The Chebyshev Fast Gauss and Nonuniform Fast Fourier Transforms and their application to the evaluation of distributed heat potentials*

Shravan K. Veerapaneni † George Biros‡

March 14, 2007

Abstract

We present a method for the fast and accurate computation of distributed (or volume) heat potentials in two dimensions. The distributed source is assumed to be given in terms of piecewise space-time Chebyshev polynomials. We discretize uniformly in time, whereas in space the polynomials are defined on the leaf nodes of a quadtree data structure. The quadtree can vary at each time step. We combine a product integration rule with fast algorithms (fast heat potentials, nonuniform FFT, fast Gauss transform) to obtain a high-order accurate method with optimal complexity. If the input contains q^3 polynomial coefficients at M leaf nodes and N time steps, our method requires $\mathcal{O}(q^3 MN \log M)$ work to evaluate the heat potential at arbitrary MN space-time target locations. The overall convergence rate of the method is of order q. We present numerical experiments for $q = 4, 8$, and 16, and we verify the theoretical convergence rate of the method. When the solution is sufficiently smooth, the 16th-order variant results in significant computational savings, even in the case in which we require only a few digits of accuracy.

1 Introduction

In this article, we present a fast, high-order algorithm for the solution of the heat equation with a distributed force:

$$\frac{\partial u}{\partial t} = \triangle u + f(\mathbf{x}, t) \quad \text{in} \quad \Omega, \quad \text{where} \quad \Omega = [0,1]^2, \quad \text{and} \quad t \in (0, T], \tag{1}$$

$$u(\mathbf{x}, 0) = 0 \quad \text{and} \quad u(\Gamma, t) = 0, \tag{2}$$

where Γ is the boundary of Ω. Using potential theory, the solution $u(\mathbf{x}, t)$ can be written as

$$u(\mathbf{x}, t) = \int_0^t \int_\Omega G(\mathbf{x}, t; \mathbf{y}, \tau) f(\mathbf{y}, \tau) dy \, d\tau, \tag{3}$$

where $G(\mathbf{x}, t; \mathbf{y}, \tau)$ is the Green's function. In our method we assume that the input f is given as a set of N quadtrees $\{Q_0, Q_1, ..., Q_{N-1}\}$ corresponding to times $\{t_0, t_1, ..., t_{N-1}\}$. The leaf nodes of each quadtree contain q^3 space-time Chebyshev polynomial coefficients of f. The evaluation points (also referred to as "*targets*") are assumed to belong to a domain ω that is embedded in Ω.

*This work was supported by the U.S. Department of Energy under grant DE-FG02-04ER25646, and the U.S. National Science Foundation grants CCF-0427985, CNS-0540372, and DMS-0612578.

†Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104, USA (shravan@seas.upenn.edu)

‡Departments of Mechanical Engineering and Applied Mechanics, and Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104, USA (biros@seas.upenn.edu)

A direct computation of (3) using standard numerical quadratures is expensive and inaccurate. It is inaccurate because the integration kernel is sharply peaked near the evaluation point. It is expensive because its evaluation requires $\mathcal{O}(NM)$ work for every $\mathbf{x}$, where N and M are the number of quadrature points in time and space respectively. Therefore, the cost of evaluation at N, M temporal and spatial points would be $\mathcal{O}(N^2M^2)$.

1.1 Time domain decomposition: far and local part

Our scheme is based on an algorithm, first proposed by Greengard and Strain [12], that eliminates the history dependence of the time integral in (3) by combining two (analytically) equivalent series expansions of the heat kernel, a Fourier based and a method-of-images based :

$$G(\mathbf{x},t;\mathbf{y},\tau) = 4\sum_{\mathbf{n}\in\mathbf{Z}^2_+}\prod_{i=1}^{2} e^{-|\mathbf{n}|^2\pi^2(t-\tau)}\sin(n_i\pi x_i)\sin(n_i\pi y_i), \quad (\mathbf{n}=\{n_1,n_2\}, |\mathbf{n}| = n_1+n_2) \tag{4}$$

$$= \frac{1}{4\pi(t-\tau)}\sum_{\mathbf{n}\in\mathbf{Z}^2}\sum_{\sigma_1=\pm1}\sum_{\sigma_2=\pm1} e^{-\frac{(x_1-\sigma_1 y_1-2n_1)^2+(x_2-\sigma_2 y_2-2n_2)^2}{4(t-\tau)}}. \tag{5}$$

The convergence rate of the two expansions depends on $|t-\tau|$. At a time τ close to the evaluation time t, the method-of-images expansion (5) converges faster than the Fourier one. At distant times the opposite is true, the Fourier expansion (4) converges faster. This fact motivates a splitting of the volume potential (3) into a far and a local part:

$$u = u_F + u_L = \int_0^{t-\delta}\int_\Omega Gf + \int_{t-\delta}^{t}\int_\Omega Gf, \tag{6}$$

where δ is a small parameter. The Fourier expansion is used to compute $u_F(\mathbf{x},t)$ and the method of images expansion for $u_L(\mathbf{x},t)$. For τ in $(0,t-\delta)$, the truncation error for the p-term Fourier series expansion of G is exponentially decaying with increasing p. If we assume that all target points are sufficiently far from the boundary of the domain Ω, then the terms with $|\mathbf{n}| > 0$ in the local expansion of the Green's function are decaying exponentially within the interval $(t-\delta,t)$.[1]

Far Part. Upon truncation of the Fourier expansion, the far part can be approximated by

$$u_F(\mathbf{x},t) \approx 4\sum_{n_2=1}^{p}\sum_{n_1=1}^{p} C_{\mathbf{n}}(t,\delta)\sin(n_1\pi x_1)\sin(n_2\pi x_2), \tag{7}$$

$$\text{where} \quad C_{\mathbf{n}}(t,\delta) = \int_0^{t-\delta} e^{-|\mathbf{n}|^2\pi^2(t-\tau)}\hat{f}_{\mathbf{n}}(\tau)\,d\tau, \tag{8}$$

$$\hat{f}_{\mathbf{n}}(\tau) = \int_\Omega f(\mathbf{y},\tau)\sin(n_1\pi y_1)\sin(n_2\pi y_2)\,d\mathbf{y}. \tag{9}$$

The following recurrence is the key feature of Greengard and Strain's algorithm:

$$C_{\mathbf{n}}(t+\triangle t,\delta) = e^{-|\mathbf{n}|^2\pi^2\triangle t}C_{\mathbf{n}}(t,\delta) + U_{\mathbf{n}}[\hat{f}], \tag{10}$$

$$\text{where} \quad U_{\mathbf{n}}[\hat{f}] = \int_{t-\delta}^{t+\triangle t-\delta} e^{-|\mathbf{n}|^2\pi^2(t+\triangle t-\tau)}\hat{f}_{\mathbf{n}}(\tau)\,d\tau, \tag{11}$$

$\triangle t$ is the size of time step. At each time step, the far part computation involves the evaluation of the discrete sum (7) and the update of the coefficients $C_{\mathbf{n}}$ using (10).

[1] See Section (5) for a discussion on how to choose p and δ. The algorithm can be extended to the case in which the evaluation points are near the boundary of Ω, but the details are tedious without offering further insight.

Local Part. Upon truncating the method-of-images expansion the local part can be approximated bye

$$u_L(\mathbf{x},t) \approx \int_{t-\delta}^{t} \int_{\Omega} \frac{e^{-\frac{|\mathbf{x}-\mathbf{y}|^2}{4(t-\tau)}}}{4\pi(t-\tau)} f(\mathbf{y},\tau)\, d\mathbf{y}\, d\tau. \tag{12}$$

We can easily show that the integrand in (12) is a smooth function in τ. Therefore, we can use any qth-order quadrature rule with nodes and weights $\{\tau_k, w_k\}_{k=1}^{q}$ (we discuss our choice in Section 5). Applying the quadrature rule, we get

$$u_L(\mathbf{x},t) = \sum_{k=1}^{q} \frac{w_k}{4\pi(t-\tau_k)} G_{4(t-\tau_k)}[f(\cdot,\tau_k)](\mathbf{x}), \tag{13}$$

$$\text{where} \quad G_{4(t-\tau_k)}[f(\cdot,\tau_k)](\mathbf{x}) = \int_{\Omega} e^{-\frac{|\mathbf{x}-\mathbf{y}|^2}{4(t-\tau_k)}} f(\mathbf{y},\tau_k)\, d\mathbf{y}. \tag{14}$$

Therefore, the local part evaluation involves performing the Gauss transform of f at the quadrature nodes τ_k and then computing the discrete sum (13).

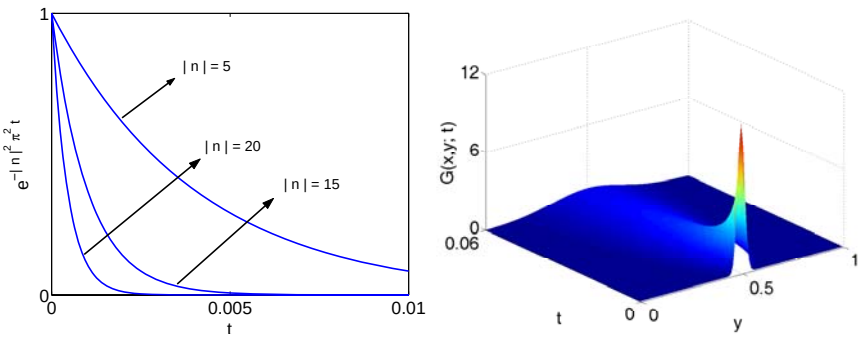
Now we can outline the algorithm. Given f, we use a time marching scheme in which the solution is time decomposed into two parts, the far and local. To compute the far part, we need to evaluate equations (9), (11),(10), and (7)—in that order. These computations involve evaluations of the sine transform of f, using the Fourier coefficients of f to update the coefficients of u_F, and use of the inverse Fourier transform to get back to space coordinates. To compute the local part, we need a few fast Gauss transforms to evaluate (14) at the τ_k time points needed for the time quadrature (13). This scheme, however, suffers from several technical difficulties: it should support non-uniform discretizations, and it should be fast. Below we describe algorithmic challenges related to complexity and high-accuracy.

Complexity. A direct evaluation of the far and local parts of u becomes non-optimal because of the following operations,

- Computing the sine transform of f can be very expensive since f is not given at a uniform grid, computing $\{\{\hat{f}_{\mathbf{n}}(\tau)\}_{n_1=1}^{p}\}_{n_2=1}^{p}$, requires $\mathcal{O}(Mp^2)$, can be very expensive as we typically expect $p^2 = \mathcal{O}(M)$. (For example, in the case in which we need to build a high-order representation of the solution, we would have that the as we increase the accuracy we need more targets M and the value at each target should be increasingly accurate.)
- Similarly computing the inverse sine transform of $C_{\mathbf{n}}$ is also expensive, since the evaluation points are non uniform.
- Finally, computing the Gauss transform of f at M target locations from the M leaf nodes requires $\mathcal{O}(M^2)$.

Accuracy. The computation of u_F, u_L involves the numerical evaluation of the sine transform (9), Gauss transform (14) and time integrals (11,12). The main challenges in constructing high-order convergent schemes for these integrals are,

- The kernel $e^{-|\mathbf{n}|^2\pi^2 t}$ in the far part update (11) is sharply peaked for large values of $|\mathbf{n}|$ (see Figure 1(a)). Special quadrature rules are needed to compute it accurately for all $|\mathbf{n}|$.
- As shown in Figure 1(b), the kernel in the local part is sharply peaked near the evaluation point $(\mathbf{x}, t)$. Thus, the numerical evaluation of Gauss transform (14) will not be straightforward.
- The input $f(\mathbf{x}, t)$ is given on adaptive data structures. Hence, the numerical integration schemes should be designed to handle adaptivity.



(a) Far part kernel (b) Near part kernel

Figure 1: *Plot of kernels that appear in the far and near parts of the volume potential. The computation of far part update (11) requires convolution with the kernel* $e^{-|\mathbf{n}|^2\pi^2 t}$ *that is plotted in 1(a) for different values of* $|\mathbf{n}|$*, with* $t = 0.01$*. The kernel is sharply peaked near* $t = 0$ *for higher values of* $|\mathbf{n}|$*. In 1(b) we plot the one dimensional kernel* $e^{-|x-y|^2/4t}/4\pi t$ *that appears in the local part evaluation, with* $x = 0.5$*. At* $t = 0$ *and as* $y \to x$*, the kernel is sharply peaked. Hence, a direct discretization of (11, 14) using standard quadrature rules is not computationally efficient.*

1.2 Synopsis of our method

One can "easily" design schemes that are high-accurate but are too expensive to be practical. Our goal is to devise a method that can deliver both accuracy and speed. This design goal led us to choose a Chebyshev polynomial representation for f. As we will see in later sections, this choice allows accurate precomputation of difficult integrals using product-quadratures [2], fast transformations between frequency and space-time, it allows non-uniform representation, it is numerically stable, and can be integrated nicely with state-of-the-art versions of the fast Gauss and non-uniform fast Fourier transforms.

The main features of our algorithm are:

- We propose a spectral version of fast Gauss transform to optimally compute the continuous Gauss transform of a function given by a piecewise polynomial representation (Chebyshev). We shall term this approach as the "*CFGT*" for Chebyshev Fast Gauss Transform. CFGT has a complexity and order of accuracy that are independent of the bandwidth the Gaussian. We compute the spatial integral in the local part (14) using CFGT.

- We propose a spectral version of the nonuniform fast Fourier transform to optimally compute the Fourier coefficients of a piecewise Chebyshev polynomial function, which we term "*CNUFFT*" for Chebyshev Nonuniform Fast Fourier Transform. We use CNUFFT to accelerate the computation of $\hat{f}_{\mathbf{n}}$ in (9). We use the inverse nonuniform FFT (INUFFT) for the fast computation of discrete sum (7).

- We compute the Chebyshev representation of $\hat{f}_{\mathbf{n}}(\tau)$ and then compute the time integral (11) exactly using recurrences. Since the kernel in (11) is not approximated, the convergence of this scheme is independent of $|\mathbf{n}|$.

[2] To illustrate the construction of product integration rule consider the computation of the transform, $F(x) = \int_{-1}^{1} e^{-xy} f(y)\, dy$ at some particular x. Assume that $f(y) = \sum_{k=0}^{q-1} f_k T_k(y)$, where $T_k(y)$ is Chebyshev polynomial of order k and f_k is the corresponding coefficient of f. Then, we have

$$F(x) = \sum_{k=0}^{q-1} f_k I_k(x) \quad \text{and} \quad I_k = \int_{-1}^{1} e^{-xy} T_k(y)\, dy.$$

The product integration rule is based on computing the moments I_k recursively starting from a base condition. In this example all the higher moments can be computed from I_0.

In a nutshell, the main contributions of this work are the extensions of the nonuniform FFT and the fast Gauss transform for piecewise Chebyshev polynomial representations and combining the two transforms to get a fast, high-order accurate volume heat potential evaluation scheme.

1.3 Related work

The need to integrate over the entire history has been a major hindrance for using integral equation formulations in large scale numerical simulations. The fast Gauss transform [13], reduces the complexity of computing the discrete Gauss transform

$$G_\delta(\mathbf{x}_i) = \sum_{j=1}^{N} q_j e^{-|\mathbf{x}-\mathbf{s}_j|^2/\delta}, \quad i = 1, 2, \ldots, M; \tag{15}$$

from $\mathcal{O}(NM)$ to $\mathcal{O}(N+M)$. The FGT can be used to accelerate the computation of the the heat equation in a bounded domain, by solving integral equations that require evaluation of layered heat potentials on the boundary of domain. The algorithm proposed in [12] provides a strategy for the rapid evaluation of those potentials. Fast algorithms for the exterior domain problems were developed in [11].

Along with fast summation techniques, high-order convergent schemes are also required for the efficient evaluation of volume heat potentials. To our knowledge, the only fast, high-order method for the diffusion equation based on integral equations, is that proposed by Strain in [21]: a high-order convergence was obtained by analytic integration of the monic polynomial approximation of the input. This method is difficult to extend beyond the presented fourth-order convergent scheme, since a monic-polynomial basis is ill-conditioned. A natural question is, do we really need a higher-order method? The answer, provided the solution is sufficiently regular, is yes. This is because the higher-order method is significantly faster—even in the case in which we require only a few correct digits (see Section 6).

Gauss transform. Apart from applications in computational physics, FGT has been applied in diverse fields like computer vision [7], management sciences [4, 5] and others [13]. The central idea of the FGT is to use a degenerate approximation of Gaussian kernel based on Hermite polynomials. It turns out that the most expensive part of the FGT algorithm is translating the Hermite expansions from the source to the target boxes. A plane-wave expansion for the Gaussian that makes the translation operator diagonal was proposed in [14]. The complexity of translation per target box was thereby reduced from $\mathcal{O}((2n+1)^d dp^{d+1})$ to $\mathcal{O}((2n+1)^d p^d)$, where p is number of terms retained in the truncated expansion, d is the number of dimensions and $(2n+1)^d$ are the number of cells that cover the support of the Gaussian. In [14] the authors present a modified FGT algorithm, to further reduce the complexity of the FGT to $\mathcal{O}(3dp^d)$ by transforming efficiently local expansions of the potential to neighboring boxes using plane-wave-based translation operators. Our implementation also uses such plane-wave expansions of the Gaussian.

Although much attention was devoted in the acceleration of the discrete Gauss transform, only the algorithm of Strain [21] discusses the fast computation of the continuous Gauss transform. In this paper we extend that work to fast and higher-order computation of the continuous Gauss transform of a function defined by a piecewise Chebyshev polynomial representation.

Nonuniform FFT. The nonuniform FFT [6] is a fundamental algorithm that has been successfully applied in diverse fields. We do not attempt to review all the related literature. Representative articles that discuss the application of nonuniform FFT to accelerate the computation of Fourier coefficients for piecewise smooth functions include [3, 8, 19]. The first one uses projection of f on a spline multiresolution analysis which is then used to compute the Fourier coefficients; the second one is using Lagrange interpolation, and the last one considers functions. In all papers there is a discussion for fast computation in the case of non-smooth data. In this paper we consider the situation in which the data is given in terms of piecewise Chebyshev polynomial coefficients rather than a discrete points. It handles easily discontinuities along quadtree nodes, but can is not an efficient scheme for general discontinuous functions.

Product integration. Recursive product integration rules are not new. Integration based on recursion using Chebyshev polynomials dates back to the earlier papers by Clenshaw and Curtis [24] and Filippi [9]. The ideas were generalized by Piessens [16] to compute a variety of integral transforms. We only need recurrences for two kernels in the evaluation of (3). We construct the recurrences for these kernels in our context, which are a trivial extension of the ones presented in [16]. These kind of recurrences were also used in [23] in the context of developing high-order solvers for one dimensional diffusion equation in moving domains.

1.4 Contents.

In Section 2 we discuss Chebyshev polynomials and give the construction of the product integration rules for the heat kernel. We discuss the CFGT and CNUFFT in Sections 3 and 4 respectively. In Section 5 we present the overall algorithm for the evaluation of the heat potential. We discuss numerical experiments for the transforms and the heat equation in Section 6.

2 Chebyshev representation and product integration

First, we review a few basic properties of Chebyshev polynomials (see [17] for more detailed exposition). A nth-order Chebyshev polynomial denoted by $T_n(\xi)$ defined on the interval $\xi \in [-1,1]$ has a closed form expression given by $T_n(\xi) = \cos(n \cos^{-1} \xi)$ and $T_n(\xi)$ can be constructed from lower-order polynomials using the recurrence

$$T_n(\xi) = 2\xi T_{n-1}(\xi) - T_{n-2}(\xi), \quad T_0(\xi) = 1, \quad T_1(\xi) = \xi. \tag{16}$$

We denote the leaf nodes of a quadtree Q by $\ell_1, \ell_2, \ldots, \ell_M$. In a leaf node ℓ, the function $f(\mathbf{x}, t)$ is represented using its Chebyshev coefficients

$$f(\mathbf{x}, t) \approx \sum_{n=0}^{q-1} \sum_{m=0}^{q-n-1} f^{\ell}_{n,m}(\tau) T_n(\xi_1) T_m(\xi_2) \qquad (\xi_1, \xi_2) \in [-1,1]^2, \tag{17}$$

$$\text{and} \qquad f^{\ell}_{n,m}(\tau) \approx \sum_{k=0}^{q-1} f^{\ell}_{n,m,k} T_k \left(\frac{2}{\triangle t}(\tau - t) - 1 \right), \qquad \tau \in [t, t + \triangle t], \tag{18}$$

where (ξ_1, ξ_2) are local coordinates of ℓ. Since we include all the polynomials of order $q-1$, this is a qth-order approximation of $f(\mathbf{x}, t)$. We use indexing scheme of [2] to represent the quadtree.

2.1 Recurrence relations

To illustrate the ideas for the construction of product integration rules, we discuss the computation of $I(\alpha) = \int_0^1 e^{-\alpha x} f(x)\, dx$, where $f(x)$ is a locally smooth function and α is a real (or complex) number. The interval $[0,1]$ is divided into arbitrarily-sized cells, such that in each cell, a qth-order Chebyshev polynomial approximation of $f(x)$ is within the prescribed accuracy. The integration is carried out over each cell C with center c and radius r.

$$I(\alpha) = \sum_{C} \sum_{n=0}^{q-1} f_n \int_{c-r}^{c+r} e^{-\alpha x} T_n \left(\frac{x-c}{r} \right) dx = \sum_{C} \sum_{n=0}^{q-1} f_n I_n(\alpha). \tag{19}$$

We now derive recurrence relationships to compute the moments I_n,

$$I_n = \int_{x_a}^{x_b} e^{-\alpha x} T_n(\lambda x + \eta)\, dx, \tag{20}$$

$$= \left[\frac{e^{-\alpha x}}{2\lambda} \left(\frac{T_{n+1}}{n+1} - \frac{T_{n-1}}{n-1} \right) \right]_{x_a}^{x_b} + \alpha \int_{x_a}^{x_b} \left[\frac{e^{-\alpha x}}{2\lambda} \left(\frac{T_{n+1}}{n+1} - \frac{T_{n-1}}{n-1} \right) \right], \tag{21}$$

$$\Rightarrow I_{n+1} = c_{n+1} + (n+1)\left(\frac{2\lambda}{\alpha} I_n + \frac{I_{n-1}}{n-1}\right),$$
$$\text{where } c_{n+1} = \frac{n+1}{\alpha}\left[e^{-\alpha x}\left(\frac{T_{n+1}}{n+1} - \frac{T_{n-1}}{n-1}\right)\right]_{x_a}^{x_b}. \tag{22}$$

If $\alpha \gg 1$, then $e^{-\alpha x}$ is sharply peaked and hence computing $I(\alpha)$ using numerical quadratures is quite expensive. Whereas, the convergence rate of the product integration rule discussed here depends only on the order of approximation of the input $f(x)$. Since we used polynomials of order $q-1$, the overall rate of convergence is q. However, the recurrence (22) is unstable for smaller values of $|\alpha|$ and hence higher moments cannot be computed accurately. In this case, we resort to the Olver's algorithm [25]. (We give details in the Appendix.)

3 The Chebyshev Fast Gauss Transform

In this section, we present a fast, high–order algorithm for evaluating the continuous Gauss transform defined by

$$G_\delta f(\mathbf{x}) = \int_\Omega e^{-\frac{|\mathbf{x}-\mathbf{y}|^2}{\delta}} f(\mathbf{y})\, d\mathbf{y}, \quad \Omega = [0,1]^2. \tag{23}$$

The input is a quadtree with leaf nodes $\{\ell_k\}_{k=1}^M$. Each leaf node contains a qth-order Chebyshev polynomial representation of f. The aim is to evaluate $\{G_\delta f(\mathbf{x}_i)\}_{i=1}^M$ within a desired accuracy ϵ, for any required δ and in an optimal $\mathcal{O}(M)$ time.

For a given δ, the Gaussian centered at $\mathbf{x}$ decays exponentially outside a fixed interval $\mathbf{x} + [-r_g\sqrt{\delta}, r_g\sqrt{\delta}]^2$, where $r_g = \sqrt{\log(1/\epsilon)}$. We call this interval the *support* of Gaussian. Below we describe two methods to compute $G_\delta f$: the direct evaluation and the kernel expansion-based algorithm. The overall algorithm combines these two methods. In this way, we obtain an algorithmic complexity that is independent of δ.

3.1 Direct Evaluation

We can compute $G_\delta f$ from the piecewise polynomial representation of f using the following steps:

- We find the set of leaf nodes that cover the support of Gaussian centered at $\mathbf{x}$. We shall call this set the *interaction list* of $\mathbf{x}$ and denote it by $\mathcal{I}[\mathbf{x}]$.
- We truncate the domain of integration to $\mathcal{I}[\mathbf{x}]$.
- We compute and add the contribution of each $\ell \in \mathcal{I}[\mathbf{x}]$, to obtain $G_\delta f(\mathbf{x})$.

The contribution of a leaf node ℓ, with center $\mathbf{c}$ and side length $2r$, to $G_\delta f(\mathbf{x})$ is given by

$$\begin{aligned} G_\delta^\ell(\mathbf{x}) &= \int_{c_1-r}^{c_1+r}\int_{c_2-r}^{c_2+r} e^{-\frac{|\mathbf{x}-\mathbf{y}|^2}{\delta}} f(\mathbf{y})\, d\mathbf{y}, \\ &= \sum_{k=0}^{q-1}\sum_{j=0}^{q-k-1} f_{k,j}^\ell \int_{-1}^{1} e^{-\left(\frac{x_1-c_1}{\sqrt{\delta}} - \frac{r}{\sqrt{\delta}}\xi_1\right)^2} T_k(\xi_1)\, d\xi_1 \int_{-1}^{1} e^{-\left(\frac{x_2-c_2}{\sqrt{\delta}} - \frac{r}{\sqrt{\delta}}\xi_2\right)^2} T_j(\xi_2)\, d\xi_2, \\ &= I^T[x_1-c_1]\mathbf{f}^\ell I[x_2-c_2], \end{aligned}$$

where $\mathbf{f}^\ell$ is the matrix that contains the Chebyshev coefficients of f in ℓ. Here, we performed the change of variables $\xi_k = \frac{y_k - c_k}{r}$, $k = 1, 2$. The entries of the moment vectors are given by,

$$I_j[x_k - c_k] = \int_{-1}^{1} e^{-\left(\frac{x_k-c_k}{\sqrt{\delta}} - \frac{r}{\sqrt{\delta}}\xi_k\right)^2} T_j(\xi_k)\, d\xi_k \quad k = 1,2 \quad \text{and} \quad 0 \le j < q. \tag{24}$$

These moments are computed exactly using the recurrence relation (66) and appropriate scaling factors λ, η. The direct evaluation is useful when the size of Gaussian support is small compared to the smallest leaf node; in this case we need $\mathcal{O}(Mq^2)$ work to evaluate $G_\delta f$ at M target points.

3.2 Degenerate Kernel Expansion

When the support of the Gaussian spans $\mathcal{O}(M)$ leaf nodes, direct evaluation leads to $\mathcal{O}(M^2)$ complexity. Instead, following [13] and [14], we use the following plane-wave expansion of the Gaussian

$$e^{-\frac{|\mathbf{x}-\mathbf{y}|^2}{\delta}} = \frac{1}{4\pi}\int_{\mathbb{R}^2} e^{-\frac{|\mathbf{z}|^2}{4}} e^{\frac{i\mathbf{z}\cdot(\mathbf{x}-\mathbf{y})}{\sqrt{\delta}}}\, d\mathbf{z}. \tag{25}$$

The numerical evaluation of the above equation involves the following approximations:

- The domain of integration is truncated to a finite interval $[-L, L]^2$.
- Numerical integration of the truncated plane–wave expansion.

The truncation of $\mathbb{R}^2$ by $[-L, L]^2$ results in a e^{-L^2} rate of convergence for the trapezoidal rule. Indeed, let $2p+1$ be the number of trapezoidal nodes. Then in [22] it shown that

$$e^{-\frac{|\mathbf{x}-\mathbf{y}|^2}{\delta}} \approx \frac{L^2}{4\pi p^2} \sum_{-p\le\mathbf{k}\le p} e^{-\left(\frac{L|\mathbf{k}|}{4p}\right)^2} e^{\frac{iL\mathbf{k}\cdot(\mathbf{x}-\mathbf{y})}{p\sqrt{\delta}}}. \tag{26}$$

Following [13], we use the multi-index notation by which $-p \le \mathbf{k} \le p$ implies $-p \le k_j \le p$, for $j = 1, 2$. Clearly p depends on how oscillatory the integrand in the truncated plane–wave expansion is, which in turn depends on $|\mathbf{x}-\mathbf{y}|_{\max} = r_g\sqrt{\delta}$. For a required precision $\epsilon = 10^{-14}$ the parameters should be

$$r_g = 6, \quad L = 11 \quad \text{and} \quad p = 21. \tag{27}$$

The domain $\Omega = [0, 1]^2$ is divided into *uniform boxes* of length $2r_b$. In our implementation, we set $r_b = 2^{\lfloor \log_2 \sqrt{\delta} \rfloor - 1}$ so that a particular box is either fully embedded in some leaf node or vice–versa.

Remark. Note that we have two grids: the first one is a quadtree to represent $f(\mathbf{y})$ and the second one is a uniform grid required by FGT (see Figure 2(a)).

Since $r_b = \mathcal{O}(\sqrt{\delta})$, the support of Gaussian centered at a particular box C spans a fixed number of boxes. These boxes belong to C's *interaction list* that has $(2n+1)^2$ entries, where $n = \left\lceil \frac{r_g\sqrt{\delta}}{2r_b} \right\rceil - 1$. We denote the interaction list of the box C by $\mathcal{I}[C]$[3]. The Gauss transform at a particular point $\mathbf{x}$ that belongs to a target box C can be written as

$$G_\delta f(\mathbf{x}) \approx \sum_{-p\le\mathbf{k}\le p} \mathbf{v}_{\mathbf{k}}^C(\mathbf{x})\mathbf{l}_{\mathbf{k}}^C, \quad \mathbf{l}^C = \sum_{B\in\mathcal{I}[C]} \mathbf{w}^B \odot \mathbf{t}^{BC}, \tag{28}$$

where $\odot$ symbolizes element–wise multiplication (Hadamard product). Let $\lambda_{\mathbf{k}} = \frac{iL\mathbf{k}}{p\sqrt{\delta}}$ and $\mathbf{x}_C, \mathbf{y}_B$ denote the centers of boxes C, B respectively. Then,

$$\mathbf{w}_{\mathbf{k}}^B = \int_B e^{\lambda_{\mathbf{k}}\cdot(\mathbf{y}_B-\mathbf{y})} f(\mathbf{y})\, d\mathbf{y}, \quad -p \le \mathbf{k} \le p, \tag{29}$$

$$\mathbf{t}_{\mathbf{k}}^{BC} = e^{\lambda_{\mathbf{k}}\cdot(\mathbf{x}_C-\mathbf{y}_B)}, \tag{30}$$

$$\mathbf{v}_{\mathbf{k}}^C(\mathbf{x}) = \frac{L^2}{4\pi p^2} e^{-\left(\frac{L|\mathbf{k}|}{2p}\right)^2} e^{\lambda_{\mathbf{k}}\cdot(\mathbf{x}-\mathbf{x}_C)}. \tag{31}$$

[3]Notice that we already used the symbol $\mathcal{I}$ in the direct evaluation version and we also use the same symbol in the next Section in a different context. The meaning of $\mathcal{I}$ should be clear from the context.

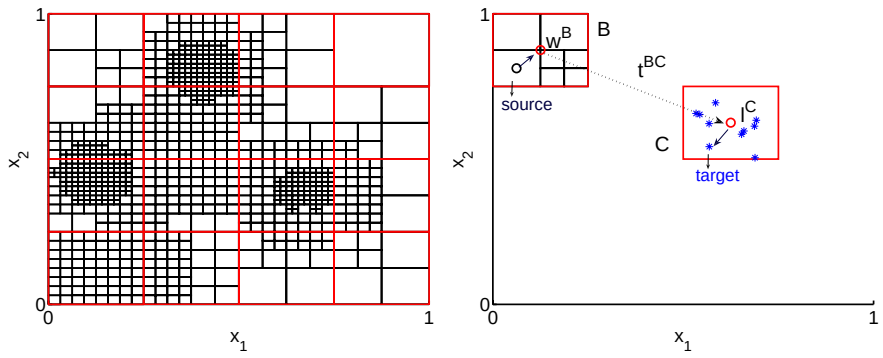


Figure 2: *Here we illustrate of the main steps involved in FGT. a) The leaf nodes (shown in black) of the given quadtree are assigned to regular boxes (shown in red). The size of the boxes is chosen so that each box corresponds to a node at some level of the quadtree. b) At a source box B, the influence of all the constituent leaf nodes is encoded in the wave expansion $\mathbf{w}^B$. The translation operator $\mathbf{t}^{BC}$ transfers the wave expansion of B to the target box C. At C, the influence of all the source boxes in $\mathcal{I}[C]$ is encoded in the local expansion $\mathbf{l}^C$. Finally, $G_\delta f$ at a target that belongs to C is given by $< \mathbf{l}^C, \mathbf{v}^C(\mathbf{x}) >$.*

$\mathbf{w}^B$	the wave expansion of box B.
$\mathbf{l}^C$	the local expansion of box C.
$\mathbf{t}^{BC}$	the operator that translates the wave expansion of box B to C.

The main steps involved in FGT can be summarized as follows,

- At each source box B form $\mathbf{w}^B$.
- At each target box C gather all the wave expansions from boxes belonging to $\mathcal{I}[C]$ and form $\mathbf{l}^C$.
- From $\mathbf{l}^C$ compute the Gauss transform at each target that belongs to C.

For additional details with respect to the implementation of the FGT algorithm we refer the reader to [13, 14]. The main difference of CFGT from FGT is the computation of $\mathbf{w}$ at each source box, which we discuss next.

Computing wave expansions. The computation of $\mathbf{w}^B$ involves finding the contribution of the leaf nodes within[4] the box B (see Figure 2(b))

$$\mathbf{w}^B = \sum_{\ell \in B} \mathbf{w}^\ell \odot \mathbf{t}^{\ell B}. \tag{32}$$

The definitions for $\mathbf{w}^\ell$ and $\mathbf{t}^{\ell B}$ are similar to (29, 30) i.e.,

$$\mathbf{w}^\ell_{\mathbf{k}} = \int_\ell e^{\lambda_{\mathbf{k}} \cdot (\mathbf{c}-\mathbf{y})} f(\mathbf{y})\, d\mathbf{y}, \quad \mathbf{t}^{\ell B}_{\mathbf{k}} = e^{\lambda_{\mathbf{k}} \cdot (\mathbf{y}_B - \mathbf{c})}. \tag{33}$$

Let $\mathbf{c}$ be the center and $2r$ be the length of ℓ. Then, the wave expansion of ℓ can be computed as follows:

$$\mathbf{w}^\ell_{\mathbf{k}} = \sum_{n=0}^{q-1} \sum_{j=0}^{q-n-1} f^\ell_{n,j} \int_{c_2-r}^{c_2+r} e^{\lambda_{k_2}(c_2-y_2)} T_n\left(\frac{y_2-c_2}{r}\right) dy_2 \int_{c_1-r}^{c_1+r} e^{\lambda_{k_1}(c_1-y_1)} T_n\left(\frac{y_1-c_1}{r}\right) dy_1. \tag{34}$$

[4] Apparently, it is possible that the box B could also be contained in a leaf node. It would be evident after we formally state CFGT that in this case, the leaf node containing B interacts with the targets directly instead of forming wave expansions.

By the change of variables $\xi = \frac{y_k - c_k}{r}$ for $k = 1, 2$ we have,

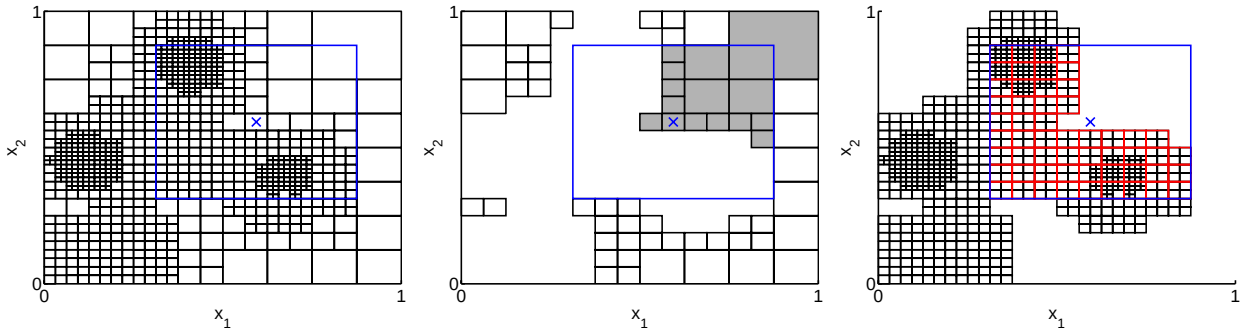
$$\mathbf{w}_{\mathbf{k}}^{\ell} = r^2 \sum_{n=0}^{q-1} \sum_{j=0}^{q-k-1} f_{n,j}^{\ell} \int_{-1}^{1} e^{-\lambda_{k_2} r\xi} T_n(\xi) d\xi \int_{-1}^{1} e^{-\lambda_{k_1} r\xi} T_j(\xi) d\xi, \qquad \text{and,} \quad \mathbf{w}^{\ell} = I^T[r] \mathbf{f}^{\ell} I[r], \tag{35}$$

$$\text{where} \quad I_{jk}[r] = r \int_{-1}^{1} e^{-\lambda_k r\xi} T_j(\xi) d\xi, \quad -p \leq k \leq p \quad \text{and} 0 \leq \quad j < q. \tag{36}$$

Each column of I is computed using the recurrences (22).

Acceleration Techniques. Although there seem to be lot of exponential evaluations, most of them can be precomputed.

1. Given the depth of the quadtree, we precompute the moments (36) for each level of the tree.
2. The translation operator $\mathbf{t}^{\ell B}$ depends on the relative position of ℓ within B. We first determine the distinct relative positions possible for the given quadtree and box size r_b. Then, we precompute the translation operator for each of these.
3. The operator $\mathbf{t}^{BC}$ depends on the relative position of B and C. The number of these is bounded from above by $(2n+1)^2$ and we precompute the corresponding translation operators.



(a) Q (b) Q_{Direct} (c) Q_{Expand}

Figure 3: *In this figure we illustrate the tree splitting that we use in CFGT to obtain an algorithmic complexity that is independent of the value of δ. Subfigure (a) shows the quadtree, a target $\mathbf{x}$, and the support of the Gaussian centered at $\mathbf{x}$ (blue box). For this δ, we can observe that a direct evaluation would be expensive because we have to loop through all the leaf nodes within the support. A kernel expansion based approach would also be expensive because it generates too many boxes. Hence, for optimal complexity, we must split the tree into two parts. Subfigure (b) shows the tree with the size of leaf nodes being greater than r_b. To evaluate the Gauss transform at a point $\mathbf{x}$, we just need to visit the leaf nodes in the support of the Gaussian (the shaded ones). There can be a maximum of $(2n+1)^2$ of such leaf nodes. Subfigure (c) show the tree with the size of leaf nodes being smaller or equal than r_b. In this tree we use the kernel expansion-based algorithm. Instead of visiting all the leaf nodes within the support, we only have to gather the wave expansion of the source boxes (red boxes) that are within the support. The number of these source boxes is bounded from above by $(2n+1)^2$.*

3.3 The complete algorithm

The implementation of the original FGT [13] is not efficient for the cases that δ is too small. In our context δ is dictated by the quadrature nodes in time that are used to evaluate $u_L(x,t)$. These nodes can be arbitrarily close to zero. We now describe an algorithm that attains optimal complexity for any δ.

We split the given quadtree into two trees (Figure 3): one tree in which the size of the leafs is greater than r_b and one tree that all its leaves have a size that is smaller or equal than r_b (3(c)). The Gauss transform at the targets is the sum of contributions from these two trees. In the first tree we use the direct evaluation version, and in the second we use the kernel expansion-based algorithm.

Algorithm 1 *Spectral Fast Gauss Transform*

SPLIT THE TREE
for $k = 1 : M$ **do**
  if $r_k \leq r_b$ **then**
    Assign the leaf node $Q(k)$ to Q_{Direct}.
  else
    Assign $Q(k)$ to Q_{Expand}.
  end if
end for
$G_\delta f(\mathbf{x})$ = DirectEvaluation$(Q_{\text{Direct}}, \mathbf{x})$ + KernelExpansion $(Q_{\text{Expand}}, \mathbf{x})$.

Algorithm 2 *DirectEvaluation*

INITIALIZATION
for $k = 1 : M$ **do**
  $G_\delta f(\mathbf{x}_k) = 0$.
end for

EVALUATION
for $k = 1 : M$ **do**
  Determine $\mathcal{I}[\mathbf{x}_k]$ (the leaf nodes in the interaction list).
  for each $\ell \in \mathcal{I}[\mathbf{x}_k]$ **do**
    Add contribution of each leaf node in the interaction list
    Compute the moments $I[x_1 - c_1]$ and $I[x_2 - c_2]$ (equation 24).
    $G_\delta f(\mathbf{x}_k) += I^T[x_1 - c_1]\mathbf{f}^\ell I[x_2 - c_2]$.
  end for
end for

Complexity Let the number of leaf nodes be M and the number of targets be N. The cost of direct evaluation is $\mathcal{O}((2n+1)^2 q^2 N \log M)$. The factor $(2n+1)^2 \log M$ is the cost of finding the at most $(2n+1)^2$ leafs in the interaction list of each target and q^2 is the cost of computing moments (66) at each target.

Computing the wave expansions at the leaf nodes is[5] $\mathcal{O}((2p+1)^2 q^2 N)$ since computing (35) for each $\mathbf{k}$ is $\mathcal{O}(q^2)$ and we have $-p \leq \mathbf{k} \leq p$. The translation $\mathbf{w}^\ell \rightarrow \mathbf{w}^B$ requires $\mathcal{O}((2p+1)^2)$. Therefore, computing the wave expansions at all the source boxes requires $\mathcal{O}((2p+1)^2 q^2 N)$. The computation of the local expansion at a target box

[5] A qth-order Chebyshev polynomial approximation has $q(q+1)/2$ coefficients. By using symmetries, we can show that the computation of wave expansions at the leaf nodes can be reduced to $\mathcal{O}\left(N\,(p+1)(p+2)q(q+1)/4\right)$. This reduction in constants would be significant in 3D.

Algorithm 3 *Kernel Expansion*

INITIALIZATION

for $k = 1 : M$ **do**

$G_\delta f(\mathbf{x}_k) = 0$.

end for

Precompute the moments I (equation (36)), given the depth of the quadtree.

FLAGGING

Determine the source and target boxes that interact

$\mathcal{B}_0 :=$ the minimal set of source boxes that covers all the leaf nodes.

$\mathcal{C}_0 :=$ the minimal set of target boxes that covers all the targets.

Find the source boxes that are in the interaction list of target boxes

$\mathcal{B} := \mathcal{B}_0 \cap \mathcal{I}[\mathcal{C}_0]$.

Find the target boxes that are in the interaction list of source boxes

$\mathcal{C} := \mathcal{C}_0 \cap \mathcal{I}[\mathcal{B}]$.

COMPUTE WAVE EXPANSIONS

for each $B \in \mathcal{B}$ **do**

for each $\ell \in B$ **do**

Compute the wave expansion of ℓ

$\mathbf{w}^\ell = I^T[r]\mathbf{f}^\ell I[r]$.

Add the contribution of ℓ *to wave expansion of* B

$\mathbf{w}^B += \mathbf{w}^\ell \odot \mathbf{t}^{\ell B}$.

end for

end for

COMPUTE LOCAL EXPANSIONS

for each $C \in \mathcal{C}$ **do**

for each box $B \in \mathcal{B} \cap \mathcal{I}[C]$ **do**

Translate the wave expansion of source box B *to target box* C

$\mathbf{l}^C += \mathbf{w}^B \odot \mathbf{t}^{BC}$.

end for

end for

EVALUATE AT THE TARGETS

for each $C \in \mathcal{C}$ **do**

for each $\mathbf{x} \in C$ **do**

$G_\delta f(\mathbf{x}) = < \mathbf{v}^C(\mathbf{x}), \mathbf{l}^C >$.

end for

end for

requires visiting all the $(2n+1)^2$ source boxes in its interaction list. The article [14] discusses a strategy to compute the local expansions by just visiting the neighbors. Using this construct[6], the complexity of forming the local expansion at a target box is $\mathcal{O}((2p+1)^2)$. Computing the Gauss transform at the targets from the local expansions is $\mathcal{O}((2p+1)^2 N)$.

[6] In order to use this construct, it is advisable to sort the target boxes using Morton ordering so that the local expansions of spatially close boxes are formed consecutively.

Hence, the overall complexity of the kernel expansion version of CFGT is $\mathcal{O}((2p+1)^2q^2M + (2p+1)^2N)$.

4 The Chebyshev Nonuniform Fast Fourier Transform

We present the algorithm in one dimensional setting. Consider the computation of Fourier coefficients of $f(x)$ for $x \in [0, 2\pi]$, defined by

$$\hat{f}_k = \int_0^{2\pi} f(y)e^{-iky}\,dy, \quad k = -\frac{M}{2}, ..., \frac{M}{2} - 1. \tag{37}$$

The input is a Chebyshev polynomial representation of $f(x)$ on a binary tree data structure imposed on $[0, 2\pi]$, with M leaf nodes. A direct computation of the integral (37) would lead to $\mathcal{O}(M^2)$ complexity. We closely follow the notations of [15] and describe an extension of nonuniform FFT for this situation that reduces the complexity to $\mathcal{O}(M \log M)$.

The first step in nonuniform FFT is to convolve $f(x)$ with the periodic heat kernel $g_\tau(x) = \sum_{k=-\infty}^{\infty} e^{-(x-2k\pi)^2/4\tau}$, the result of which would be a periodic function denoted by $f_\tau(x)$.

$$f_\tau(x) = f * g_\tau(x) = \int_0^{2\pi} f(y)g_\tau(x-y)dy. \tag{38}$$

Since $f_\tau(x)$ is smooth and 2π–periodic, its Fourier coefficients can be computed by the trapezoidal rule with M_r nodes and by computing the discrete sum using FFT. So, the task at hand is to compute f_τ on an oversampled regular grid i.e., at points $x_j = jh, j = 0, 1, .., M_r - 1$, where $h = 1/M_r$. The oversampling ratio $R = M_r/M$ is chosen based on the desired accuracy. In the definition of g_τ, each Gaussian has a support given by $4r_g\sqrt{\tau}$, where $r_g = \sqrt{\log(1/\epsilon)}$, for desired precision ϵ. The number of oversampled grid points within the support of a Gaussian in each direction, denoted by M_{sp}, is given by $M_{sp} = 2r_g\sqrt{\tau}/h$.

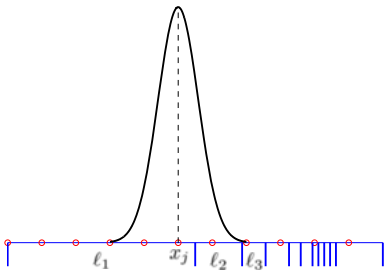


Figure 4: *The support of a Gaussian centered at x_j covers the leaf nodes ℓ_1, ℓ_2 and ℓ_3. Hence, only three leaves contribute to $f_\tau(x_j)$. The contribution of each leaf node is given by the inner product of its constituent polynomial coefficients of f and the moments of polynomials with the Gaussian. These moments are recursively computed using equation (66), which is discussed in the Appendix.*

Consider the computation of $f_\tau(x_j)$. The kernel $g_\tau(x_j - y)$ in (38), is the sum of the Gaussian centered at x_j and its images. Depending on the choice of τ and the location of target x_j, only a few images of g_τ are non–negligible within $[0, 2\pi]$. For illustration purposes, it is sufficient to consider just the Gaussian centered at x_j. In figure (4), the support of Gaussian spreads over three leaf nodes ℓ_1, ℓ_2 and ℓ_3. The interval of integration in (38) can thus be truncated to these three leaf nodes. The contribution of ℓ_1 that has Chebyshev coefficients $\{f_k^{\ell_1}\}_{k=0}^{q-1}$ is given by,

$$f_\tau^{\ell_1}(x_j) = \sum_{k=0}^{q-1} f_k^{\ell_1} I_k, \quad \text{where} \quad I_k = \int_{c_1-r_1}^{c_1+r_1} e^{-\frac{(x_j-y)^2}{4\tau}} T_k\left(\frac{y-c_1}{r_1}\right) dy, \tag{39}$$

$$I_k = \lambda \int_{-1}^{1} e^{-\left(\frac{y-\eta}{\lambda}\right)^2} T_k(y)\,dy, \quad \text{where} \quad \lambda = \frac{2\sqrt{\tau}}{r_1}, \eta = \frac{x_j - c_1}{r_1}. \tag{40}$$

The contributions of ℓ_2 and ℓ_3 are computed similarly. Hence, the computation of f_τ at a particular target involves visiting all the leaf nodes in which g_τ is non–negligible. Instead of finding the leaf nodes that influence a target x, we loop over the leaf nodes and determine which targets they influence. This is efficient because all the targets belong to a Cartesian grid.

Notation. The interaction list of a particular leaf node ℓ_k is the set of all regular grid points influenced by it, denoted by $\mathcal{I}[\ell_k]$.

Notice that computing f_τ at a particular target is equivalent to computing the Gauss transform of f. However, we do not require FGT here because the targets belong to a Cartesian grid. The following lemma establishes the complexity of computing f_τ at the oversampled regular grid points.

Lemma 4.1 *The cumulative sum of length of the interaction lists of all the leaf nodes is* $\mathcal{O}(M)$.

Proof Consider a particular leaf node ℓ_k. The number of targets within ℓ_k is bounded from above by $\frac{2r_k}{h_r}+1$, where $h_r = 2\pi/RM$. The number of targets in $\mathcal{I}[\ell_k]$ that are to the right (and similarly to the left) of ℓ_k is given by M_{sp}. Therefore, we have

$$\begin{aligned}\sum_{k=1}^{M} \text{length}(\mathcal{I}[\ell_k]) &\leq \sum_{k=1}^{M} \frac{2r_k RM}{2\pi} + 2M_{sp} + 1, \\ &= (2M_{sp}+1)M + RM\sum_{k=1}^{M}\frac{r_k}{\pi}.\end{aligned}$$

The sum of lengths of leaf nodes ($\sum_{k=1}^{M} 2r_k$) is the length of domain (2π), and thus

$$\sum_{k=1}^{M} \text{length}(\mathcal{I}[\ell_k]) \leq (2M_{sp}+1+R)M. \tag{41}$$

It follows that the complexity of computing f_τ, at M_r equispaced points, is $\mathcal{O}\left(2M(1+R+M_{sp})q\right)$, since each leaf node contains q polynomial coefficients.

Once f_τ is computed at all the points on the oversampled grid, FFT is used to compute $\hat{f}_\tau(k)$ in $\mathcal{O}(M_r \log M_r)$ time. The Fourier coefficients of f are extracted from $\hat{f}_\tau$ using the relation ([15]),

$$\hat{f}_k = \sqrt{\frac{\pi}{\tau}} e^{k^2\tau} \hat{f}_{\tau k}. \tag{42}$$

The accuracy of the transforms $f \rightarrow f_\tau \rightarrow \hat{f}_\tau$, depends on the choice of parameters τ, R, M_{sp}. The details of the analysis can be found in [6]. We follow [15] and note that for double precision $M_{sp} = 12, R = 4$ and $\tau = \pi^2/4M^2$.

Precomputation. Although the moments I in equation (39) can be computed in constant time on the fly using the recurrence (66), we can further accelerate the computations by by precomputing the Chebyshev moments of the Gaussian for a set of scaling values that only depend on the depth of the tree. We set $M_r = 2^{\lceil \log_2 RM \rceil}$ to minimize the number of such distinct values (see Appendix, for λ and η).

Sine transform. To compute the far part u_F, we need to accelerate the sine transform (9) computation. To accomplish this, CNUFFT can be used. The 1D analogue of sine transform is given by,

$$f_n^s = \int_0^1 f(y)\sin(n\pi y)\,dy, \quad n = 1, 2, ..., M. \tag{43}$$

We cannot use the Fourier coefficients of f to compute its sine transform, since the period of underlying basis for both these transforms differ by a factor of two. Instead, we extend f to the domain $(0, 2)$ by

$$F(y) = \begin{cases} f(y) & y \in (0,1), \\ 0 & \text{otherwise.} \end{cases} \tag{44}$$

The data structure for F is obtained by adding an extra leaf node to the binary tree representation of f with $c = \frac{3}{2}$, $r = \frac{1}{2}$ and setting the q polynomial coefficients of this leaf node to zero. The transforms on F are given by,

$$F_n^s = \frac{1}{2}\int_0^2 F(y)\sin(n\pi y)\,dy; \quad \hat{F}_n = \frac{1}{2}\int_0^2 F(y)e^{-n\pi y}\,dy. \tag{45}$$

We can compute $\hat{F}$ using CNUFFT and then F^s can be computed using the relation $F_n^s = -\text{Im}[\hat{F}_n]$. The sine transform of f can then retrieved from the relation $f_n^s = \frac{1}{2}F_n^s$. Although, the extension $f \to F$ is most likely discontinuous, we do not lose accuracy since CNUFFT requires only piecewise smoothness.

4.1 2D CNUFFT

Input: A quadtree data structure Q with M leaf nodes, each containing the qth order Chebyshev coefficients of $f(\mathbf{x})$[7].

Output: The $M_o \times M_o$ Fourier coefficients $\hat{f}_{k_1k_2}$, where $-\frac{M_o}{2} \le k_1, k_2 \le \frac{M_o}{2} - 1$.

Algorithm 4 *2D CNUFFT*

INITIALIZATION

Given target precision, choose R, τ, M_{sp}.
Set $M_r = 2^{\lceil \log_2 RM_o \rceil}$.
Precompute the FGT moments I for all the possible scaling values.
for $k = 1 : M_r^2$ **do**
  $f_\tau(\mathbf{x}_k) = 0$.
end for

CONVOLUTION

for $k = 1 : M$ **do**
  Determine $\mathcal{I}[\ell_k]$ (the interaction list of ℓ_k).
  for each $\mathbf{x} \in \mathcal{I}[\ell_k]$ **do**
    Compute t scaling factors $\{\lambda_i, \eta_i\}_{i=1}^2$;
    For these values, extract the moments I_1, I_2 from the precomputed values.
    Add the contribution of ℓ_k to $f_\tau(\mathbf{x})$: $f_\tau(\mathbf{x}) += I_1^T \mathbf{f}^{\ell_k} I_2$.
  end for
end for $\mathcal{O}(q^2M_r^2 + (2M_{sp}+1)^2q^2M)$

FFT ON OVERSAMPLED REGULAR GRID

Using FFT, compute $\hat{f}_{\tau k_1 k_2}$, for $-\frac{M_r}{2} \le k_1, k_2 \le \frac{M_r}{2} - 1$ $\mathcal{O}(M_r^2 \log M_r)$.
Compute $\hat{f}$ from $\hat{f}_\tau$ using analytic relations.

[7] The linear quadtree representation of [2] stores just one index corresponding to each leaf node, the geometric information of the leaf node can be extracted from this index.

5 Computing heat potentials

Using CFGT and CNUFFT, we can optimally compute the volume potential (3). Here, we describe our treatment of far and local parts after providing analysis for the optimal choice of the parameters p and δ.

5.1 Optimal splitting in time

The choice of parameters $\{p, \delta\}$ affects the overall complexity and the errors due to the truncation of the kernel expansions (4, 5). For our purposes, it is sufficient to note that the following estimates for the truncation errors hold true:

$$\mathcal{E}_F(p,\delta) = \mathcal{O}\left(\frac{e^{-\pi^2 p^2 \delta}}{p\delta}\right) \quad \text{and} \quad \mathcal{E}_L(d,\delta) = \mathcal{O}\left(\frac{e^{-d^2/\delta}}{\delta}\right). \tag{46}$$

Here d is the distance between the unit box and the nearest target.

In [20, 18] δ was chosen so that the truncation errors get reduced as N, M are increased. In our case, N and M are given parameters, which are decided solely based on resolving $f(\mathbf{x}, t)$ within desired accuracy. Instead, we proceed as follows. From the estimates (46), we can infer that

- Given d, we obtain $\mathcal{E}_L < \epsilon$ by requiring that $\delta \leq \delta_\epsilon$. Similarly, $\mathcal{E}_F < \epsilon$ if $p \geq p_\epsilon$.
- δ_ϵ and p_ϵ are threshold values that depend on the domains Ω and ω.

For example, if $\omega = [0.4, 0.6]^2$, then for double precision, $\delta_\epsilon = 0.002$ and $p_\epsilon = 55$.

So, for a given ϵ we must pick δ, p such that $\delta \leq \delta_\epsilon$ and $p \geq p_\epsilon$. When $\triangle t$ is bigger than δ_ϵ, we set $\delta = \delta_\epsilon$ and $p = p_\epsilon$. When $\triangle t$ is smaller than δ_ϵ, we have to reduce δ to preserve optimality. This in turn forces p to be more than p_ϵ to restrict $\mathcal{E}_F$ within ϵ. We set $\delta = l\triangle t$ which implies that the local part computation involves l time steps. Using CFGT, the local part requires $\mathcal{O}(Mlq^3)$ work per time step. By using CNUFFT and INUFFT, the far part requires $\mathcal{O}((M + p^2)q^3 \log p)$ work per time step. Hence, the overall complexity is $\mathcal{O}((Ml + M \log p + p^2 \log p)Nq^3)$. We chose $p = c_1\sqrt{M}$ and $l = \left\lceil c_2 \frac{\log M}{N\triangle t} \right\rceil$. Then, the truncation errors are given by

$$\mathcal{E}_F(M,N) = \mathcal{O}\left(\frac{N}{\sqrt{M}\log M} e^{-\frac{\pi^2 c_1^2 c_2 M \log M}{N}}\right), \quad \mathcal{E}_L(M,N) = \mathcal{O}\left(\frac{N}{\log M} e^{-\frac{d^2 N}{c_2 \log M}}\right). \tag{47}$$

So, by appropriately choosing the constants c_1, c_2, we can ensure that the truncation errors are within the required accuracy. *The overall complexity for this choice of parameters would then be* $\mathcal{O}(MNq^3 \log M)$.

Next, we illustrate our construction of quadrature rules for the far and local parts for the case in which $\delta = l\triangle t$. It is straightforward to extend for the case where $\delta = \delta_\epsilon$.

5.2 Far part

The far part $u_F(\mathbf{x}, t)$ is computed by evaluating the discrete sum (7) using INUFFT. To update the coefficients $C_\mathbf{n}$ we first compute the Chebyshev coefficients of $\hat{f}_\mathbf{n}(\tau)$ and then use product integration rule to evaluate the update (11) and then the recurrence (10).

In the interval $[t - l\triangle t, t + \triangle t - l\triangle t]$, $\hat{f}_\mathbf{n}(\tau)$ is computed at the Chebyshev nodes given by

$$\tau_i = t - l\triangle t + \frac{\triangle t}{2}\left(1 + \cos(\pi(i-1)/q)\right), \quad i = 1, .., q. \tag{48}$$

At each τ_i we use CNUFFT to optimally compute $\hat{f}_{\mathbf{n}}$, defined in (9). From $\hat{f}_{\mathbf{n}}(\tau_i)$, we compute its Chebyshev coefficients[8] $\hat{f}_{\mathbf{n}}(k), k = 0, .., q-1$. By the change of variable $\tau = t + (1-\theta-l)\triangle t$,

$$U_{\mathbf{n}}[\hat{f}] = \triangle t e^{-|\mathbf{n}|^2\pi^2 l\triangle t} \sum_{k=0}^{q-1} \hat{f}_n(k) \int_0^1 e^{-(|\mathbf{n}|^2\pi^2\triangle t)\theta} T_k(-2\theta+1)\, d\theta = \sum_{k=0}^{q-1} \hat{f}_{\mathbf{n}}(k) E_{\mathbf{n}k}, \tag{49}$$

$$E_{\mathbf{n}k} = \triangle t e^{-|\mathbf{n}|^2\pi^2 l\triangle t} \int_0^1 e^{-(|\mathbf{n}|^2\pi^2\triangle t)\theta} T_k(-2\theta+1), \quad n_1 = 1, .., p, n_2 = 1, .., p,\ k = 1, .., q. \tag{50}$$

The $p^2 \times q$ entries of the matrix E can be computed in an optimal $\mathcal{O}(p^2 \times q)$ using the recurrences (22).

5.3 Local part

Let us rewrite (12) as,

$$u_L(x,t) = \int_{t-l\triangle t}^{t} g(\tau)\, d\tau, \quad g(\tau) = \int_\Omega \frac{e^{-\frac{|\mathbf{x}-\mathbf{y}|^2}{4(t-\tau)}}}{4\pi(t-\tau)} f(\mathbf{y},\tau)\, d\mathbf{y}\, d\tau. \tag{51}$$

We can easily show that g is a smooth function in time. To compute u_L, we first compute the Chebyshev coefficients of $g(\tau)$ in each of the l previous time steps.

$$\begin{aligned} u_L(x,t) &= \sum_{k=1}^{l} \int_{t-k\triangle t}^{t-(k-1)\triangle t} g(\tau)\, d\tau \\ &= \sum_{k=1}^{l}\sum_{n=0}^{q-1} g_n^k I_n, \quad \text{where} \quad I_n = \int_{-1}^{1} T_n(\xi)\, d\xi = \frac{1+(-1)^n}{1-n^2}, \end{aligned}$$

where g^k is the Chebyshev polynomial representation of g in $(t-k\triangle t, t-(k-1)\triangle t)$. We compute g^k by evaluating $g(\tau)$ at the Chebyshev nodes

$$\tau_i = t - k\triangle t + \frac{\triangle t}{2}\left(1+\cos(\pi(i-1)/q)\right), \quad i = 1, .., q. \tag{52}$$

6 Numerical results

In this section, we illustrate the performance of the algorithms presented on a suit of test problems. The criterion we used to construct the quadtree is spectral thresholding, as discussed in [10]: a leaf node is subdivided if the tail of Chebyshev coefficients it owns has not decayed enough. Upon completion of the tree construction, every leaf node contains the coefficients $f_{m,n}$ such that

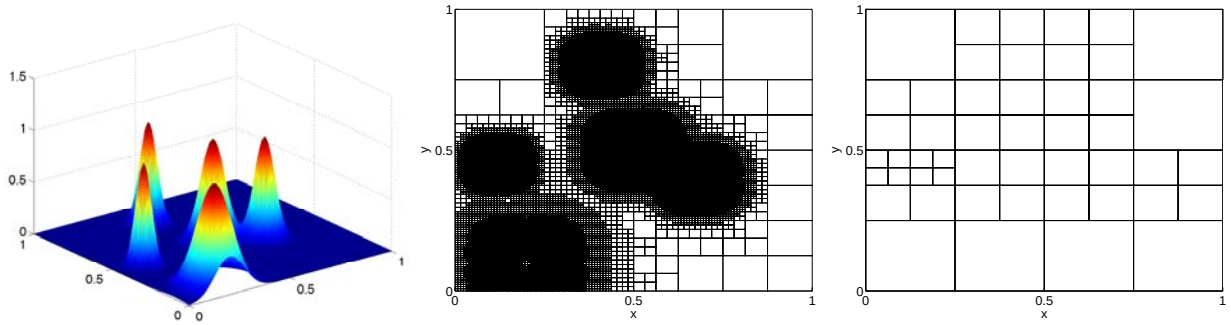
$$\sum_{k=0}^{q-1} |f_{k,q-k-1}| < \epsilon^*. \tag{53}$$

We implemented the algorithms presented here in MATLAB. For each test problem, we report CPU times w.r.t. a base case.

6.1 Chebyshev FGT

We present the convergence results for CFGT on two examples. In Table 1, we present the results for the Gauss transform of a function that is uniformly smooth. In Table 2, we consider the function shown in Figure 5.

[8] It is well-known that the q Chebyshev coefficients can be computed in $\mathcal{O}(q \log q)$ time using FFT. Also to compute $\hat{f}_{\mathbf{n}}$, we need $f(\mathbf{y}, \tau_i)$ from the piecewise Chebyshev polynomial representation of f. This again can be accomplished in optimal time using inverse FFT.



(a) Original function (b) q = 4 (c) q = 16

Figure 5: *In this figure we give an example of a tree-based representation of the function f shown at the left most figure. The quadtrees were constructed top-down, starting from the unit square, and refining using thresholding (53) with $\epsilon^* = 10^{-5}$. The leaf nodes of the quadtree shown in the middle use cubic polynomials, while the rightmost tree uses fifteenth-degree polynomials. While the cubic polynomial tree uses 40513 leaf nodes, the fifteenth-order polynomial tree meets the required accuracy criterion with just 49 leaf nodes. Taking into account the storage per leaf node (q^2), the high-order requires 50 times less storage. For tighter discretization tolerances, this factor becomes even greater.*

$q \rightarrow$	4		8		16	
$N = M$	CPU time	error	CPU time	error	CPU time	error
16	1	5.56e-02	6.67e-01	1.68e-04	1.00e+00	2.41e-08
64	1.67e+00	5.66e-03	1.67e+00	7.04e-06	2.00e+00	1.22e-12
256	2.33e+00	6.83e-05	3.00e+00	4.11e-09	4.33e+00	3.00e-15
1024	7.00e+00	3.60e-06	8.00e+00	1.28e-11	1.07e+01	3.74e-15
4096	2.20e+01	2.17e-07	2.57e+01	4.65e-14	3.43e+01	4.86e-15
16384	8.03e+01	1.34e-08	9.60e+01	4.67e-15	1.23e+02	4.48e-15

Table 1: *CPU Time and relative errors in computing $G_\delta f(\mathbf{x}) = \int_\Omega e^{-\frac{|\mathbf{x}-\mathbf{y}|^2}{\delta}} f(\mathbf{y})\, d\mathbf{y}$ with bandwidth $\delta = 0.1$; here $f(\mathbf{y}) = \sin^2(20y_1)\sin^2(20y_2)$. N is the number of leaf nodes, M is number of target points (which are chosen randomly in the unit box $[0,1]^2$) and q is the order of Chebyshev polynomials used to approximate $f(\mathbf{y})$ in each leaf node. CPU times are relative to the base case $N = 16$ and $q = 4$.*

$\delta \rightarrow$		10^{-2}		10^{-4}		10^{-6}	
ϵ^*	$N = M$	CPU time	error	CPU time	error	CPU time	error
10^{-6}	79	1	5.50e-012	2.35	3.26e-010	0.02	8.53e-010
10^{-8}	142	1.27	2.40e-014	3.99	4.69e-012	0.04	1.22e-010
10^{-10}	265	1.95	7.91e-016	5.99	2.07e-013	0.07	8.39e-014
10^{-12}	481	2.50	7.91e-016	9.53	2.11e-013	0.15	2.70e-016

Table 2: *Performance of 16th-order CFGT in computing $G_\delta f(\mathbf{x})$ of the function shown in Figure 5, for different values of δ. The exact solution is analytically computed. Here ϵ^* is the discretization tolerance based on which the quadtree is constructed using the criterion (53). We can see that the convergence is independent of δ. The trend in the CPU times can be explained as follows: when δ decreases, the number of source/target boxes increase and that in turn increases the total time. As δ becomes really small, the direct evaluation version of CFGT is invoked and hence CPU time is reduced. CPU times are relative to the base case $\delta = 10^{-2}$ and $N = 79$.*

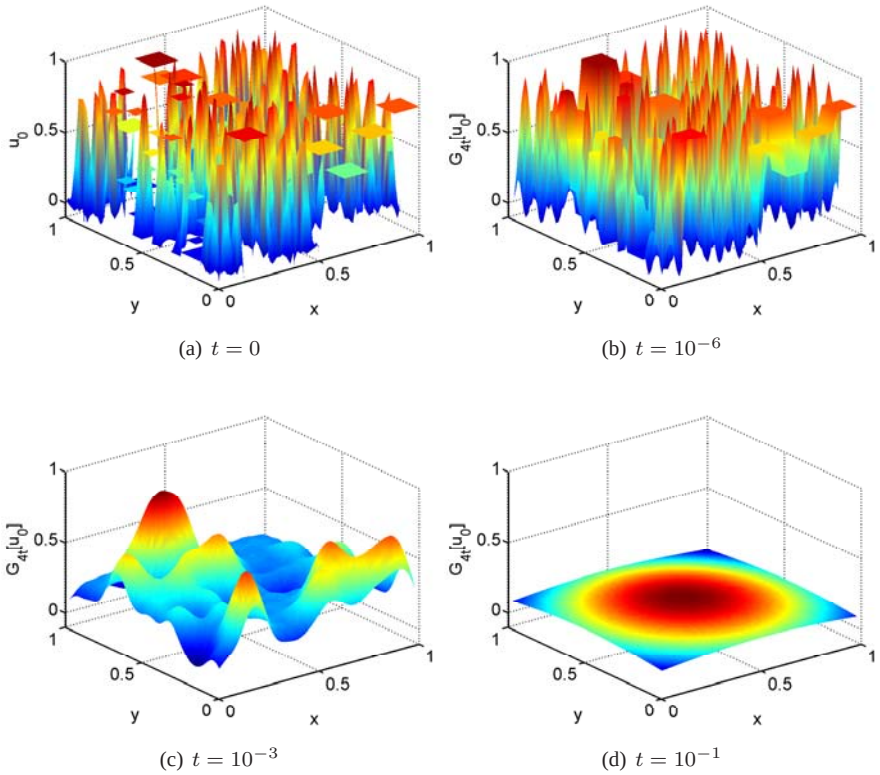
Free space initial condition problem. Consider the initial condition problem,

$$\frac{\partial u}{\partial t} = \triangle u \quad \text{in} \quad \mathbf{R}^d, \quad \text{s.t.} \quad u(\mathbf{x}, 0) = u_0(\mathbf{x}). \tag{54}$$

The exact solution is given by

$$u(\mathbf{x}, t) = \frac{1}{4\pi t} \int_\Omega e^{-\frac{|\mathbf{x}-\mathbf{y}|^2}{4t}} f(\mathbf{y})\, d\mathbf{y}. \tag{55}$$

Now, assuming $\Omega = [0,1]^2$ and u_0 be given in terms of piecewise Chebyshev polynomial coefficients, we can use CFGT for fast and high–order computation of (55). Here, we solve the free space initial condition problem (55) with complicated function u_0, as shown in Figure 6(a).



(a) $t = 0$ (b) $t = 10^{-6}$

(c) $t = 10^{-3}$ (d) $t = 10^{-1}$

Figure 6: *Solution of the free space initial condition problem (54) at different times. The initial condition u_0 is discontinuous and oscillatory. The input is a quadtree that has 94 leaf nodes, each containing a 16th-order Chebyshev polynomial representation of u_0.*

6.2 Chebyshev NUFFT

Here, we present the convergence results for computing the Fourier coefficients using CNUFFT. We test two functions: the first is discretized using a uniform tree, while the second is discretized using a nonuniform tree.

$q \rightarrow$	4		8		16	
M	CPU time	error	CPU time	error	CPU time	error
64	1	2.91e-01	1.0	1.18e-05	2.0	1.52e-14
256	1.0	1.16e-02	2.0	2.20e-08	2.0	5.67e-14
1024	5.5	5.71e-04	7.0	2.06e-10	8.5	9.76e-14
4096	19.5	3.30e-05	21.5	9.41e-13	24.5	1.22e-13

Table 3: *Results for CNUFFT of a non–periodic smooth function,* $f(\mathbf{x}) = \sin(20x_1)\sin(20x_2)$ *for* $\mathbf{x} \in [0,1]^2$*. The oversampling ratio was set to* $R = 4$ *and the spreading distance to* $M_{sp} = 12$*. CPU times are relative to* $q = 4$ *and* $M = 64$*.*

$q \rightarrow$	8		16	
Discretization tolerance	M	CPU time	M	CPU time
10^{-4}	244	5.7	37	1
10^{-6}	859	5.3	79	2.0
10^{-8}	3163	2.7e+01	142	2.3
10^{-10}	11380	1.8e+02	265	4.0
10^{-12}	42460	2.1e+03	481	4.9

Table 4: *Performance of 8th- and 16th-order methods in computing the Fourier coefficients of the function given in Figure 5. Here* M *is the number of leaf nodes. CPU times are relative to the base case* $q = 16$ *and* $\epsilon^* = 10^{-4}$*.*

6.3 Volume Potential

Here we present two examples in which we test our method for the case of the heat potential evaluation. In the first example we use a spatial grid that is regular at each time step. In the second example, the spatial grid is nonuniform and is different at each time step.

Example 1. Here, we evaluate the volume potential (3) of the function,

$$f(\mathbf{x},t) = \sin(n\pi x_1)\sin(n\pi x_2)\left(n\pi\cos(n\pi t) + 2n^2\pi^2\sin(n\pi t)\right) \quad \text{with} \quad n = 10. \tag{56}$$

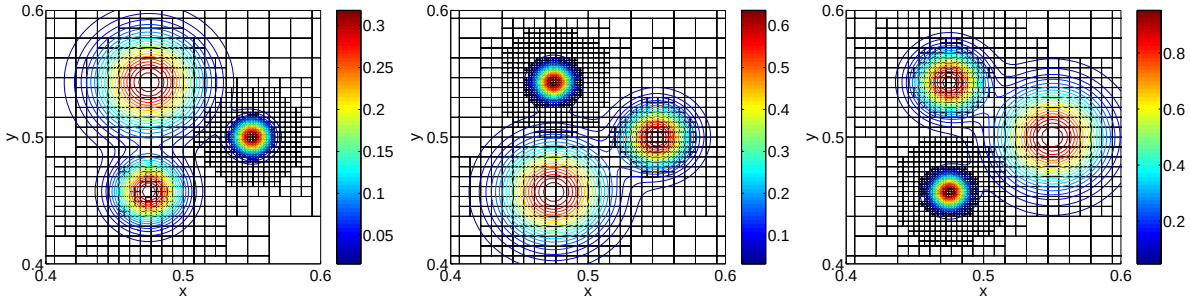
The exact solution is given by,

$$u(\mathbf{x},t) = \sin(n\pi x_1)\sin(n\pi x_2)\sin(n\pi t). \tag{57}$$

which satisfies the initial and boundary conditions (2). Starting from a regular grid in space–time, we progressively refine it and the absolute errors for each discretization is reported in Table 5.

N	M	$q = 4$	$q = 8$	$q = 16$
8	4	5.83e+00	4.57e-02	7.42e-08
16	16	1.08e-01	4.03e-05	3.22e-13
32	64	1.72e-03	8.47e-08	1.89e-15
64	256	3.18e-06	3.52e-10	2.16e-15

Table 5: *Absolute errors in computing the volume potential for example 1. The number of targets is* M *and they are randomly generated within* $\omega = [0.4, 0.6]^2$*. The total time is taken as* $T = 1$*. Here we report the discrete* $L^\infty(\omega \times [0,T])$ *norm of the error. In this example, we fixed the parameters to* $\delta = 0.002$ *and* $p = 55$ *so that truncation errors are within machine precision.*



(a) t = 1/3 (b) t = 2/3 (c) t = 1

Figure 7: *Quadtree for $f(\mathbf{x}, t)$, defined in example 2. The tree is generated using 8th-order Chebyshev polynomials, with discretization tolerance $\epsilon^* = 10^{-7}$. A contour plot of the solution $u(\mathbf{x}, t)$ is superimposed on the quadtrees. Evidently, the data structure changes at each step. We numerically compute $u(\mathbf{x}, t)$ at targets that are randomly located within $\omega = [0.4, 0.6]^2$ for each time step and compare with the exact solution in Table 6.*

ϵ^*	10^{-4}	10^{-6}	10^{-8}	10^{-10}
M_{max}	91	136	277	415
N	8	16	32	64
error	2.35e-06	1.00e-07	2.32e-11	1.78e-12

Table 6: *Performance of the 16th-order method on example 2. At each time step, the quadtree is constructed using the criterion $\sum_{k=0}^{q-1} |f_{k,q-k-1}| < \epsilon^* \|f(\mathbf{x}, k\triangle t)\|_\infty$ for $k = 0, 1, \ldots, N-1$. M_{max} is the maximum number of leaf nodes over all the quadtrees at different times. Here, the targets are randomly located in $\omega = [0.4, 0.6]^2$ and the error is measured in discrete $L^\infty(\omega \times [0, T])$ norm.*

Example 2. In this example, we compute the volume potential due to a forcing function that requires time dependent spatially adaptive grid. Here $f(\mathbf{x}, t)$ is constructed by the exact solution given by[9]

$$u(\mathbf{x}, t) = \sum_{i=1}^{3} t\, e^{-\alpha_i r_i^2} \quad r_i^2 = \|\mathbf{x} - \mathbf{c}_i(t)\|_2^2$$

$$\mathbf{c}_i(t) = 0.5 + 0.05 \begin{bmatrix} \cos(2\pi(t + i/3)) \\ \sin(2\pi(t + i/3)) \end{bmatrix}, \alpha_i = 400 + 100 \times 4^i. \tag{58}$$

The Gaussian centers $\mathbf{c}_i(t)$ move in a circular motion around the center of Ω. As the bandwidth of the Gaussians differ, the adaptive data structure is unsymmetric and hence it varies with time as shown in Figure (7).

7 Conclusions

We presented three transforms that can be applied to functions represented using piecewise Chebyshev polynomials: we have extended fast Gauss and nonuniform FFT transforms and, based on the fast algorithm of Greengard and Strain [12], we have presented a fast and high-order accurate algorithm for the computation of volume heat potential. We presented several numerical results in which the convergence rates of our 16th-order accurate scheme are verified.

We have set the initial condition in (2) to zero just for the ease of exposition. Our algorithm extends in an obvious way in the more general case. Given a piecewise Chebyshev polynomial representation of $u_0(\mathbf{y})$, we can use CFGT and CNUFFT for the fast and accurate evaluation of the volume potential that arises due to u_0 given by

$$\mathcal{V}[u_0](\mathbf{x}, t) = \int_\Omega G(\mathbf{x}, t; \mathbf{y}) u_0(\mathbf{y})\, dy.$$

Besides extending the method to the 3D case, there are two important directions that we have not explored: a posteriori error-based adaptive scheme; and the case of of imposing boundary conditions on domains with complex geometries. Both directions are part of ongoing work.

Returning to the 3D case, the extension of the method to higher dimensions is straightforward using tensor products for the Gauss transform, and 3D FFTs. The complexity constants for the FGT, however, increase exponentially (in the dimension) [14, 15]. Optimizing and parallelizing the code becomes crucial for practical applications.

8 Acknowledgements

We would like to thank John Strain for bringing the reference [25] to our attention.

9 Appendix

9.1 Olver's Algorithm

Consider the computation of the moments,

$$I_n(\alpha) = \int_0^1 e^{-\alpha x} T_n(2x - 1)\, dx, \quad 0 \le n \le q - 1. \tag{59}$$

We can use the recurrence (22) to compute I_n for $n > 2$ from base conditions I_0, I_1, I_2. The recurrence (22) is an instance of second-order nonhomogeneous difference equation for which Olver's algorithm can be applied. Let us rewrite the recurrence as,

$$I_n + a_n I_{n+1} + b_n I_{n+2} = f_n, \quad 1 \le n \le q - 1, \tag{60}$$

[9] Here $u(\mathbf{x}, t)$ satisfies the boundary conditions approximately.

with the coefficients, a_n, b_n, f_n defined appropriately. Since this is a second-order equation, it requires two initial conditions[10] I_1 and I_2. The scheme of Olver can be written as follows,

$$\text{Solve:} \quad I_n^N + a_n I_{n+1}^N + b_n I_{n+2}^N = f_n, \quad 1 < n \leq N. \tag{61}$$

$$\text{s.t.} \quad I_{N+1}^N = 0, \quad I_1^N = I_1. \tag{62}$$

For the recurrence (22), we can prove that,

$$\lim_{N\to\infty} I_n^N = I_n, \quad n \geq 1. \tag{63}$$

When α is small, for which direct computation using recurrence (22) is unstable, we use Olver's algorithm. For these values of α, the convergence (63) is quite rapid i.e., even for moderate values of N,

$$\max_{1<n\leq q-1} \left| I_n^N - I_n \right| < \epsilon. \tag{64}$$

More precisely, for double precision ($\epsilon = 10^{-12}$), we use Olver's algorithm with $N = 2q + 1$ to compute (22), if $|\alpha| < |\alpha_{th}|$. For different values of q, we tabulate the threshold values in Table 7

q	4	8	16
$\lvert\alpha\rvert_{th}$	1/2	4	16

Table 7: The threshold values $|\alpha|_{th}$ for different values of q. Whenever $\alpha \geq |\alpha|_{th}$, we compute $I_n(\alpha)$ using the recurrence (22). For $\alpha < |\alpha|_{th}$, we use Olver's algorithm with $N = 2q + 1$.

We solve (61) using LU decomposition. For smaller values of $|\alpha|$ than the threshold, we can achieve the required precision even with smaller N, but we do not carry the analysis any further. Since, the system (61) results in a sparse linear system, the complexity of solving it is $\mathcal{O}(N)$.

9.2 Recurrences for the Gauss kernel

Defining $I_k(\lambda, \eta)$ as,

$$I_n(\lambda, \eta) = \frac{1}{\lambda} \int_{-1}^{1} e^{-\left(\frac{y-\eta}{\lambda}\right)^2} T_n(y)\, dy, \quad 0 \leq n \leq q-1, \tag{65}$$

we can derive the following recurrence, that is valid for $n > 3$,

$$I_n = 2\eta I_{n-1} + 2\left((n-1)\lambda^2 + \frac{1}{n-3}\right) I_{n-2} - 2\eta\left(\frac{n-1}{n-3}\right) I_{n-3} + \frac{n-1}{n-3} I_{n-4} - C_n, \tag{66}$$

$$\text{where} \quad C_n = \lambda(n-1)\left[e^{-\left(\frac{y-\eta}{\lambda}\right)^2}\left(\frac{T_{n-1}}{n-1} - \frac{T_{n-3}}{n-3}\right)\right]_{-1}^{1}. \tag{67}$$

The recurrence (66) is unstable for larger values of λ. Since, (66) is a fourth-order nonhomogeneous system, one can use Lozier's algorithm [25] for stable computation for larger values of λ.

Since, however, we can precompute these moments, [11] the Gaussian moments, we simply used a high-order smooth quadrature rule to compute the moments I_n. Whenever $q > 4$ and $\lambda > \frac{1}{8}$, we use a 24th-order quadrature rule of [1], with q trapezoidal nodes to compute I_n for $4 < n < q$.

[10] When the recurrence (22) is written in the form (60), we see that $b_0 = 0$. Hence $n = 1$ pertains to the first initial condition, instead of $n = 0$. So, we require three base conditions for the recurrence (22).

[11] The direct evaluation version of CFGT does not precompute the moments. But, for the cases in which this algorithm is applicable, we can show that $\lambda \leq \frac{1}{8}$, so that the recurrence (66) can be used to compute the moments directly.

9.3 An extension of FGT

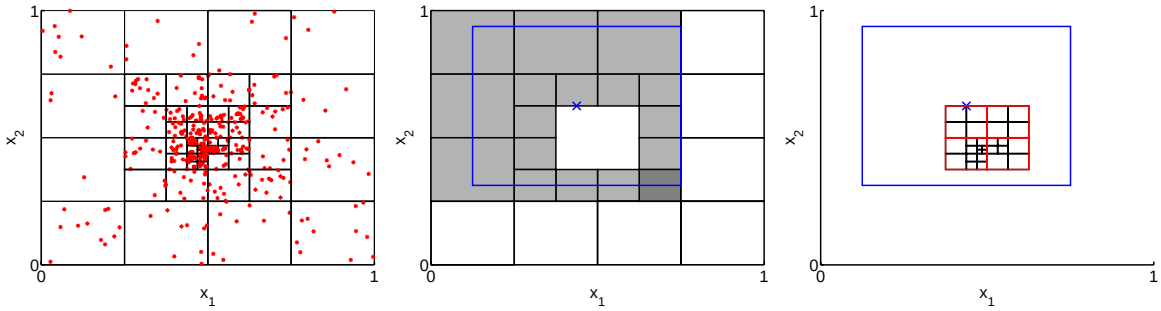
For some applications, one might be interested in computing the discrete Gauss transform (15) for relatively small values of δ. For instance, consider the free space initial condition problem (54) with the initial condition given by,

$$u_0(\mathbf{x}) = \sum_{j=1}^{M} q_j \delta(\mathbf{x} - \mathbf{y}_j). \tag{68}$$

As this function cannot be represented using piecewise polynomials, CFGT is not applicable. Substituting u_0 in (55), we get

$$u(\mathbf{x}, t) = \frac{1}{4\pi t} \sum_{j=1}^{M} q_j e^{-\frac{|\mathbf{x}-\mathbf{y}_j|^2}{4t}}. \tag{69}$$

The aim is to compute u at $\{\mathbf{x}_k\}_{k=1}^{M}$ and at any given time t. A direct application of FGT on (69) would be expensive at small values of t. This is because the number of boxes that FGT requires is inversely proportional to $\sqrt{t}$. We now describe an extension of the fast Gauss transform [13] that would enable the fast computation of (69) for any required t. The strategy is an extension of the idea used in CFGT to the discrete case. We explain this in Figure 8.



(a) Q (b) Q_{Direct} (c) Q_{Expand}

Figure 8: Here we illustrate the extension of FGT for efficient computation of (69) at smaller times. a) From the source distribution $\{\mathbf{y}_j\}_{j=1}^{M}$ we construct a quadtree Q so that each of its leaf contains no more than s points. In this example, we have set $s = 12$. We split Q into two trees: Q_{Direct} has leaves that are larger than the source boxes and Q_{Expand} has smaller ones. b) Here a target $\mathbf{x}$ and the support of Gaussian centered at $\mathbf{x}$ is shown. The leaf nodes that are shaded belong to the interaction list of $\mathbf{x}$. The sources that belong to these leaves interact directly. This would avoid the need to generate source boxes that are mostly empty. c) The red boxes are the source boxes that are within the interaction list of the target box containing $\mathbf{x}$. The sources that belong to these boxes interact with $\mathbf{x}$ through Hermite/Wave expansions.

References

[1] BRADLEY K. ALPERT, *Hybrid Gauss-trapezoidal quadrature rules*, SIAM Journal on Scientific Computing, 20 (1999), pp. 1551–1584.

[2] LAURENT BALMELLI, JELENA KOVACEVIC, AND MARTIN VETTERLI, *Quadtrees for embedded surface visualization: Constraints and efficient data structures*, pp. 487–491.

[3] GREGORY BEYLKIN, *On the fast Fourier transform of a function with singularities*, Applied and Computational Harmonic Analysis, 2 (1995), pp. 363–381.

[4] MARK BROADIE AND YUSAKU YAMAMOTO, *Application of the fast Gauss transform to option pricing*, Management Science, 49 (2003), pp. 1071–1088.

[5] ——, *A double-exponential fast Gauss transform algorithm for pricing discrete path-dependent options*, OPERATIONS RESEARCH, 53 (2005), pp. 764–779.

[6] ALOK DUTT AND VLADIMIR ROKHLIN, *Fast Fourier transforms for nonequispaced data*, SIAM Journal on Scientific computing, 14 (1993), pp. 1368–1393.

[7] AHMED ELGAMMAL, RAMANI DURAISWAMI, AND LARRY S. DAVIS, *Efficient kernel density estimation using the fast Gauss transform with applications to color modeling and tracking*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 25 (2003), pp. 1499–1504.

[8] GUO-XIN FAN AND QING HUO LIU, *Fast Fourier transform for discontinuous functions*, Antennas and Propagation, IEEE Transactions on, 52 (2004), pp. 461–465.

[9] S. FILIPPI, *Angenaherte Tschebysceff-approximation einer stammfunction–eine modifikation des verfahrens von clenshaw and curtis*, Numerische Mathematik, 6 (1964).

[10] LESLIE GREENGARD AND JUNE-YUB LEE, *A direct adaptive Poisson solver of arbitrary order accuracy*, Journal of Computational Physics, 125 (1996), pp. 415–424.

[11] LESLIE GREENGARD AND PATRICK LIN, *Spectral approximation of the free–space heat kernel*, Applied and Computational Harmonic Analysis, 9 (2000), pp. 83–97.

[12] LESLIE GREENGARD AND JOHN STRAIN, *A fast algorithm for the evaluation of heat potentials*, Communications on Pure and Applied Mathematics, XLIII (1990), pp. 949–963.

[13] ——, *The fast Gauss transform*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 79–94.

[14] LESLIE GREENGARD AND XIAOBAI SUN, *A new version of the fast Gauss transform*, Documenta Mathematica, III (1998), pp. 575–584.

[15] JUNE-YUB LEE AND LESLIE GREENGARD, *Accelerating the nonuniform fast Fourier transform*, SIAM Review, 46 (2004), pp. 443–454.

[16] PIESSENS R AND BRANDERS M, *Numerical-solution of integral-equations of mathematical physics using Tschebyscheff polynomials*, Journal of Computational Physics, 21 (1976), pp. 178–196.

[17] THEODORE J. RIVLIN, *Chebyshev Polynomials*, Wiley–Interscience, 1990.

[18] J. A. SETHIAN AND J. STRAIN, *Crystal growth and dendritic solidification*, Journal of Computational Physics, 98 (1992).

[19] EUGENE SORETS, *Fast Fourier transforms of piecewise constant functions*, Journal of Computational Physics, 116 (1995), pp. 369–379.

[20] JOHN STRAIN, *Fast potential theory. II. Layer potentials and discrete sums*, Journal of Computational Physics, 99 (1992), pp. 251–270.

[21] ——, *Fast adaptive methods for the free–space heat equation*, SIAM Journal on Scientific Computing, 15 (1994), pp. 185–206.

[22] XIAOBAI SUN AND YUJUAN BAO, *A Kronecker product representation of the fast Gauss transform*, SIAM Journal on Matrix Analysis and Applications, 24 (2002), pp. 768–786.

[23] SHRAVAN K. VEERAPANENI AND GEORGE BIROS, *A high-order fast solver for the heat equation in 1D domains with moving boundaries (submitted).*

[24] CLENSHAW C. W AND A. R. CURTIS, *A method for numerical integration on an automatic computer*, Numerische Mathematik, 2 (1960), pp. 197–205.

[25] JET WIMP, *Computation with Recurrence Relations*, Pitman Advanced Pub. Program, 1984.