FAST RADIAL BASIS FUNCTION INTERPOLATION VIA PRECONDITIONED KRYLOV ITERATION

NAIL A. GUMEROV AND RAMANI DURAISWAMI
UNIVERSITY OF MARYLAND, COLLEGE PARK

Abstract. We consider a preconditioned Krylov subspace iterative algorithm presented by Faul et al. (IMA Journal of Numerical Analysis (2005) 25, 1–24) for computing the coefficients of a radial basis function interpolant over N data points. This preconditioned Krylov iteration has been demonstrated to be extremely robust and the iteration rapidly convergent. However, the iterative method has several steps whose computational and memory costs scale as $O(N^2)$, both in preliminary computations that compute the preconditioner and in the matrix-vector product involved in each step of the iteration. We effectively accelerate the iterative method to achieve an overall cost of $O(N \log N)$. The matrix vector product is accelerated via the use of the Fast Multipole Method. The preconditioner requires the computation of a set of closest points to each point. We develop an $O(N \log N)$ algorithm for this step as well. Results are presented for multiquadric interpolation in $\mathbb{R}^2$ and biharmonic interpolation in $\mathbb{R}^3$. A novel FMM algorithm for the evaluation of sums involving multiquadric functions in $\mathbb{R}^2$ is presented as well.

Key words. Radial Basis Function Interpolation, Preconditioned Conjugate Gradient, Cardinal Function Preconditioner, Computational Geometry, Fast Multipole Method

AMS subject classification. 65D07, 41A15, 41A58

1. Introduction. Interpolation of scattered data of various types using radial-basis functions has been demonstrated to be useful in different application domains [22, 7, 8, 26]. Much of this work has been focused on the use of polyharmonic and multiquadric radial-basis functions $\left[\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}\right]$ that fit data as

$$s(\mathbf{x}) = \sum_{j=1}^{N} \lambda_j \phi(|\mathbf{x} - \mathbf{x}_j|) + P(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d \tag{1.1}$$

where several classes of radial basis functions may be chosen for ϕ. Popular choices include

$$\phi(r) = \begin{cases} r & \text{biharmonic } (\mathbb{R}^3) \\ r^{2n+1} & \text{polyharmonic } (\mathbb{R}^3) \\ \left(r^2 + c^2\right)^{1/2} & \text{multiquadric } (\mathbb{R}^d) \end{cases},$$

and λ_j are coefficients while $P(\mathbf{x})$ is a global polynomial function of total degree at most $K-1$. The coefficients $\{\lambda\}$ and the polynomial $P(\mathbf{x})$ functions are chosen to satisfy the N fitting conditions

$$s(\mathbf{x}_j) = f(\mathbf{x}_j), \quad j = 1, \ldots, N$$

and the constraints

$$\sum_{j=1}^{N} \lambda_j Q(\mathbf{x}_j) = 0,$$

for all polynomials Q of degree at most $K-1$.

Let $\{P_1, P_2, ..., P_k\}$ be a basis for the polynomials of total degree $K-1$. Then the corresponding system of equations that needs to be solved to find the coefficients

λ and the polynomial P, are

$$\begin{pmatrix} \Phi & P^T \\ P & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ b \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix},$$

where

$$\Phi_{ij} = \phi\left(\mathbf{x}_i - \mathbf{x}_j\right),\ i,j = 1,\ldots,N;\ \ P_{ij} = P_j\left(\mathbf{x}_i\right),\ i = 1,\ldots,N;\ j = N{+}1,\ldots,N{+}K.$$

Usually the degree K of the polynomial is quite small, and quite often it is 1, in which case the polynomial $P(\mathbf{x})$ is just a constant, and this is the case that will be considered in the sequel. Micchelli [23] has shown that this system of equations is nonsingular.

For large N, as is common in many applications, the computational time and memory required for the solution of the fitting equations is a significant bottleneck. Direct solution of these equations requires $O\left(N^3\right)$ time and $O\left(N^2\right)$ memory. Iterative approaches to solving this set of symmetric equations require evaluations of dense matrix vector products $\Phi\lambda$ which have a $O(N^2)$ computational cost and require $O\left(N^2\right)$ memory. Furthermore, for rapid convergence of the iteration, the system must be preconditioned appropriately.

Considerable progress in the iterative solution of multiquadric and polyharmonic RBF interpolation was made with the development of preconditioners using cardinal functions based on a subset of the interpolating points. This approach was first proposed by Beatson and Powell [1]. Various versions of the idea have since been tried including those based on choosing a set of data near each point, a mix of near and far points, and via modified approximate cardinal functions that have a specified decay behavior [3]. An alternative group of methods that attempt to speed up the algorithm via domain decomposition have also been developed [4].

Recently, an algorithm was published by Faul et al. [13], which uses the cardinal function idea, and in which for each point a carefully selected set of q points was used to construct the preconditioner. In informal tests we found that this algorithm appears to achieve a robust performance, and was able to interpolate large problems in a few iterations, where other cardinal function based strategies sometimes took a large number of steps. In subsequent discussions we will refer to this algorithm as FGP05. This algorithm reduces the number of iterations required; however, its cost remains at $O(N^2)$, and it requires $O(N^2)$ memory. The purpose of this paper is the acceleration of this algorithm.

Similarly to many other iterative techniques the FGP05 algorithm can be accelerated by the use of the Fast Multipole Method (FMM) for the matrix vector product required at each step, making the cost of each iteration $O\left(N \log N\right)$ or so [14]. While the possibility of the use of the FMM was mentioned in [13], results were not reported. Fast multipole methods have been developed for various radial basis kernels including the multiquadric in $\mathbb{R}^d$ [11], the biharmonic and polyharmonic kernels in $\mathbb{R}^3$ [18] and the polyharmonic kernels in $\mathbb{R}^4$ [6]. Use of the FMM can thus reduce the cost of the matrix vector multiply required in each iteration of the FGP05 algorithm to $O(N \log N)$.

However, there is a further barrier to achieving a truly fast algorithm using the FGP05 algorithm, as there is a precomputation stage that requires the construction of approximate cardinal function interpolants centered at each data point and q of its carefully selected neighbors ($q \ll N$). These interpolants are used to provide a set of directions in the Krylov space for the iteration. The procedure for these preliminary

calculations reported in [13] has a formal complexity of $O(N^2)$. Progress towards reducing the cost of this step to $O(N \log N)$ was reported by Goodsell in two dimensions [17]. However, his algorithm relied on Dirichlet tessellation of the plane, and does not extend to higher dimensions.

In this paper we present a modification to this preliminary set up stage, use the FMM for the matrix vector product, and reduce the complexity of the overall algorithm. Section 2 provides some preliminary information on preconditioning with cardinal functions, details of the Faul et al. algorithm, and some details of the Fast Multipole Method that are necessary for the algorithms developed in this paper. Section 3 describes the two improvements that let us achieve improved performance in our modified algorithm: finding the q closest points to each point in the set in $O(N \log N)$ time and use of the fast multipole method. Since the latter method is well discussed in other literature, we only provide some details that are unique to our implementation. In particular we present novel FMM algorithms for the multiquadric (and the inverse multiquadric) in $\mathbb{R}^2$. Section 4 provides results of the use of the algorithm. Section 5 concludes the paper.

2. Background.

2.1. Preconditioning using approximate cardinal functions. A long sequence of papers [1, 3, 4] use approximate cardinal functions to create preconditioners for RBF fitting. The basic iterative method consists at updating at each iteration step the approximation to the function f, $s^{(k)}$

$$s^{(k)}(\mathbf{x}) = \sum_{j=1}^{N} \lambda_j^{(k)} \phi(|\mathbf{x} - \mathbf{x}_j|) + a^{(k)}, \quad \mathbf{x} \in \mathbb{R}^d, \tag{2.1}$$

by specifying new coefficients $\lambda_j^{(k)}$, $j = 1, \ldots, N$ and the constant $a^{(k)}$. The residual at step k is obtained as

$$r_i^{(k)} = f(\mathbf{x}_i) - s^{(k)}(\mathbf{x}_i), \quad i = 1, \ldots, N,$$

at the data points $\mathbf{x}_i$.To establish notation, and following FGP05, we denote as s^* the function that arises at the end of the iteration process.

The cardinal or Lagrange functions corresponding to any basis play a major role in interpolation theory. The cardinal function χ_i corresponding to a particular data point i, satisfy the Kronecker conditions

$$z^l(\mathbf{x}_j) = \delta_{lj}, \quad j, l = 1, \ldots, N, \tag{2.2}$$

i.e., $z^{(l)}$ takes on the value unity at the lth node, and zero at all other nodes. It can itself be expressed in terms of the radial basis functions

$$z^l(\mathbf{x}) = \sum_{j=1}^{N} \zeta_j^l \phi(|\mathbf{x} - \mathbf{x}_j|) + a_l^{(k)}, \tag{2.3}$$

where the ζ_j^i are the interpolation coefficients. If we now form a matrix of all the interpolation coefficients $T_{ji} = \zeta_j^i$, then we have

$$\begin{pmatrix} \Phi & \mathbf{1}^T \\ \mathbf{1} & 0 \end{pmatrix} \begin{pmatrix} T \\ \mathbf{a} \end{pmatrix} = \begin{pmatrix} I \\ 0 \end{pmatrix}, \tag{2.4}$$

where I is the identity matrix, and $\mathbf{1}$ is a row vector of size N containing ones, and $\mathbf{a}$ a row vector containing the coefficients a_l. Thus the matrix of interpolation coefficients corresponding to the cardinal functions is the inverse of the matrix Φ, and if these coefficients were known, then it would be easy to solve the system. On the other hand, computing each of these sets of coefficients for a single cardinal function χ_i (corresponding to a column of T) by a direct algorithm is an $O\left(N^3\right)$ procedure, and there are N of these functions.

The approach pursued by Beatson and Powell [1] was to compute *approximations* to these cardinal functions (so that a column $\zeta_j^{(l)}$) is approximated by a sparse set of coefficients, say q in number, corresponding to the cardinality condition being satisfied at the point l and $q-1$ other chosen points. Then the matrix $\hat{T}$ corresponding to the approximate cardinal function coefficients is used as a preconditioner in an appropriate Krylov subspace algorithm. In this case the complexity of the preconditioning step is $O\left(Nq^3\right)$ and for $q \ll N$,and independent of N, this strategy, provided it converges rapidly, can lead to a fast algorithm.

Early versions of these algorithm were applied to relatively modest sized problems and the points at which the cardinality conditions were satisfied were chosen as near neighbors of the point corresponding to the row index. This strategy was observed to not work for larger sets, and various modifications were tried. The algorithm of Faul et al. [13] presents a variation that works on all tried data sets. It is described in the next section.

REMARK 2.1. *The FMM applies to matrices whose entries can be characterized as radial basis functions. For smoothly varying functions, the approximate cardinal functions are likely to be good approximations to the true cardinal functions. As such, the idea of using approximate cardinal functions to construct preconditioners may be of practical use in a large number of problems where the FMM can be used to accelerate matrix vector products, by using the radial basis function itself, or in the singular case, some appropriately mollified version of the FMM radial basis function.*

2.2. Algorithm of Faul et al.

2.2.1. Choice of points for fitting cardinal functions. The FGP05 algorithm implements a Krylov method for the solution of Eq. (1.1). In this algorithm the point sets on which the approximate cardinal functions are constructed are chosen in a special way, so that a rapidly converging preconditioned conjugate gradient algorithm is achieved. This algorithm was extensively tested in [13], and its convergence properties established theoretically. We also tested our implementation of it practically, and found it to exhibit remarkably fast and robust convergence.

Each row of the matrix has associated with it a particular set of q or fewer points. The $N+1 \times N+1$ system of equations 2.4 can be expressed as an equivalent $N-1 \times N-1$ system, by eliminating λ_n from the system using

$$\lambda_n = -\sum_{j=1}^{N-1} \lambda_j.$$

To build a preconditioner we need to consider $N-1$ sets of approximate RBF interpolants and associated points, $\mathcal{L}_l$, $l = 1, \ldots, N-1$, which we call L-sets. Each set contains the point corresponding to the row index we are working with and other points. To choose the first of these sets, using the inter point distances, the points that are closest to each other chosen, and one of these points is marked as the first

point; $q-2$ further closest neighbors of the marked points are chosen, and the marked point is removed from consideration. In the remaining $N-1$ points, again the closest pair of points is chosen, one of these points is marked, and $q-2$ further closest neighbors of the marked points are chosen. At each step a marked point is selected from the closest pair, a further set of closest points chosen, and a set for the marked point built. In this way the algorithm proceeds to build sets of q points, and the procedure continues till only q points remain, which forms the $(N-q)$th set. A further $q-2$ sets of points are constructed from these points by taking $q-1$, $q-2$, $\ldots$, 2 points from the remaining points, and in each case setting one of these points as the marked point, and removing it. In this way a total of $N-1$ groups of points are established.

The procedure described above, though indicated as preferred, was not what is ultimately implemented in FGP05. Rather an approximate version was used. However, the procedure is nonetheless $O(N^2)$, as it involves computation of pairwise distances between points. A further crucial aspect of the FGP05 procedure is that at each step the algorithm requires deletion of points being considered making the process of constructing the list of closest points a dynamic one.

Now using these point sets, radial-basis function fits of the Lagrange function over these point sets, of the form

$$\hat{z}_l(\mathbf{x}) = \sum_{j\in\mathcal{L}^l} \zeta_j^{(l)} \phi(\mathbf{x}-\mathbf{x}_j) + const., \quad \hat{z}_l(\mathbf{x}_j) = \delta_{lj}, \quad l = 1,\ldots,N-1,$$

are constructed. The coefficients of the Lagrange function fit $\zeta_j^{(l)}$ are used to define the search directions in a conjugate gradient algorithm that is described below. In this algorithm, the function $\tilde{z}_l$ is defined as $\hat{z}_l$ with the constant part dropped

$$\tilde{z}_l(\mathbf{x}) = \sum_{j\in\mathcal{L}^l} \zeta_j^{(l)} \phi(\mathbf{x}-\mathbf{x}_j).$$

By construction

$$\sum_{j\in\mathcal{L}^l} \zeta_j^{(l)} = 0.$$

2.2.2. Semi norm and inner product. Given functions s and t at the data interpolation locations $\{\mathbf{x}\}$, and having these functions interpolated using radial basis functions as

$$s(\mathbf{x}) = \sum_{j=1}^{N} \lambda_j \phi(\mathbf{x}-\mathbf{x}_j) + \alpha, \qquad t(\mathbf{x}) = \sum_{i=1}^{N} \mu_i \phi(\mathbf{x}-\mathbf{x}_i) + \beta;$$

FGP05 define the following semi-norm

$$\|s\|_\phi = -\left(\lambda^T \Phi \lambda\right)^{\frac{1}{2}}, \qquad \|t\|_\phi = -\left(\mu^T \Phi \mu\right)^{\frac{1}{2}}$$

and inner product between two functions as

$$\begin{aligned}\langle s,t\rangle_\phi &= \frac{1}{2}\left(\|s+t\|_\phi^2 - \left(\|s\|_\phi^2 + \|t\|_\phi^2\right)\right) = -\frac{1}{2}\left((\lambda+\mu)^T \Phi (\lambda+\mu) - \left(\lambda^T\Phi\lambda\right) - \left(\mu^T\Phi\mu\right)\right) \\ &= -\frac{1}{2}\left(\lambda^T\Phi\mu + \mu^T\Phi\lambda\right) = -\sum_{j=1}^{N}\sum_{i=1}^{N} \lambda_i \phi(\mathbf{x}_i-\mathbf{x}_j)\mu_j = -\sum_{i=1}^{N} \lambda_i t(\mathbf{x}_i) = -\sum_{j=1}^{N} \mu_j s(\mathbf{x}_j).\end{aligned}$$

If the function values at the interpolation points are known, these inner products can be computed without any matrix-vector products, i.e., in $O(N)$ operations.

2.2.3. Krylov subspace. The Krylov subspace considered is that induced by the operator Ξ acting on functions s that are expressible via a RBF expansion. This operator is defined using the $N-1$ approximations to the cardinal functions centered at the $N-1$ points that are constructed $\tilde{z}_l$, and is given as

$$(\Xi s)(\mathbf{x}) = \sum_{l=1}^{N-1} \frac{\langle \tilde{z}_l, s \rangle_\phi}{\|\tilde{z}_l\|_\phi^2} \tilde{z}_l(\mathbf{x}).$$

2.2.4. The preconditioned conjugate gradient iteration. Let s^* be the function that is sought as the solution. Define an initial guess $s^{(1)}$ as

$$\lambda_j^{(1)} = 0, \quad j = 1, \ldots, N; \qquad \alpha_1 = \frac{1}{2}(\min(f) + \max(f)),$$

which leads to the initial residual

$$r_i^{(k)} = f_i - s^{(k)}(\mathbf{x}_i), \qquad \mathbf{r} = \{r_i\}. \tag{2.5}$$

Two functions expressed as rBF sums are used in the iteration:

$$t^{(k)} = \sum_{j=1}^{N} \tau_j^{(k)} \phi(|\mathbf{x} - \mathbf{x}_j|), \quad \sum_{j=1}^{N} \tau_j^{(k)} = 0,$$
$$d^{(k)} = \sum_{j=1}^{N} \delta_j^{(k)} \phi(|\mathbf{x} - \mathbf{x}_j|), \quad \sum_{j=1}^{N} \delta_j^{(k)} = 0.$$

The first function, t, is constructed using the RBF approximation to the cardinal functions on the specially chosen FGP05 points, and the value of the residual at the current iteration at these points:

$$\mu_l^{(k)} = \frac{1}{\zeta_{ll}} \sum_{i \in \mathcal{L}_l} \zeta_{li} r_i^{(k)}, \qquad \tau_j^{(k)} = \mu_l^{(k)} \zeta_{lj}.$$

It is shown that

$$t^{(k)} = \Xi\left(s^* - s^{(k)}\right),$$

while the second function, $d^{(k)}$ satisfies

$$d^{(1)} = t^{(1)}, \quad d^{(k)} = t^{(k)} - \frac{\left\langle t^{(k)}, d^{(k-1)} \right\rangle_\phi}{\left\langle d^{(k-1)}, d^{(k-1)} \right\rangle_\phi} d^{(k-1)}.$$

Because of this definition, the conjugacy criterion holds

$$\left\langle d^{(k)}, d^{(k-1)} \right\rangle_\phi = \left\langle t^{(k)}, d^{(k-1)} \right\rangle_\phi - \frac{\left\langle t^{(k)}, d^{(k-1)} \right\rangle_\phi}{\left\langle d^{(k-1)}, d^{(k-1)} \right\rangle_\phi} \left\langle d^{(k-1)}, d^{(k-1)} \right\rangle_\phi = 0.$$

The solution is advanced by a step length γ_k along the conjugate direction $d^{(k)}$, so that where

$$\lambda_j^{(k+1)} = \lambda_j^{(k)} + \gamma_k d^{(k)}, \qquad \gamma_k = \frac{\sum_{i=1}^{N} \delta_i^{(k)} r_i^{(k)}}{\sum_{i=1}^{N} \delta_i^{(k)} d^{(k)}(\mathbf{x}_i)}$$

while the constant is updated to minimize the maximum element in the new residual vector. The choice γ_k can be shown to be minimize $\left\|s^* - s^{(k+1)}\right\|_\phi$ along d^k.

Crucial to the implementation is the construction of the approximate cardinal functions. It is indicated in [13] that a Fortran implementation is available from Prof. Powell. Our Matlab implementation of the FGP05 algorithm is available online at http://www.umiacs.umd.edu/~ramani/fmm/FGP_RBF.

2.3. The fast multipole method for accelerating matrix vector products. There are several papers and books that introduce the fast multipole method in details [14, 15, 2], and our purpose is not to reproduce that literature here. However, certain details of geometric data structures in the fast multipole method will be important in the initial computations necessary for the preconditioner. Our implementation of the FMM for polyharmonic radial basis functions is described in [18], where we also provided operational and memory complexity.

The algorithm consists of two main parts: the preset step, which includes building the FMM data structure (building and storage of the neighbor lists, etc.) and precomputation and storage of all translation data. The data structure is generated using the bit interleaving technique described in [20], which enables spatial ordering, sorting, and bookmarking. While the FMM algorithm is designed for two independent data sets (N arbitrary located sources and M arbitrary evaluation points), for the RBF fitting we will have the same source and evaluation sets of length N. For a problem size N, the cost of building the data structure based on spatial ordering is $O(N \log N)$, where the asymptotic constant is much smaller than the constants in the $O(N)$ asymptotics of the main algorithm. The number of levels could be arbitrarily set by the user or found automatically based on the clustering parameter (the maximum number of sources in the smallest box) for optimization of computations of problems of different size.

Of particular use in developing the $O(N \log N)$ algorithm for the preliminary steps are the octree data structures that are employed in the FMM. A good introduction to these are provided in Greengard's thesis [14]. Fast algorithms for computing neighbor lists, point assignment to boxes and other necessary operation in the FMM using bit-interleaving and de-interleaving operations are provided in [20].

3. Fast algorithms. The original form of the FGP05 algorithm has a complexity of $O(N^2)$ due to two reasons: first, the step of setting the data structure requires $O(N^2)$ operations due to pairwise distance search to form $O(N)$ subsets of length $O(q)$ for approximation of cardinal functions centered at each point, and, second, the step of matrix vector multiplication of $O(N \times N)$ matrix by a $O(N)$ vector, also requires $O(N^2)$ operations. As mentioned before the second step can be done using the FMM for $O(N \log N)$ operations. In the subsections below we describe efficient $O(N \log N)$ algorithms for both steps, which bring the FGP iterative method to $O(N \log N)$ complexity.

3.1. Data structure for the FGP05 algorithm. While the problem of determining the closest points in a data set is a relatively well studied problem in computational geometry, no standard computational geometry algorithms that solve the particular problem required in the FGP05 algorithm. As mentioned earlier, a reason for the dramatic performance of the algorithm, as discussed in [13] is the careful selection of the points over which the cardinal function is selected. As such a selection might be useful in the construction of preconditioners with other radial basis function applications with the FMM, we provide details of this algorithm below.

3.1.1. Problem statement. While the selection of the point sets has been described above in words, we will provide a more precise mathematical statement here, mainly in the interest of establishing notation.

Given a set $\mathbb{X}$ of N points in d dimensional Euclidean space, $\mathbf{x}_i \in \mathbb{R}^d$, $i = 1, ..., N$ and $q < N$ build $N-1$ sets $\mathbb{L}_j \subset \mathbb{X}$, $j = 1, ..., N-1$ satisfying the following conditions:

- For $j = 1, ..., N-q+1$ the sets $\mathbb{L}_j$ consist of q points $\mathbf{x}_{i_{j1}}, ..., \mathbf{x}_{i_{jq}} \in \mathbb{X}$ while for $j = N-q+2, ..., N-1$ the sets $\mathbb{L}_j$ consist of $N-j+1$ points, $\mathbf{x}_{i_{j1}}, ..., \mathbf{x}_{i_{j,N-j+1}} \in \mathbb{X}$. The number of entries in the set is called its "power", so $Pow(\mathbb{L}_j) = q$ for $j = 1, ..., N-q+1$ and $Pow(\mathbb{L}_j) = N-j+1$ for $j = N-q+2, ..., N-1$.
- Each set $\mathbb{L}_j$ is associated with a point $\mathbf{x}_{i_{j1}} \in \mathbb{L}_j$, called the set center. All set centers are different, $i_{j1} \neq i_{k1}$, for $k \neq j$. Further we introduce sets of set centers $\mathbb{C}_j = \left\{\mathbf{x}_{i_{11}}, ..., \mathbf{x}_{i_{j1}}\right\}$, $j = 1, ..., N-1$, $Pow\left(\mathbb{C}_j\right) = j$.
- The set $\mathbb{L}_1$ consists of points $\mathbf{x}_{i_{11}}, ..., \mathbf{x}_{i_{1q}} \in \mathbb{X}$ such that

$$|\mathbf{x}_{i_{11}} - \mathbf{x}_{i_{12}}| = \min_{\mathbf{x}_i, \mathbf{x}_k \in \mathbb{X}, i \neq k} |\mathbf{x}_i - \mathbf{x}_k|, \tag{3.1}$$

$$|\mathbf{x}_{i_{11}} - \mathbf{x}_{i_{1l}}| \leqslant \min_{\mathbf{x}_k \in \mathbb{X} \setminus \mathbb{L}_1} |\mathbf{x}_{i_{11}} - \mathbf{x}_k| \quad \text{for} \quad l = 3, ..., q. \tag{3.2}$$

  In other words, the distance between the center of this set and the second element is minimal among all pairwise distances in $\mathbb{X}$. Besides the set center the set consists of the $q-1$ closest points to the set center.
- The set $\mathbb{L}_j$, $j = 1, ..., N-1$ consists of points $\mathbf{x}_{i_{j1}}, ..., \mathbf{x}_{i_{jp}} \in \mathbb{X} \setminus \mathbb{C}_{j-1}$, $p = Pow(\mathbb{L}_j)$, such that

$$\left|\mathbf{x}_{i_{j1}} - \mathbf{x}_{i_{j2}}\right| = \min_{\mathbf{x}_i, \mathbf{x}_k \in \mathbb{X} \setminus \mathbb{C}_{j-1}, i \neq k} |\mathbf{x}_i - \mathbf{x}_k|, \tag{3.3}$$

$$\left|\mathbf{x}_{i_{j1}} - \mathbf{x}_{i_{jl}}\right| \leqslant \min_{\mathbf{x}_k \in \mathbb{X} \setminus (\mathbb{L}_j \cup \mathbb{C}_{j-1})} \left|\mathbf{x}_{i_{j1}} - \mathbf{x}_k\right| \quad \text{for} \quad l = 3, ..., q. \tag{3.4}$$

  In other words, the distance between the center of this set and the second element is the minimal among all pairwise distances in $\mathbb{X}$ excluding the centers of the sets $\mathbb{L}_1, ..., \mathbb{L}_{j-1}$.Besides the set center the set consists of $q-1$ closest points to the set center for $j = 1, ..., N-q+1$ and $N-j+1$ closest points to the set center for $j = N-q+2, ..., N-1$.

REMARK 3.1. *If we construct the complete inter-point distance matrix (requiring computation of $N(N-1)/2$ distances), this problem can be solved simply. However this requires $O\left(N^2\right)$ time and memory.*

3.1.2. General algorithm. There exist fast closest neighbor search algorithms in d dimensional space of complexity $O(\log N)$ per query. These algorithms are based on hierarchical space subdivision using k-d trees or similar ideas [24]. The cost of building a tree data structure is $O(N \log N)$. Furthermore, an initial spatial ordering of the data requires $O(N \log N)$ operations. However, these structures do not enable easy deletion of points.

Deletion is enabled in the heap data structure, which also requires $O(N \log N)$ construction cost [10]. Organization of the data using heaps enables $O(\log N)$ insertion/deletion operations for a single entry. However even with these preliminary steps the design of the actual algorithm requires several additional data structures which take care of the dynamic nature of the data (the deletions of the set center at each step). These additional structures account for the modifications of the initial set $\mathbb{X}$ due to deletion operations and enable $O(\log N)$ complexity for any operation on the

modified data set. Such a performance is needed for these operations, as they must be repeated $O(N)$ times as the cardinal point sets are built for the $N-1$ centers.

We assume the initial set $\mathbb{X}$ consists of points randomly distributed over the computational domain, which can be enclosed by a d dimensional cube of side D. In practice this set may not be random, as the data may come imbued with a certain order because of the way it was generated. Such an order may not be desirable for the performance of the preconditioner. This is why in [13] the authors suggested an initial random permutation of all initial data. We follow this approach and to avoid additional complexity in notations we assume that such a random permutation is performed as a first step and points in $\mathbb{X}$ are already arranged in random order.

Let $\mathbb{X}'$ be a spatially ordered set of points from $\mathbb{X}$. Spatial ordering in d dimensions, when d is not large (say $d = 2$ or $d = 3$, which is the case for most applications of the FMM) can be performed efficiently using the bit-interleaving technique. We use the same technique to generate the FMM quadtree or octree data structure. A detailed description of the bit interleaving technique can be found in [21, 20]. The ordered set $\mathbb{X}'$ can be characterized by a permutation index Π. So if $\mathbf{x}_i$ is the i th element of $\mathbb{X}$, then the jth element of $\mathbb{X}'$, where $j = \Pi(i)$ corresponds to the same point. Thus, as Π is available there is no need for additional memory to store $\mathbb{X}'$. The spatially ordered list is needed to provide the following operations with a complexity $O(\log N)$ for any point:

1. deletion, and,
2. q-closest point search in the subset of remaining points after the deletion of previously considered point centers.

These procedures are described below separately.

The second virtual set which we introduce is $\mathbb{H}$, which consists of the elements of $\mathbb{X}$ partially ordered according to some ranking criterion $\mathcal{R}$ using the heap data structure [10]. This set also can be characterized completely by the permutation index H. We will operate only with first k elements of $\mathbb{H}$ which thus form a subset $\mathbb{H}^k \subset \mathbb{H}$. The top ($j = 1$) element of $\mathbb{H}^k$ has the highest rank. Elements below this element are arranged in a binary tree, with each leaf being outranked by its parent ("heap property"). The heap structure is needed to provide the following operations of complexity $O(\log N)$:

1. find the highest ranked element;
2. delete the highest ranked element;
3. change the rank of an element.

After performing any of these operations, the heap structure (characterized by a permutation index H) can be recovered for a cost of $O(\log N)$ operations. The ranking which we introduce can be applied only to elements of $\mathbb{H}^k$. The rank $\mathcal{R}_i$ of element $\mathbf{x}_i \in \mathbb{H}^k$ is the nearness of the point to the set center. So the the closest neighbor to the top element of $\mathbb{H}^k$ is closer (or at the same distance) than the closest neighbors to other points from $\mathbb{H}^k$.

We need to introduce two more lists or indices. The first list can be denoted as C and $C(i)$ is the index of the closest neighbor to $\mathbf{x}_i$ if both these points are in $\mathbb{H}^k$. The second list can be denoted as C^*, where

$$C(C^*(j,i)) = i \quad \text{for} \quad j = 1, ..., M_i^k. \tag{3.5}$$

In some sense this mapping is inverse to $C(i)$. However, as the closeness relationship is not unique, a point $\mathbf{x}_i$ can be the closest neighbor to several points. Since this mapping is not one-to-one, $C^*(:, i)$ provides the indices of all M_i^k points to which

$\mathbf{x}_i$ is the closest neighbor. To bound the size of the array (and the memory and operations), we need to bound the quantity M_i^k. There exists a strong bound for $M_i^k \leqslant K_{(d)}$, which shows that for a given distribution of points in $\mathbb{R}^d$ the number of entries in M is $O(N)$. We present this result in the form of the following theorem.

THEOREM 3.2. *Let $\mathbb{X}$ be a set of N points in $\mathbb{R}^d$, $(d \geqslant 1)$, with the elements denoted $\mathbf{x}_i \in \mathbb{R}^d$, $i = 1, ..., N$. Let C be a list of closest neighbors to points $i = 1, ..., N$, and let M_i be the number of entries of point i in this list. Then $M_i \leqslant K_{(d)}$, where $K_{(d)}$ is the kissing number for dimensionality d*[1].

Proof. Without any loss of generality we can assign index $i = 1$ to a given point and assign indices $2, ..., M+1$ to points for which this point is the closest neighbor. Denote $D_j = |\mathbf{x}_j - \mathbf{x}_1|$ and $D_{\max} = \max_{j=2,...,M+1} D_j$. Consider then a sphere of radius $D_{\max}$ centered at $\mathbf{x}_1$. For each point $\mathbf{x}_j$ we will correspond point $\mathbf{y}_j$ on the sphere surface: $\mathbf{y}_j = \mathbf{x}_1 + (D_{\max}/D_j)(\mathbf{x}_j - \mathbf{x}_1)$, $j = 2, ..., M+1$. Let us show that $|\mathbf{y}_j - \mathbf{y}_k| \geqslant D_{\max}$ for $j, k = 2, ..., M+1$, $j \neq k$. Indeed, since for these j and k we have $|\mathbf{x}_j - \mathbf{x}_k| \geqslant \max(D_j, D_k)$ (otherwise $\mathbf{x}_1$ is not the closest point to $\mathbf{x}_j$ or $\mathbf{x}_k$) we obtain

$$2(\mathbf{x}_j - \mathbf{x}_1)\cdot(\mathbf{x}_k - \mathbf{x}_1) = |\mathbf{x}_j|^2 + |\mathbf{x}_k|^2 - |\mathbf{x}_j - \mathbf{x}_k|^2 \leqslant D_j^2 + D_k^2 - [\max(D_j, D_k)]^2 \leqslant D_j D_k.$$

Therefore,

$$\begin{aligned}|\mathbf{y}_j - \mathbf{y}_k|^2 &= |\mathbf{y}_j - \mathbf{x}_1|^2 + |\mathbf{y}_k - \mathbf{x}_1|^2 - 2(\mathbf{y}_j - \mathbf{x}_1)\cdot(\mathbf{y}_k - \mathbf{x}_1) \\ &= 2D_{\max}^2 - \frac{D_{\max}^2}{D_j D_k} 2(\mathbf{x}_j - \mathbf{x}_1)\cdot(\mathbf{x}_k - \mathbf{x}_1) \geqslant 2D_{\max}^2 - D_{\max}^2 = D_{\max}^2.\end{aligned}$$

Now we see that there are M points on the sphere surface the pairwise distance between which is not less than the radius of this sphere. Introducing spheres of radius $R = D_{\max}/2$ centered at $\mathbf{x}_1$and $\mathbf{y}_j$, $j = 2, ..., M+1$ we can see that this is equivalent to placing of M spheres of radius R around the sphere of the same radius centered at $\mathbf{x}_1$. It is well known that this number cannot exceed the kissing number. □

REMARK 3.3. *The exact kissing number is well known for dimensionalities $d \leqslant 9$ and $d = 24$, while for other dimensionalities bounds for $K_{(d)}$ are established. Particularly, we have $K_{(1)} = 2$, $K_{(2)} = 6$, $K_{(3)} = 12$.*

The procedure of building sets satisfying the above requirements then can be reduced to procedures of q-closest neighbor search and deletion of found set centers from the data structures.

Initialization procedure.

- Build a hierarchical space subdivision, using the 2^d trees, and initialize data structure for neighbor search and deletion procedures;
- Set $k = N$; $M_i^k = 0$, $i = 1, ..., N$.
- For $j = 1 : N$
  1. Get the closest neighbor for point j (let this be i). This is provided by the q-neighbor search procedure for $q = 1$.
  2. Set $C(j) = i$, $d(j) = |\mathbf{x}_j - \mathbf{x}_i|$, $M_i^k = M_i^k + 1$; $C^*\left(M_i^k, i\right) = j$.
- Based on ranking $\mathcal{R} = -d$ build the heap permutation index H.

Main algorithm.

- For $i = 1 : N - 1$
  1. Find the size of set $\mathbb{L}_i$: $q' = \min(q, N - i + 1)$.

[1] The kissing number is defined as the maximum number of unit spheres which can surround a given unit sphere without intersection (excluding the touching of points on the boundary) [27].

2. Find the index of the set center: $j = H(1)$. This is the top ranked point in the heap. Find the index of the closest to j neighbor: $m = C(j)$.
3. Find all the entries to $\mathbb{L}_i$. The first entry is $\mathbf{x}_j$, and the rest are $q' - 1$ closest neighbors, provided by the q-neighbor search procedure for $q = q' - 1$.
4. Delete point j from the hierarchical data structure for neighbor search. Delete point j from the heap. The last procedure sets the heap length to $k = k - 1$ and updates the heap permutation index H.
5. Modify M_m^k and $C^*\left(M_m^k, m\right)$. As m is the closest to j it is listed in $C^*(:, m)$. So set $M_m^k = M_m^k - 1$ and squeeze $C^*(:, m)$ to exclude j (this will change the indexing in $C^*(:, m)$).
6. Update lists C and C^* and the heap permutation index H as follows
7. For $n = 1 : M_j^k$
   (a) Get the closest neighbor, s, for point $p = C^*(n, j)$. This is provided by the q-neighbor search procedure for $q = 1$.
   (b) Set $C(p) = s$, $d(p) = |\mathbf{x}_p - \mathbf{x}_s|$, $M_s^k = M_s^k + 1$; $C^*\left(M_s^k, s\right) = p$.
   (c) Update the heap permutation index for new rank of point p, $\mathcal{R}(p) = -d(p)$.

3.1.3. Data structure for deletion operation and q-closest neighbor search using 2^d-trees. For fast neighbor search we employ 2^d-tree data structures, which for $d = 2$ and $d = 3$ are known as the quadtree and octree, respectively [25]. In the 2^d-tree we have one box at level 0 (which is a cube including all points from $\mathbb{X}$), 2^d cubical boxes at level 1 obtained by division by half of the initial (parent) box along each dimension, and so on up to level $l_{\max}$, where we have $2^{l_{\max} d}$ boxes. As the data are sorted using the bit-interleaving techniques they can be easily boxed due to the hierarchical numbering system [20], which also provides a fast way to obtain parents and neighbors for all boxes. Of course, not all the boxes may be occupied, and the "empty" boxes are not listed in the data structure. Selection of $l_{\max}$ is dictated by the parameter q and one of the possibilities to determine $l_{\max}$ is to define it as the level at which the finest box contains not more than $q + 1$ points. For every box we maintain a list of points which reside in this box. This list is dynamic and provided by an additional data structure described below. The same relates to the list of non-empty boxes neighboring to the given box. For the neighbor search procedure we use, the so called 2-neighborhoods [21], which are cubic domains consisting at most of 5^d boxes where $5^d - 1$ neighbor boxes surround the central (given) box.

The selection of 2-neighborhoods is motivated by the space dimensions $d = 2$ and $d = 3$, which are in the focus of the present paper. Assume that there are $q + 1$ points located in some box, then for $d = 2, 3$ we can guarantee that for any point in the dataset, its q closest (Euclidean) neighbors are located inside the 2-neighborhood. Indeed, the maximum distance between some point and its qth closest neighbor does not exceed $Dd^{1/2}$, where D is the box size. At the same time the minimum distance from this point to the boundary of the 2-neighborhood is $2D$. As $d^{1/2} \leqslant 2$ (which also means that the 2-neighborhood can be used for $d = 4$) we have such a guarantee and can use the search algorithm described below.

A complication is that the general algorithm above needs an efficient deletion procedure, so the search required to form the ith set $\mathbb{L}_i$ is performed only on the set $\mathbb{X} \backslash \mathbb{C}_{i-1}$. We resolve this issue by introducing linked lists [10] for all points and boxes in the neighborhood of each box. The point linking list, $L_p(2, N)$, is an array, where $L_p(1, i)$ is the index of point from the ordered set $\mathbb{X}'$ preceding point i and $L_p(2, i)$

is the index of point next to i. This list is initialized by setting $L_p(1,i) = i-1$ and $L_p(1,i) = i+1$ and is updated each time as some point i should be deleted. Note that all points $\{i\}$ residing in box j can be characterized by the indices of the first and the last point in this box (bookmarks), as the point linked list is available.

The data structure for handling the neighborhoods is the following. For each box, i, we have a list of boxes in the 2-neighborhood of this box, $N_{neigh}(:,i)$, where the first index takes values from 1 to 5^d. The boxes in this neighborhood are linked in a similar way as the points in the original set. This is provided by the linked lists $L_n(:,i)$, where $L_n(1,i)$ is the relative index of the first element, $L_n(2,i)$ is the link from the first element to the second, and so on. This array is initialized as $L_n(m,i) = m$, $m = 1,...,N_n(i)$, $i = 1,...N_{b,tot}$, where $N_n(i)$ is the number of boxes in the neighborhood of box i and $N_{b,tot}$ is the total number of boxes. For conveniences we also maintain the arrays $N_p(i)$, which shows the number of points in box i, $B(j)$, which is the index of the box containing point j, and $N_{pn}(i)$, which is the total number of points in the neighborhood of box i.

With these arrays the deletion of point j (index in $\mathbb{X}'$) and the q-closest neighbor search procedures for point j (index in $\mathbb{X}'$) can be described by the following algorithms.

Deletion procedure.

- Set $i = B(j)$; $n_p = N_p(i) - 1$; exit if $n_p < 0$ (nothing to delete), else:
- Update the linked list L_p: $a = L_p(1, L_p(j)), b = L_p(2, L_p(j))$, $L_p(1, L_p(j)) = b, L_p(2, L_p(j)) = a$; (note that this should be modified for the first and the last points; also we need to update the bookmarks if the deleted point is the first or the last point in the box); $N_p(i) = n_p$.
- Set $l = l_{\max}$ and perform the following loop while $l \geqslant 1$ and $n_p = 0$ (this procedure excludes all boxes, which become empty due to point deletion):
  1. For $m = 1 : N_n(i)$ (go through all boxes which are in the neighborhood of a box i)
     a. $p = N_{neigh}(L_n(m,i), i)$ $(p \neq i)$; $N_{pn}(p) = N_{pn}(p) - 1$. Update the linked lists $L_n(:,p)$.
  2. $l = l - 1$; $i = Parent(i)$; $n_p = N_p(i) - 1$; $N_p(i) = n_p$ (also check/update bookmarks for box i if $n_p > 0$).

For the sake of clarity, we drop details of our procedure for updating $L_n(:,p)$, which are rather obvious, as one should reconnect the broken links caused due to the exclusion of box i. Note then that the major complexity of this algorithm is due to the loop over all boxes, for which a given box i lie in its neighborhood. This requires at most $\left(5^d - 1\right)^2$ simple operations (repairing links). Also it may appear that this operation is needed for all parent boxes. If $l_{\max} \sim \log N$ the total cost will be $O(\log N)$, with a remark, that the updates of the neighborhood data structure is needed only when the deletion of a point results in an empty box, which then should be deleted. This situation appears relatively rarely. Otherwise, the deletion of point is computationally cheap as it requires just repairing of a broken link (2 operations) and a check/update of bookmarks (2 operations).

q-Closest neighbor search.

- Set $i = B(j)$, $l = l_{\max}$, $n_p = N_p(i)$.
- While $k < q+1$ and $l > 1$ perform the following loop (determine the level at which the box containing point j has at least $q+1$ entries):
  1. $l = l - 1$; $i = Parent(i)$; $k = N_p(i)$.
- Determine $D_{\max}$ and $D_{\min}$ which are the maximum and the minimum dis-

tances from j to the boundary of box i.

- Determine all distances from point j to points in box i, which are different from j. Put them into the distance array *dist* and sort in the ascending order.
- If $dist(q) > D_{\min}$ perform the following procedure for boxes $p \neq i$ from the 2-neighborhood of i :
  1. Determine $D'_{\min}$, which is the minimum distance from j to the boundary of box p.
  2. If $D'_{\min} < D_{\max}$ determine all distances from point j to points in box p. Union them with array *dist*. Sort *dist* in the ascending order.

An informal description of the algorithm is the following. First, we are trying to find a box big enough to contain q neighbors of point j. For this purpose we consider the parent of the box, if the box does not contain sufficient number of points. This procedure is finite (at most $\sim l_{\max} \lesssim \log N$). After such a box found we introduce two threshold distances $D_{\max}$ and $D_{\min}$ as defined above. It may happen that the closest q points to j occupying box i are closer than $D_{\min}$. In this case there is no need to check points from the neighborhood, since they are further than that q points, which provide the solution to the problem. On the other hand, we know that there are at least q neighbors located at distance smaller than $D_{\max}$. Hence there is no need to check the boxes from the neighborhood with $D'_{\min} \geqslant D_{\max}$. It is remarkable that this constraint substantially reduces the number of boxes to check. For example, for $d = 2$ we should check at most 11 neighbor boxes, instead of 24, and for $d = 3$ we have 55 instead of 124 ($5^d - 1$). The numbers in practice are even smaller, as not all points are located in the worst position and the statistical means can be substantially lower.

The above algorithm does not have $O(\log N)$ guaranteed complexity for a single point search and an arbitrary distribution. For very non-uniform distributions an additional research is needed as the cost of search can substantially vary from point to point and it is not clear theoretically what is the overall complexity of the above adaptive scheme.[2] However, guarantees can be provided for uniform and some other typical distributions. Indeed, in the case of a uniform distribution we can assume that all boxes in the neighborhood has approximately the same number of points $\sim q$. Therefore the search is performed on the set of size $O(5^d q)$ which does not depend on N, and so the $\log N$ complexity arises only from the first step of the procedure (in practice for larger q this step is cheaper than the computation of distances due to its small asymptotic constant).

3.2. Fast multipole acceleration of the matrix vector product.

3.2.1. Polyharmonic kernels in 3D. An approach for efficient computation of sums with biharmonic kernel in $\mathbb{R}^3$, $\phi(r) = r$, using the FMM is described in details in [18], where we also show how this method can be extended to computation of polyharmonic kernels in $\mathbb{R}^3$, $\phi(r) = r^{2n+1}$. The major result there can be formulated as follows. The FMM for kernels $\phi(r) = r^{2n+1}$, $n = 0, 1, ...$ can be efficiently reduced to application of $n + 2$ FMMs for the Laplace equation in $\mathbb{R}^3$ (with kernel $\phi(r) = r^{-1}$) for which the FMM is well developed and studied. In practice the cost for the polyharmonic kernel is even smaller, as the same data structure is needed, direct summation in the neighborhood is only needed once, and only a slight modification of the translation operators is needed.

The efficiency of the FMM is determined by the length of function representation,

[2] This remark applies equally to the FMM, where for carefully contrived distributions the $O(N \log N)$ complexity may not be achieved.

which for polyharmonic kernel $\phi(r) = r^{2n+1}$ can be referred as $(n+2)\,p^2$, where p is the truncation number for the equivalent Laplace equation determined by the prescribed accuracy of computations, and the cost of a single translation, which for the fastest practical methods available is $O(p^3)$ (with different asymptotic constants [9, 19]) ($O(p^2 \log p)$ methods with larger asymptotic constant are also available, e.g. [16, 12]). In tests used in the present paper we employed our version for summations with the biharmonic kernel [18], where we used rotational-coaxial translation decomposition of the translation operators, which result in $O(p^3)$ complexity for a single translation.

3.2.2. A novel FMM for the multiquadric kernel in $\mathbb{R}^2$. A fast multipole method for the multiquadric kernel, $\phi(r) = \left(r^2 + c^2\right)^{1/2}$, in $\mathbb{R}^n$ was presented in [11]. While this algorithm can be used for the multiquadric function in $\mathbb{R}^2$, we present a new FMM based approach to evaluate sums with the multiquadric kernel.

First, we note that representation of general functions via expansion coefficients or samples in two dimensions requires double sums or $O(p^2)$ function representation length for a truncation number p (e.g. via truncated Taylor series expansions). In contrast functions that are constrained to satisfy some partial differential equation, such as the harmonic function, can be expressed with just $O(p)$ coefficients in $\mathbb{R}^2$. General translation operators for functions with a $O(p^2)$ representation can then be represented via a $p^2 \times p^2$ truncated matrix, which results in a computational cost of $O(p^4)$ per translation. For large p, methods that employ the convolutional characteristics of the translation can exploit the Fast Fourier Transform to reduce the asymptotic complexity $O(p^2 \log p)$. However, these methods will in practice not be useful for typical problems, and further suffer an inefficiency as they require zero padding. (See [16] for a discussion of this issue). The proposed method also uses $O(p^2)$ representation – however, the translation cost can be reduced to $O(p^3)$, which is faster than the direct translation for $p \geq 3$.

The essence of the method is to extend the space dimensionality from 2 to 3 and separate the source and receiver points, even though in $\mathbb{R}^2$ these points can be the same. Indeed, let $\mathbf{x}_i$ and $\mathbf{y}_j$ be the source and receiver in two dimensions. Their interaction is described then as

$$\phi\left(|\mathbf{y}_j - \mathbf{x}_i|\right) = \left(|\mathbf{y}_j - \mathbf{x}_i|^2 + c^2\right)^{1/2} = \left|\mathbf{y}'_j - \mathbf{x}'_i\right|; \quad \mathbf{y}'_j = \mathbf{y}_j + c\mathbf{i}_z,\ \mathbf{x}'_i = \mathbf{x}_i + 0\mathbf{i}_z,$$

where $\mathbf{y}'_j$ and $\mathbf{x}'_i$ are points in $\mathbb{R}^3$ and $\mathbf{i}_z$ is the unit vector orthogonal to the original plane on which $\mathbf{x}_i$ and $\mathbf{y}_j$ are located. It seen then that this transform leaves the sources on the original plane, while it shifts the locations of all evaluation points by a constant displacement c, so $\mathbf{y}'_j$ is located on the plane parallel to the original one (the case $c = 0$ is a particular, but not special, case). Therefore the problem of computation of sums with the multiquadric kernel in $\mathbb{R}^2$ is equivalent to the computation of sums with the biharmonic kernel in $\mathbb{R}^3$ with the sources and receivers having special locations (placed on parallel planes).

REMARK 3.4. *The increase of dimensionality should not frighten us and, in fact, this does not increase either the memory, or the computational cost. Indeed, in the FMM, that we use all "empty" boxes are skipped.*

REMARK 3.5. *As the sets of sources and receivers are separated by distance not less than c there is no sense in dividing the space up to levels where the size of the boxes are less than c/2 (if the division goes deeper there will be no sources in the neighborhood of the receivers and the computational cost will just increase due to additional translation operations).*

REMARK 3.6. *This method also can be successfully applied for computations of sums with the inverse multiquadric kernel* $\phi(r) = \left(r^2 + c^2\right)^{-1/2}$ *in* $\mathbb{R}^2$. *Sums involving this kernel can be reduced to a summation with the Laplacian, kernel* $1/r$ *in* $\mathbb{R}^3$.

4. Numerical tests. To check the accuracy and performance of the fast algorithms and compare them with [13] we implemented the described algorithms and the FGP05 algorithm in Fortran 90 in double precision arithmetic and performed some tests on an Intel Xeon 3.2 GHz processor with 3.5 GB RAM. Performance tests were conducted using two basic random point distributions for $\mathbb{R}^2$ (with the multiquadric kernel) and $\mathbb{R}^3$ (with the biharmonic kernel). The parameter c for the multiquadric kernel was varied over some range. The first distribution coincides with test case A of [13]. In this case points were uniformly distributed inside a d dimensional unit sphere, we refer this case also as "A". The second distribution is substantially non-uniform, as the points in this case were uniformly distributed over the surface of a d dimensional unit sphere and so were on manifolds of lower than d dimensionality. Such distributions are typical for interpolation of curves and surfaces. We refer this case as "E". In both cases the entries of the right hand side vector were generated as random numbers uniformly distributed on $[-1, 1]$. We also performed several illustrative computations for some standard problems appearing in 2D and 3D computer graphics.

4.1. Preconditioner tests. As there are new elements in the present algorithm which are related to the construction of the preconditioner (or approximate cardinal functions) and to the approximate FMM based matrix vector multiplication, we, first studied the algorithm where the matrix vector product was performed in the straightforward way.

The first test was to ensure that our implementation of the original FGP05 algorithm is correct. We found that we were able to produce results consistent with those reported in [13]. We next modified the algorithm to use the L-sets produced by our algorithm, which implement exactly what is implemented approximately in [13]. Table 1 shows the number of iterations required to converge to an accuracy $\epsilon = 10^{-10}$ for the test case A at different N and q. Despite the fact that for some selected values of these parameters we did several tests trying to find averages, the variation from test to test was rather small (usually within 1 iteration), so we put in the table values for a single random realization. The stability of the number of iterations for different realizations was also pointed out in [13].

The results for case $d = 2, c = 0$ are very close to that reported in [13] (we checked also $q = 20$ and $q = 40$, but did not put it in the table to reduce its size). As one can see from the table the number of iterations for preconditioners produced by the both algorithms are almost the same with possible variations of a few percent in either direction.

The time required to build the L-sets and compute approximate cardinal function approximation is almost insensitive to parameter c of the multiquadric kernel, while it depends on the point distribution. Table 2 shows the CPU time (in seconds) required to build preconditioners using the original and the present algorithm for four cases (A and E, $d = 2$ and $d = 3$) at $q = 30$ and different N. These results are also presented in Figure 4.1. As before, we report the timing for a single random realization, while several entries into the table were recomputed several times to insure that the variations in timing are rather small (within a couple of percent).

It is clearly seen that for $N < N_b$, which is of order of thousands, the origi-

Problem	N	$q = 10$		$q = 30$		$q = 50$	
		Present	FGP05	Present	FGP05	Present	FGP05
$d = 2, c = 0$	200	15	16	8	8	7	7
$d = 2, c = 0$	500	18	19	9	9	8	7
$d = 2, c = 0$	1000	18	20	10	9	8	8
$d = 2, c = 0$	2000	21	22	10	10	9	9
$d = 2, c = 0$	5000	23	26	11	11	10	10
$d = 2, c = 0$	10000	25	27	13	12	11	10
$d = 2, c = N^{-1/2}$	200	20	20	8	8	7	7
$d = 2, c = N^{-1/2}$	500	23	24	11	10	9	8
$d = 2, c = N^{-1/2}$	1000	25	26	11	11	9	9
$d = 2, c = N^{-1/2}$	2000	27	28	11	12	9	10
$d = 2, c = N^{-1/2}$	5000	31	33	12	12	11	10
$d = 2, c = N^{-1/2}$	10000	35	36	13	13	11	11
$d = 3, c = 0$	200	22	23	11	12	8	9
$d = 3, c = 0$	500	29	30	14	14	10	11
$d = 3, c = 0$	1000	36	39	17	16	12	11
$d = 3, c = 0$	2000	41	44	19	19	13	13
$d = 3, c = 0$	5000	57	58	23	23	15	15
$d = 3, c = 0$	10000	68	74	26	26	17	17

TABLE 4.1

Number of iterations required for test problem A of [13], implemented using the cardinal point set algorithm of [13] and our algorithm.

$N \rightarrow$	300	1000	3000	10000	30000	100000	300000	1000000
(A) $d = 2$:								
Present	0.2	0.7	2.1	7.2	22	81	256	854
FGP05	0.1	0.5	2.2	17	157	2445	(19900)	(221100)
(A) $d = 3$:								
Present	0.2	0.8	3.0	12	43	184	623	2130
FGP05	0.1	0.5	2.1	15	139	2211	(19900)	(221100)
(E) $d = 2$:								
Present	0.2	0.6	1.8	6	18	62	188	623
FGP05	0.1	0.5	2.4	18	177	2816	(19900)	(221100)
(E) $d = 3$:								
Present	0.2	0.8	2.7	9.2	32	110	330	1125
FGP05	0.1	0.6	2.4	17	155	2451	(19900)	(221100)

TABLE 4.2

The time required for initial computations in the FGP05 iterative algorithm

nal algorithm performs better, as an additional overhead due to more complex data structure is needed for the present algorithm. However for $N > N_b$ the execution time for the original algorithm is scaled as $O(N^2)$, while for the present algorithm it is scaled almost linearly with N. The scaling constant and N at which the present algorithm starts to perform linearly depends on the problem dimensionality and point distribution. The problem dependence of the present algorithm is stronger than for the original algorithm. We note also that due to large time required by the original algorithm for large N we did not compute the values for $N > 100000$ and put estimated times in parenthesis based on the fit line shown in the figure. It is seen that for large N the ratio of times for the algorithms tested can be orders of magnitude.

Table 1 shows that the number of iterations decreases as the size q of the L-sets

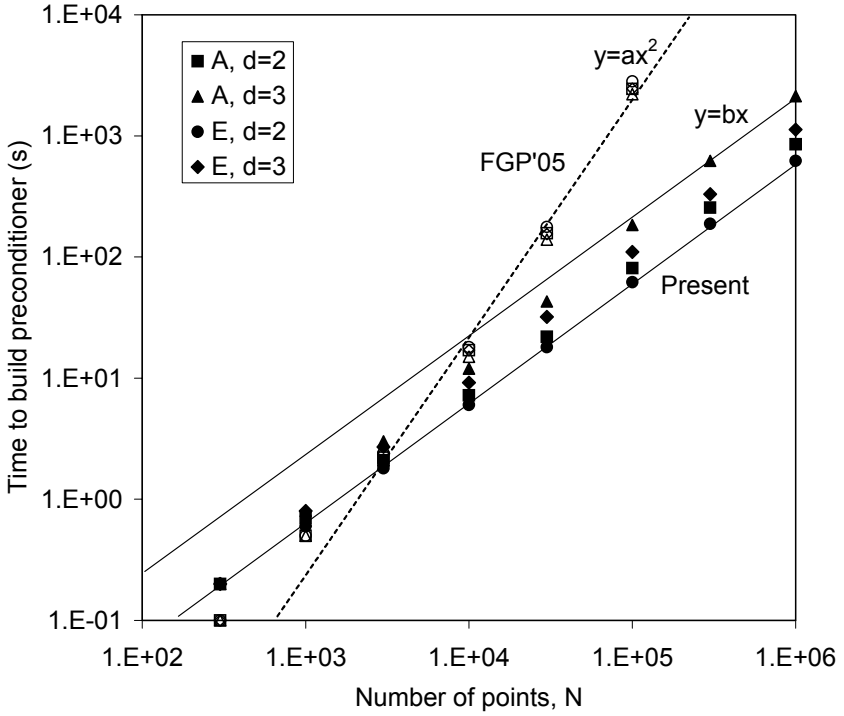

FIG. 4.1. *Dependences of the CPU time for construction of the preconditioner ($q = 30$) for algorithm FGP05 (shown by the light markers) and the present algorithm (the dark markers) for 4 different problems. The solid lines show the linear function and the dashed line shows the quadratic function in the log-log coordinates.*

increases. So it may preferable to use larger q to reduce the iteration time. On the other hand, the time required to build the L-sets and compute cardinal function approximations increases at increasing q, which is the limiting factor to use larger q. This time depends on N and particular distribution. Theoretically, the time to build the L-sets for a fixed distribution and N should grow almost linearly with q (asymptotically as $q \log q$; the logarithmic complexity appears due to sorting procedure used for determination of closest neighbors). The time to solve a system of q equations to build the approximate cardinal function grows $\sim q^3$ as we used a standard LAPACK symmetric solver routine for this purpose. Figure 4.2 presents actual computation times for problems A and E and $d = 2$ and 3 at $N = 100000$. It is seen that the time depends on the problem dimensionality and the largest times obtained for $d = 3$. We note also that an effective dimensionality of problem E for large N is rather $d-1$, than d, which explains the smaller times required for this problem compared to problem A.

4.2. Tests of the algorithm with the FMM. Speeding up of the matrix vector product using approximate methods, such as the FMM, raises several issues on the accuracy, convergence, and the speed of the algorithm. The accuracy of the FMM is controlled by the truncation number p. For multiquadric functions the error of matrix vector multiplication in the L^∞ norm, ϵ_N, decays exponentially at growing p, $\epsilon_N \sim N\eta^{-p}$, assuming that they contribute to the error in the worst way (no error cancellation). Actual errors usually are orders of magnitude lower than theoretical

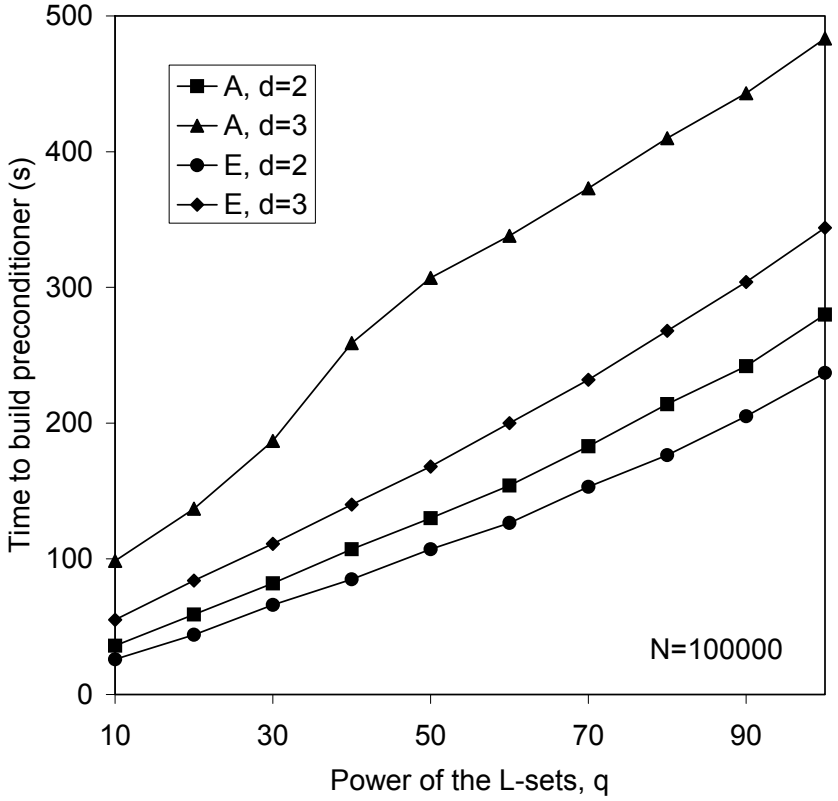

FIG. 4.2. *Dependences of the time required to construct preconditioner for problems A and E on the size of the L-sets q. The number of points for all cases is the same,* $N = 100000$.

bounds and depend on the problem dimensionality and particular point distributions. Concerning convergence of the algorithm we found for given prescribed accuracy empirically numbers $p_*(N)$, $p > p_*(N)$. Here p_* depends on the problem the number of iterations to converge to any prescribed accuracy is the same as in the case when direct matrix vector multiplication is used. Moreover the norm of the residuals in this process do not depend on p. Despite this, because of conditioning of the matrix, the solution may converge to a solution that is not exact. The error of this solution can be evaluated for smaller N as we can check it using one straightforward matrix vector multiplication to evaluate the residual and the results can be extrapolated to larger N as we know qualitatively the dependence of the matrix vector product on N and p. Of course, the error criterion used to stop the iterative process, should be selected in some consistent way with this error. This whole issue is somewhat complex since in many application the data are noisy [8], and the fitting equations may themselves be regularized.

Note then the tuning (optimization) of the overall algorithm is not a simple procedure. Indeed, for some prescribed error, the truncation number should be a function of N (e.g. $p \sim \log CN$). Based on p the optimal level of space subdivision, $l_{\max}$, for the FMM should be selected. The complexity of the FMM for optimal level depends on the complexity of translation procedures. For example, for the rotational-coaxial

$N \rightarrow$	1000	3000	10000	30000	100000	300000	1000000
(A) $d = 2, c = 0$:							
Present	1.1	3.0	13	43	163	618	2512
FGP05	0.7	4.1	42	424	5477	(49320)	(547700)
(A) $d = 2, c = N^{-1/2}$:							
Present	1.1	3.7	15	50	217	740	3319
FGP05	0.8	4.6	47	430	6463	(49320)	(547700)
(A) $d = 3, c = 0$:							
Present	1.9	5.5	36	126	736	3460	18660
FGP05	0.7	5.5	61	634	5248	(49320)	(547700)
(E) $d = 3, c = 0$:							
Present	1.2	3.6	15	69	324	1152	5332
FGP05	0.8	4.1	38	384	5486	(49320)	(547700)

TABLE 4.3
Total CPU time required by the present and FGP05 algorithms.

translation decomposition in some range of p for three dimensional problems this can be $O(p^{3/2}N) = O(N \log^{3/2} N)$. Furthermore one should select an appropriate q for the preconditioner. As the time to build the preconditioner depends on q and the number of iterations decays with q one can expect minima for the cost. We found that there can be several minima (e.g. we found four for $N = 100000$, $d = 2$, $c = 0$, problem A in range $15 \leqslant q \leqslant 30$). Dependence of the number of iterations on N is also important. As an accurate addressing of these issues goes outside the scope of this paper, plus fundamental studies of the FMM induced matrix perturbations on the accuracy of the overall solution are required, we will further limit ourselves by presentation of several results of the algorithm performance for some test cases. Further, Faul et al. [13] indicate that their intuition is that the choice of q should be independent of problem size.

Table 3 shows the CPU times required for computation of some test problems of different size. Here the CPU time includes both the time to compute the preconditioner and run the iterative process. In all cases the prescribed error was $\epsilon = 10^{-3}$, and p was increasing for increasing N. For example, for test case A, $d = 2$, $c = N^{-1/2}$ we changed p from 16 to 28 as N changed from 1000 to 1000000. The parameter q in all cases was 30 (the number suggested by [13]), which provided an acceptably small number of iterations, except of case A, $d = 3$, $c = 0$, where substantial growth in the number of iterations was observed for $N \geqslant 100000$. For these cases we set $q = 100$, which provided a substantially smaller number of iterations, and overall savings in the computational time.

Results presented in the table are also plotted in Figure 4.3. The table shows that for all examples the cross-over point between the FGP05 and present algorithm do not exceed N =3000. One can see also that at larger N the CPU time for the FGP05 in the computed range can be fitted by $O(N^2)$ dependence (in fact, it should larger than this estimate as the number of iterations slightly grow with N). We used this fit to estimate the CPU time for the FGP05 algorithm for $N > 100000$, which a bit underestimates the time and put results in Table 3 in parenthesis. For the present algorithms the data can be fitted by dependences $O(N^\alpha)$ (one can also use fits of type $O(N \log^\beta N)$, which should hold if the number of iterations growth with N as some power of $\log N$). The constant α for our computations varies from $\alpha = 1.15$ to $\alpha = 1.4$, and as we mentioned there two basic reasons for this: grows of the truncation number p and the number of iterations. Also some computational overheads for larger N contribute to the total CPU time reported. As we also mentioned above optimization

problems can be solved more accurately than some heuristic which we used for the illustration cases. In any case, one can see substantial speed ups achieved using the present method, which make practical computations for million point size sets (larger problems can be computed using parallel computations and larger memory).

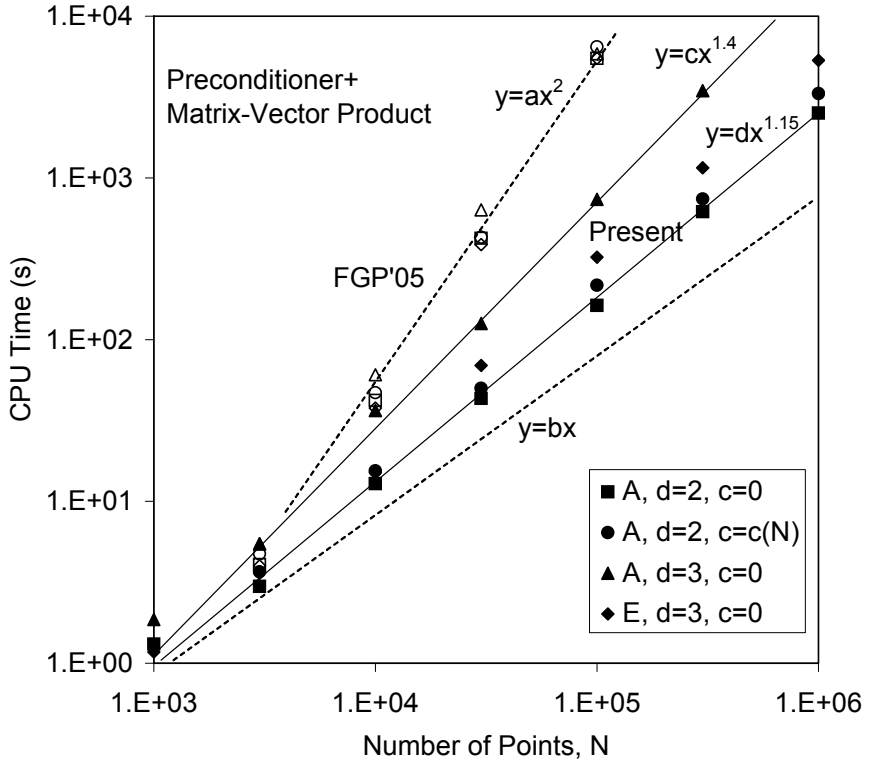

FIG. 4.3. *The total CPU time required for solution of different problems. The empty markers show computations using algorithm FGP05, while the filled markers show computations using the present algorithm. The straight lines with different angular coefficients are ploted for conveninence of data analysis.*

Figure 4.4 provides illustrations of solution of some problems from 3D and 2D computer graphics using multiquadric RBFs and the present algorithm. In the first example bunny (34834 point set) was extended to size 104502 (for points along the normal inside and outside) and after interpolation coefficients were determined the interpolant was evaluated at ~ 8000000 points (regular spatial grid $201 \times 201 \times 201$) from which isosurface was found using standard routines. Here we used the same settings as for problem A with $d = 3$ and $c = 0$. The computation of preconditioner took 310 s, and the iteration process 283 s. Evaluation was made using the FMM for (which required one matrix vector multiplication) and took 395 s.

In the second example 86% of 375050 pixels in a 2D color image were removed randomly. The remaining (N =51204) pixels were used to interpolate data back to the grid using the 2D multiqudaric kernel with $c = N^{-1/2}$. The results of recovery are presented in the figure. Here we used the same setting as for problem A with $d = 2$ and $c = N^{-1/2}$. The preconditioner should be build only once as it is suitable

for all three RGB channels and it took about 30 s to perform this task. The iteration process took 75 s per channel and the final matrix-vector multiplication took 19 s. Note that three digit accuracy, that we imposed for solution is sufficient to determine the RGB intensities as they vary between 0 to 255 for each channel.

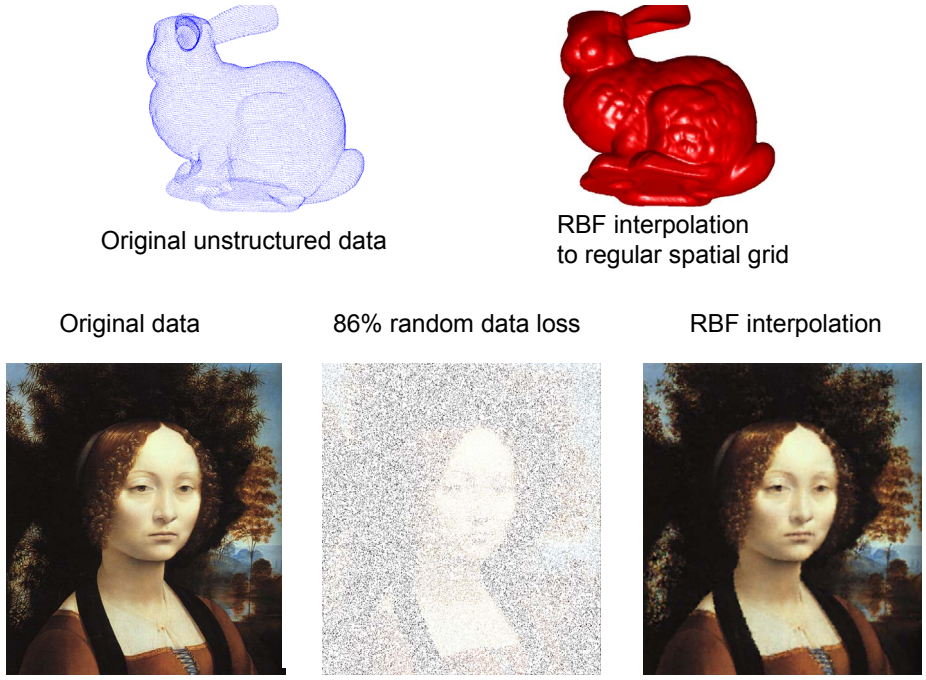

FIG. 4.4. *Illustrations of use of the RBF for 3D and 2D computer graphics problems.*

5. Discussion and conclusion. For many dense linear systems of size the availability of an algorithm based on the fast multipole method (FMM) provides a matrix vector product with time and memory complexity that is $O(N \log N)$. This reduces the cost of the matrix vector product step in the iterative solution of linear systems, and for an iteration requiring k steps, a complexity of $O(kN \log N)$ of can be expected. To bound k, appropriate preconditioning strategies must be used. However, many conventional preconditioning strategies rely on sparsity in the matrix, and applying them to these dense matrices requires computations that may have a formal time or memory complexity of $O(N^2)$, which negates the advantage of the FMM. With our construction, all steps in the FGP05 algorithm have been accelerated and to have a similar formal complexity, and in practice converge in relatively few iterations. The success of this preconditioner based on the geometry of point sets for a matrix that is capable of being accelerated using the FMM should not be surprising, as the FMM itself is based on the use of computational geometry to partition the matrix vector product.

The idea of constructing preconditioned Krylov algorithms based on the cardinal functions may have wider applicability. For any regular FMM kernel function, as the discussion in Section 2.1 shows, we can construct an approximate inverse to the cardinal function expressed in terms of the kernel function. This idea bears further exploration.

6. Acknowledgements. We would like to thank Prof. David Mount for providing us some pointers to material on computational geometry.

REFERENCES

[1] R.K. Beatson and M.J.D. Powell, "An iterative method for thin plate spline interpolation that employs approximations to Lagrange functions,". Numerical Analysis 1993. (D. F. Griffiths & G. A. Watson, eds). London: Longmans, pp. 17–39, 1994.

[2] R. K. Beatson and L. Greengard. "A short course on fast multipole methods." In M. Ainsworth, J. Levesley, W.A. Light, and M. Marletta, editors, *Wavelets, Multilevel Methods and Elliptic PDEs,* pages 1–37. Oxford University Press, 1997.

[3] R. K. Beatson, J. B. Cherrie, and C. T. Mouat, "Fast fitting of radial basis functions: methods based on preconditioned GMRES iteration," *Advances in Comput. Math.*, 11, 1999, 253–270.

[4] R. K. Beatson, W. A. Light, S. Billings "Fast solution of the radial basis function interpolation equations: domain decomposition methods," SIAM Journal on Scientific Computing, Vol. 22, pp. 1717-1740, 2000

[5] R.K. Beatson, J.B. Cherrie, and D.L. Ragozin, "Polyharmonic splines in $\mathbb{R}^d$: tools for fast evaluation, *In: A. Cohen, C. Rabut, and L. Schumaker (eds.) Curve and Surface Fitting: Saint-Malo 1999*, Vanderbuilt University Press, Nashville, TN, 2000.

[6] R. K. Beatson, J. B. Cherrie, and D. L. Ragozin, "Fast evaluation of radial basis functions: Methods for four-dimensional polyharmonic splines", SIAM J. Math. Anal., vol. 32, no. 6, pp. 1272-1310, 2001.

[7] J. Bloomenthal, C. Bajaj, J. Blinn, M.P.Cani-Gascuel, A. Rockwood, B. Wyvill, and G. Wyvill, *Introduction to Implicit Surfaces*, Morgan Kaufmann Publishers, San Francisco, CA, 1997.

[8] J.C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, B.C. McCallum, and T.R. Evans, "Reconstruction and representation of 3D objects with radial basis functions," *ACM SIGGRAPH 2001*, Los Angeles, CA, 2001, 67-76.

[9] H. Cheng, L. Greengard and V. Rokhlin, "A fast adaptive multipole algorithm in three dimensions," J. Comput. Phys. vol. 155, pp. 468–498, 1999.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein, *Introduction to Algorithms,* MIT Press, 2004.

[11] J. B. Cherrie, R. K. Beatson, and G. N. Newsam, "Fast evaluation of radial basis functions: Methods for generalized multiquadrics in R^n", SIAM J. Sci. Comput. , vol. 23, no. 5, pp. 1549-1571, 2002.

[12] W. D. Elliott and J. A. Board. Fast Fourier transform accelerated fast multipole algorithm. SIAM Journal on Scientific Computing, 17(2):398–415, 1996.

[13] A. C. Faul, G. Goodsell, M. J. D. Powell, "A Krylov subspace algorithm for multiquadric interpolation in many dimensions," IMA Journal of Numerical Analysis (2005) 25, 1–24

[14] L. Greengard and V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys., 73, 1987, 325-348.

[15] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems*, MIT Press, Cambridge, MA, 1988.

[16] L. Greengard and V. Rokhlin, On the efficient implementation of the fast multipole algorithm, Tech. Rep. RR-602, Department of Computer Science, Yale University, 1988.

[17] G. Goodsell, "On finding p-th nearest neighbours of scattered points in two dimensions for small p," Computer Aided Geometric Design, vol. 17, pp. 387-392, 2000.

[18] Nail A. Gumerov and Ramani Duraiswami, "Fast multipole method for the biharmonic equation in three dimensions," Journal of Computational Physics, Volume 215, Issue 1, 10 June 2006, Pages 363-383

[19] N.A. Gumerov and R. Duraiswami, "Comparison of the efficiency of translation operators used in the fast multipole method for the 3D Laplace equation," University of Maryland Department of Computer Science Technical Report, CS-TR-4701; UMIACS Technical Report, UMIACS-TR–2005-09, 2005.

[20] N.A. Gumerov and R. Duraiswami, *Fast Multipole Methods for the Helmholtz Equation in Three Dimensions*, Elsevier, Oxford, UK, 2004.

[21] N.A. Gumerov, Ramani Duraiswami, and Eugene A. Borovikov, "Data structures, optimal choice of parameters, and complexity results for generalized multilevel fast multipole methods in d dimensions," University of Maryland Department of Computer Science Technical Report, CS-TR-4458, 2003.

[22] R. L. Hardy, "Theory and applications of the multiquadric-biharmonic method," Comput.

Math. Appl., 19, 163–208, 1990.
[23] C. A. Micchelli, "Interpolation of scattered data: distance matrices and conditionally positive definite functions." Constr. Approx., 2, 11–22, 1986.
[24] S. Maneewongvatana and D. M. Mount, "Analysis of Approximate Nearest Neighbor Searching with Clustered Point Sets, " in *Data Structures, Near Neighbor Searches, and Methodology:* Fifth and Sixth DIMACS Implementation Challenges, eds. M. H. Goldwasser, D. S. Johnson, C. C. McGeoch, in the DIMACS Series in Discr. Math. and Theoret. Comp. Sci., Vol. 59, AMS, 105-123, 2002.
[25] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison–Wesley, New York, 1989.
[26] G. Turk and J.F. O'Brien, Modelling with implicit surfaces that interpolate, *ACM Trans. on Graphics*, 21(4), 2002, 855-873.
[27] Eric W. Weisstein. "Kissing Number." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/KissingNumber.html