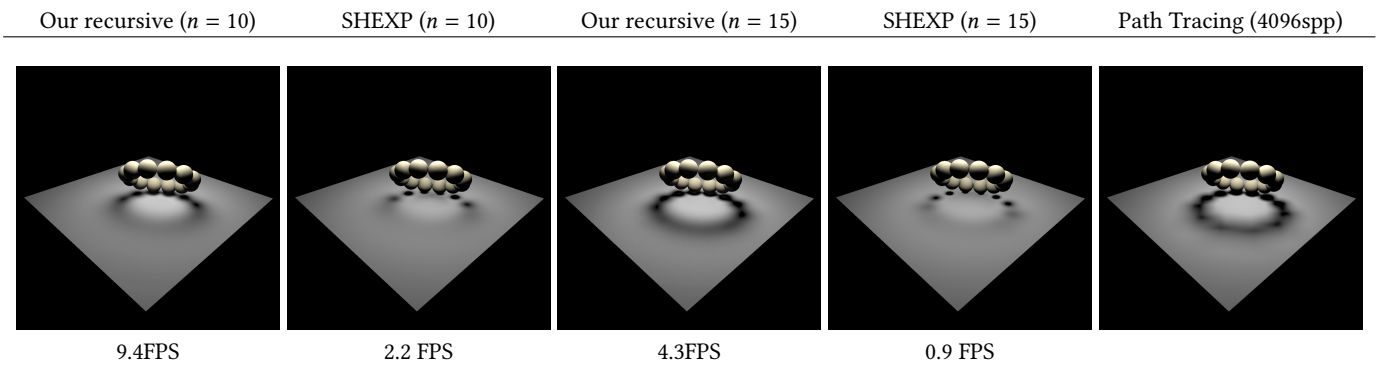


Fig. 9. Comparison between our method and SHEXP on a scene with only sphere occluders. Our method performs consistently faster than SHEXP on different SH orders. And the results of our method are much closer to the ground truth.

Table 1. Relative L2 error of the approximate multiple product method using recursive triple products.

SH Order	3	4	5	6	7	8	9	10	12	14	16	18	20
Our accurate, $k = 4/5/6/7/8$	$<10^{-6}$	$<10^{-6}$	$<10^{-6}$	$<10^{-6}$	$<10^{-6}$	$<10^{-6}$	$<10^{-6}$	$<10^{-6}$	$<10^{-6}$	$<10^{-6}$	$<10^{-6}$	$<10^{-6}$	$<10^{-6}$
Trad. / our recursive, $k = 4$	0.21	0.22	0.23	0.23	0.23	0.24	0.23	0.24	0.25	0.24	0.25	0.24	0.24
Trad. / our recursive, $k = 5$	0.35	0.36	0.35	0.35	0.34	0.35	0.35	0.35	0.35	0.35	0.34	0.34	0.35
Trad. / our recursive, $k = 6$	0.49	0.51	0.52	0.52	0.52	0.52	0.51	0.51	0.51	0.51	0.51	0.51	0.52
Trad. / our recursive, $k = 7$	0.67	0.65	0.66	0.68	0.68	0.67	0.69	0.67	0.68	0.69	0.69	0.67	0.68
Trad. / our recursive, $k = 8$	0.87	0.88	0.88	0.89	0.89	0.89	0.89	0.88	0.89	0.89	0.89	0.89	0.88

could be obtained through a multiple product:

$$V(x,\omega) = V_{\text{self}}(x,\omega) \cdot V_1(x,\omega) \cdots V_m(x,\omega), \tag{31}$$

where $V_{\text{self}}(x,\omega)$ denotes the self visibility map and $V_j(x,\omega)$ denotes the visibility map viewing towards the j-th occluder. By expressing each term in the above equation using SH coefficients, it naturally becomes a multiple product in SH space.

We have tested various scenes with different complexity, i.e., vertex number ranging from 14k to 335k, as shown in Fig. 1 (b) and Fig. 7 to 9. The performance statistics are given in Table 2. Three methods are evaluated for computing the multiple product, including our accurate method, our approximate method (Sec. 4.5), and the traditional approximate method (Sec. 3.1.5). We emphasize again that the latter two methods only differ in computational efficiency but will produce the same results, so that we do not provide duplicated rendering results for them. Compared with the traditional approximate method, both our accurate and approximate methods run faster. Specifically, our accurate method runs 1.15× faster ($n = 10$)

Table 2. Performance statistics. For each scene, we provide the name of the scene, the number of vertices, SH order (n), rendering application (PRT or Shadow Field), the performance data of the traditional method, our accurate method, and our recursive method, respectively, and the speedup on GPU we achieved. For each method, we show the frame rates and time percentage of SH products in the entire shading pass.

Scene	Figure	#vert	SH order	App.	Trad. (FPS/Time perc.)	Our acc. (FPS/Time perc.)	Our recur. (FPS/Time perc.)	Speedup
Lucy	Fig.6 (a)	69994	10	PRT	14.9/92%	26.4/85%	N/A	1.8×
Dragon & Bunny	Fig.6 (b)	36613	15	PRT	3.0/98%	25.5/87%	N/A	8.5×
Plants & Teapot	Fig.1 (a)	34213	15	PRT	3.2/98%	26/87%	N/A	8.2×
Spider & Scorpion	Fig.7(a)	30238	10	SF	12/96%	13.8/93%	24/86%	1.2×/2.0×
Ball & DNA	Fig.7(b)	24410	15	SF	1.4/99%	6.3/96%	13/92%	4.5×/9.3×
Chess board	Fig.1 (b)	14018	15	SF	2.6/98%	11.2/93%	24/86%	4.3×/8.9×
Dragon in Hall	Fig.8	335823	15	SF	0.1/98%	0.5/92%	0.9/88%	4.5×/9.2×
Spheres	Fig.9	73753	10	SF	4.9/95%	5.7/94%	9.4/91%	1.2×/1.9×
Spheres	Fig.9	73753	15	SF	0.4/98%	2.1/96%	4.3/91%	4.6×/9.1×

or 4.5× faster ($n = 15$), and our approximate method runs 2× faster ($n = 10$) or 9.3× faster ($n = 15$). Notice that there is not much visual difference between the rendered results of our accurate and approximate methods. Hence, in applications where the accuracy is not the top priority, we could still use the recursive triple product approximation to compute the multiple product for better efficiency.

We also provide rendered images generated by SH Exponentiation (SHEXP) [Ren et al. 2006] for comparison. The main idea of SHEXP to compute multiple product is to first transform each input function to log space, then perform summation in log space, and finally transform back through an exp operator. However, a SH log operator would take $O(n^6)$ time for an arbitrary spherical function. For better efficiency for rendering, they further approximate the scene geometry with a set of spheres and the SH log results of visibility maps towards the spheres are precomputed. We found that SHEXP exhibits a large visual difference from the ground truth. There are two main reasons. First, the geometries of our tested scenes are relatively complex, i.e., the antennae of spiders (Figure 7) are thin and cannot be well approximated using set of spheres. Second, the SH exponentiation and SH logarithm operators are crude approximations, especially for higher-order SHs. Even for scenes containing only sphere occluders (Figure 9), where no geometric approximations are involved, the visual differences are still large. In contrast, our methods, including both accurate and approximate versions, have a relatively small difference compared to the path traced ground truth.

Note that the rendering results may exhibit some level of ringing effects. This is a shared problem using SHs and could be alleviated through a post-process [Sloan 2008] after the triple/multiple products are computed.

6 DISCUSSIONS AND CONCLUSION

Conclusion. In this paper, we have proposed an accurate and fast method for SH triple and multiple products. The key idea is to transform SH basis functions to the Fourier space, perform efficient multiplications using FFT, then transform the result back. Theoretically, our method has the best time complexity. And in practice, we show clear benefits of using our method on both CPU and GPU, on medium to high SH orders with different numbers of multiple product components. We integrate our method into PRT applications, such as glossy relighting and shadow fields, to demonstrate that our method is able to reach real-time performance even at an SH order of 15.

Scope. We believe that our method has opened up possibilities for the use of moderately high orders of spherical harmonics in rendering applications. Moreover, despite the seemingly complex theoretical derivation, our new method could be implemented with just a few lines of code, and is orthogonal to any other parts in the rendering engine. Therefore, our method is ready to be adopted by the industry.

SH is a powerful and fundamental tool widely used across different research fields. For example, in computer graphics, it has always been a common choice to use SHs to represent light in probe based global illumination. In computer vision, SHs are pervasively used for illumination estimation in recognition and reconstruction tasks. SHs have also been used in CNN layer designs, and have shown effectiveness in point cloud classification and segmentation [Poulenard et al. 2019] and shape classification [Cohen et al. 2018; Esteves et al. 2018]. Recently, SHs have also been applied to neural rendering, e.g., using SHs as the representation of radiance [Yu et al. 2021] for real-time rendering of Neural Radiance Fields (NeRFs) [Mildenhall et al. 2020], or modulating feature values using SHs for better view coherence [Thies et al. 2019]. Since multiple products are fundamental operators of SHs, our method is expected to benefit many applications beyond PRT.

Limitations and future work. As elaborated above, our method has a great advantage on the time complexity compared to the traditional method. However, due to the large constant factor in the time complexity of FFT, our current implementation can be slower than the traditional method at low SH orders n with a few components of multiplication k. An interesting direction is to further optimize the performance for low SH orders, possibly by specially optimized FFT implementation tailored to our problem.

The recursive triple product approximation would still be favored in applications where the accuracy is not the top priority, hence, it is also interesting to efficiently estimate or bound the truncation

errors introduced in the triple product approximation. A possible way to achieve that is through analyzing the distributions of energy (i.e., coefficient values) along bands.

We also hope our work would inspire researchers to explore the Fourier transform idea deeper to accelerate other complex computations in rendering. For example, as a subset of spherical harmonics, triple and multiple product of zonal harmonics [Sloan et al. 2005] might be accelerated even further with their own properties. It is also of potential value to perform fast high-dimensional multiple product integrals (such as the rendering equation with a 4D BRDF inside).

ACKNOWLEDGMENT

We would like to thank the reviewers for their constructive comments and suggestions. This work is supported by the National Natural Science Foundation of China (Project Numbers: 61822204, 61521002, 61863031) and a research grant from the Beijing Higher Institution Engineering Research Center. Ling-Qi Yan is supported by gift funds from Adobe, Dimension 5 and XVerse.

REFERENCES

E Oran Brigham. 1988. *The fast Fourier transform and its applications*. Prentice-Hall, Inc.

Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. 2018. Spherical CNNs. In *International Conference on Learning Representations*.

Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. 2018. Learning so(3) equivariant representations with spherical cnns. In *Proceedings of the European Conference on Computer Vision*. 52–68.

Xuejun Hao and Amitabh Varshney. 2004. Real-time rendering of translucent meshes. *ACM Transactions on Graphics* 23, 2 (2004), 120–142.

DJ Hofsommer and ML Potters. 1960. Table of Fourier coefficients of associated Legendre functions. *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen: Series A: Mathematical Sciences* 63, 5 (1960), 460–480.

Teturo Inui, Yukito Tanabe, and Yositaka Onodera. 1999. *Group theory and its applications in physics*. Vol. 78. Springer Verlag.

Jan Kautz, Peter-Pike Sloan, and Jaakko Lehtinen. 2005. Precomputed Radiance Transfer: Theory and Practice. In *ACM SIGGRAPH 2005 Courses*. 1–es.

Jan Kautz, John Snyder, and Peter-Pike Sloan. 2002. Fast Arbitrary BRDF Shading for Low-Frequency Lighting Using Spherical Harmonics. *Rendering Techniques* 2, 291-296 (2002), 1.

Jaakko Lehtinen. 2007. A framework for precomputed and captured light transport. *ACM Transactions on Graphics* 26, 4 (2007), 13–es.

Jaakko Lehtinen and Jan Kautz. 2003. Matrix radiance transfer. In *Proceedings of the 2003 symposium on Interactive 3D graphics*. 59–64.

Christian Lessig, Mathieu Desbrun, and Eugene Fiume. 2014. A constructive theory of sampling for image synthesis using reproducing kernel bases. *ACM Transactions on Graphics* 33, 4 (2014), 1–14.

Xinguo Liu, Peter-Pike J Sloan, Heung-Yeung Shum, and John Snyder. 2004. All-Frequency Precomputed Radiance Transfer for Glossy Objects. *Rendering Techniques* 2004 (2004).

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*. 405–421.

Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. 2003. All-frequency shadows using non-linear wavelet lighting approximation. In *ACM SIGGRAPH 2003 Papers*. 376–381.

Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. 2004. Triple product wavelet integrals for all-frequency relighting. In *ACM SIGGRAPH 2004 Papers*. 477–487.

Derek Nowrouzezahrai, Patricio D. Simari, and Eugene Fiume. 2012. Sparse zonal harmonic factorization for efficient SH rotation. *ACM Transactions on Graphics* 31, 3 (2012), 23:1–23:9.

Jacopo Pantaleoni, Luca Fascione, Martin Hill, and Timo Aila. 2010. Pantaray: fast ray-traced occlusion caching of massive scenes. *ACM Transactions on Graphics* 29, 4 (2010), 1–10.

Adrien Poulenard, Marie-Julie Rakotosaona, Yann Ponty, and Maks Ovsjanikov. 2019. Effective rotation-invariant point cnn with spherical harmonics kernels. In *2019 International Conference on 3D Vision*. 47–56.

Ravi Ramamoorthi. 2009. Precomputation-based rendering. *Foundations and Trends in Computer Graphics and Vision* 3, 4 (2009), 281–369.

Ravi Ramamoorthi and Pat Hanrahan. 2001. An Efficient Representation for Irradiance Environment Maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. 497–500.

Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. 2006. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. In *ACM SIGGRAPH 2006 Papers*. 977–986.

Nathanaël Schaeffer. 2013. Efficient spherical harmonic transforms aimed at pseudospectral numerical simulations. *Geochemistry, Geophysics, Geosystems* 14, 3 (2013), 751–758.

Peter-Pike Sloan. 2008. Stupid spherical harmonics (sh) tricks. In *Game developers conference*, Vol. 9. 42.

Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. 2003a. Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics* 22, 3 (2003), 382–391.

Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 527–536.

Peter-Pike Sloan, Xinguo Liu, Heung-Yeung Shum, and John Snyder. 2003b. Bi-scale radiance transfer. *ACM Transactions on Graphics* 22, 3 (2003), 370–375.

Peter-Pike Sloan, Ben Luna, and John Snyder. 2005. Local, deformable precomputed radiance transfer. *ACM Transactions on Graphics* 24, 3 (2005), 1216–1224.

Nico Sneeuw and Richard Bun. 1996. Global spherical harmonic computation by two-dimensional Fourier methods. *Journal of Geodesy* 70, 4 (1996), 224–232.

John Snyder. 2006. Code generation and factoring for fast evaluation of low-order spherical harmonic products and squares. *MSR-TR-2006-53* (May 2006), 9.

Bo Sun and Ravi Ramamoorthi. 2009. Affine double-and triple-product wavelet integrals for rendering. *Transactions on Graphics* 28, 2 (2009), 1–17.

Weifeng Sun and Amar Mukherjee. 2006. Generalized Wavelet Product Integral for Rendering Dynamic Glossy Objects. In *ACM SIGGRAPH 2006 Papers*. 955–966.

Justus Thies, Michael Zollhöfer, and Matthias Nießner. 2019. Deferred Neural Rendering: Image Synthesis Using Neural Textures. *ACM Transactions on Graphics*. 38, 4 (2019), 12 pages.

Karvel K Thornber and David W Jacobs. 2006. Broadened-specular reflection and linear subspaces for object recognition. US Patent 7,058,217.

Yu-Ting Tsai and Zen-Chung Shih. 2006. All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. *ACM Transactions on Graphics* 25, 3 (2006), 967–976.

Jingwen Wang and Ravi Ramamoorthi. 2018. Analytic Spherical Harmonic Coefficients for Polygonal Area Lights. *ACM Transactions on Graphics* 37, 4 (2018).

Jiaping Wang, Peiran Ren, Minmin Gong, John Snyder, and Baining Guo. 2009. All-frequency rendering of dynamic, spatially-varying reflectance. In *ACM SIGGRAPH Asia 2009 papers*. 1–10.

Jiaping Wang, Kun Xu, Kun Zhou, Stephen Lin, Shimin Hu, and Baining Guo. 2006. Spherical Harmonics Scaling. *The Visual Computer* 22 (Sept 2006), 713–720.

Lifan Wu, Guangyan Cai, Shuang Zhao, and Ravi Ramamoorthi. 2020. Analytic spherical harmonic gradients for real-time rendering with many polygonal area lights. *ACM Transactions on Graphics* 39, 4 (2020), 134–1.

Kun Xu, Yue Gao, Yong Li, Tao Ju, and Shi-Min Hu. 2007. Real-time homogenous translucent material editing. In *Computer Graphics Forum*, Vol. 26. Wiley Online Library, 545–552.

Kun Xu, Li-Qian Ma, Bo Ren, Rui Wang, and Shi-Min Hu. 2011. Interactive hair rendering and appearance editing under environment lighting. *ACM Transactions on Graphics* 30, 6 (2011), 1–10.

Kun Xu, Wei-Lun Sun, Zhao Dong, Dan-Yong Zhao, Run-Dong Wu, and Shi-Min Hu. 2013. Anisotropic Spherical Gaussians. *ACM Transactions on Graphics* 32, 6 (2013), 209:1–209:11.

Tomoya Yamaguchi, Tatsuya Yatagawa, Yusuke Tokuyoshi, and Shigeo Morishima. 2020. Real-time rendering of layered materials with anisotropic normal distributions. *Computational Visual Media* 6, 1 (2020), 29–36.

Ling-Qi Yan, Yahan Zhou, Kun Xu, and Rui Wang. 2012. Accurate Translucent Material Rendering under Spherical Gaussian Lights. *Computer Graphics Forum* 31, 7 (2012), 2267––2276.

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. Plenoctrees for real-time rendering of neural radiance fields. *arXiv preprint arXiv:2103.14024* (2021).

Kun Zhou, Yaohua Hu, Stephen Lin, Baining Guo, and Heung-Yeung Shum. 2005. Precomputed shadow fields for dynamic scenes. In *ACM SIGGRAPH 2005 Papers*. 1196–1201.