Efficient mesh deformation based on radial basis function interpolation by means of the inverse fast multipole method

Pieter Coulier^{a,b,*}, Eric Darve^b

^a*KU Leuven, Department of Civil Engineering, Kasteelpark Arenberg 40, 3001 Leuven, Belgium*

^b*Stanford University, Department of Mechanical Engineering, 496 Lomita Mall, 94305 Stanford, CA, USA*

Abstract

Radial basis function interpolation is often employed in mesh deformation algorithms for unstructured meshes, for example in fluid-structure interaction or design optimization problems. This is known to be a robust methodology that results in high quality deformed meshes. The applicability of this method to large problems is currently hampered by its prohibitive computational cost, however, which is caused by the need to solve a dense system of equations. The computation time grows as $\mathcal{O}(N_b^3)$ if a direct solver is employed for the solution (where N_b denotes the number of boundary nodes in the mesh), while alternative iterative solvers often suffer from an unfavorable convergence behavior. In this paper, we present the inverse fast multipole method as a novel fast approximate direct solver with a computational cost scaling as $\mathcal{O}(N_b)$. The linear complexity is achieved by transforming the dense system into an extended sparse system, along with the compression of certain matrix blocks into low-rank factorizations. The solver is inexact, although the error can be controlled and made as small as needed; a low accuracy solver can hence be used as an efficient preconditioner in an iterative scheme. Numerical benchmarks are presented, demonstrating that the proposed approach enhances the computational efficiency of mesh deformation algorithms based on radial basis function interpolation significantly, without jeopardizing their robustness and quality.

Keywords: Mesh deformation, radial basis functions, fluid-structure interaction, $\mathcal{H}^2$ -matrices, fast direct solvers, preconditioned Krylov solvers.

1. Introduction

In many engineering applications, it is often necessary to update an unstructured mesh as the numerical simulation proceeds. Typical examples are fluid-structure interaction computations that involve moving boundaries for the fluid upon the deformation of the structure, or design optimization problems [1]. The regeneration of a completely new mesh in each simulation step is often very expensive, however, especially for large three-dimensional problems. It also leads to inconsistencies in numerical accuracy throughout the

*Corresponding author.

Email address: pieter.coulier@bwk.kuleuven.be (Pieter Coulier)

simulation and should be avoided if possible. It is therefore often more desirable to apply an algorithm that deforms the initial mesh in such a way that the imposed deformations are accommodated, while maintaining a reasonable quality throughout the mesh. Preserving the mesh quality is a crucial requirement to ensure that the physical processes of interest are modeled accurately, along with the need for computational efficiency.

The literature on mesh deformation algorithms is vast and the reader is referred to [2, 3, 4] for a detailed overview of this field. Generally speaking, three categories of mesh deformation algorithms can be distinguished. The first category consists of methods that exploit information regarding the connectivity of the mesh and are mainly based on some form of physical analogy. These methods are usually applicable to both structured and unstructured meshes. A common example is the spring analogy in which the mesh is represented as a network of springs with a stiffness inversely proportional to the edge lengths [5]. Many variants of this approach have been developed, such as the torsional [6], semi-torsional [7], ortho-semi-torsional [8], and ball-vertex [9] spring analogies. Other methods are based on solid body elasticity analogies [10] or on elliptic smoothing techniques [11, 12], using Laplacian or biharmonic operators. These methods usually require the solution of a sparse system of equations that involves all the internal grid points, which can be very large and hence quite expensive (especially when encountering convergence difficulties with iterative solvers). Robustness and the capability to handle large mesh deformations are other important aspects that need be considered when choosing an appropriate mesh deformation algorithm. Although the simplest versions of the aforementioned analogies might lack these properties [3], the more recently developed techniques are more robust and able to deal with large deformations [8, 13].

Point-by-point schemes form a second category of methods; they purely rely on the interpolation of the imposed boundary displacements and do not require any mesh connectivity information. Examples of such techniques are the Delaunay graph mapping (DGM) method [14], the radial basis function (RBF) interpolation method [15, 16], and the inverse distance weighting (IDW) interpolation approach [17, 18].

The third category encompasses hybrid schemes, which aim at combining the strengths of different mesh deformation methods. For example, Lefrançois [19] presents a scheme in which a deformation strategy based on an elastic analogy is applied to a coarse grid, while the resulting deformation is subsequently interpolated on a finer grid. Other hybrid methods have been presented in [20, 21].

From the methods listed above, mesh deformation based on RBF interpolation is generally considered as a robust method that result in high quality deformed unstructured meshes. This technique originated from earlier deformation methods that were restricted to structured grids, e.g., based on volume splines [22, 23] and transfinite interpolation [24]. de Boer et al. [15] generalized the method to unstructured grids and applied a broader class of radial basis functions. RBF-based mesh deformation is nowadays applied in a variety of research fields [25, 26], has been integrated in commercial software packages [27], and is the subject of ongoing research projects [28].

In RBF interpolation, the displacements at any point in the domain are expressed as a weighted sum

of radial basis functions. The method requires the assembly and solution of a (dense) system of equations that only involves the boundary nodes for obtaining the RBF weights, and subsequently proceeds with the evaluation of the displacements at all internal nodes. Its implementation is straightforward but leads to a high computational cost, however, making it rather unsuitable for large scale applications. More precisely, the solution and evaluation steps have costs that scale as $\mathcal{O}(N_b^3)$ (if a direct solver is employed) and $\mathcal{O}(N_b N_d)$, with N_b and N_d denoting the number of nodes on the boundary and inside the domain, respectively. Innovative strategies have therefore recently been developed to reduce these costs. Jakobsson and Amoignon [29] as well as Rendall and Allen [30] present a strategy in which only a limited set of boundary nodes is used to perform the RBF interpolation (i.e., a coarsening approach); these nodes are selected by means of a greedy algorithm. Although this leads to significant speed-up, it also introduces an error at the unused boundary nodes, hence necessitating an additional correction step [31] (e.g., by means of Delaunay graph mapping [32]). Other recent developments in this direction can be found in [33, 34, 35]. Coarsening strategies are usually very efficient in cases where the surface deformation is smooth, such that only a small subset of the surface nodes needs to be considered for an accurate representation of the mesh deformation. For more complicated deformations, however, a large subset (or even the whole set) of surface points is required. This can occur in multi-phase flow simulations [36] or in design optimization problems [1]. The computational cost for solving the (somewhat smaller) dense system of equations remains significant in such cases.

Methods specifically aiming at accelerating the evaluation step in interpolation methods have also been proposed, see for example Livne and Wright [37] and Luke et al. [18]. Gumerov and Duraiswami [38] have presented a specialized preconditioned iterative solver for RBF interpolation with multiquadratic functions that reduces the cost to $\mathcal{O}(N_b \log N_b)$, but this approach has not been applied so far to mesh deformation problems.

The solution of large dense systems $\mathbf{Ax} = \mathbf{b}$ not only arises when performing RBF interpolation, but also in many other applications in science and engineering (e.g., in boundary element methods [39], Kalman filtering [40], ...). As mentioned before, the computational cost of direct solvers (where “direct” refers to the use of a factorization of $\mathbf{A}$) does not scale well with the problem size, while iterative solvers such as Krylov subspace methods often suffer from an unfavorable convergence behavior (leading to a large number of iterations). Significant progress has been made during the past decade regarding the development of fast (approximate) direct solvers that scale almost linearly with the problem size. Such methods have been presented by, among others, Martinsson and Rokhlin [41], Chandrasekaran et al. [42], Xia et al. [43], Ho and Greengard [44], Gillman et al. [45], Corona et al. [46], and Ho and Ying [47]. Nevertheless, the applicability of existing fast solvers is often restricted to 1D and 2D problems; there is still a lack of fast direct solvers for general 3D problems.

In this paper, we present a novel fast direct solver for 3D problems that reduces the computational cost

for (approximately) solving a dense system of equations from $\mathcal{O}(N_b^3)$ to $\mathcal{O}(N_b)$, hence eliminating the main bottleneck associated with mesh deformation based on RBF interpolation. This solver, termed “the inverse fast multipole method”, relies on the fact that the dense RBF-matrix can be represented in a hierarchical format, where some matrix blocks are replaced by low-rank representations [48, 49]. This hierarchical structure is exploited to express the original system as an extended sparse system. Furthermore, the fill-in¹ arising during the elimination² of the latter is avoided through additional matrix compressions. This is one of the first direct solvers that achieves a linear complexity $\mathcal{O}(N_b)$ for a broad category of dense matrices [50, 51]. The use of low-rank approximations makes the solver inexact, but the error can be controlled and made as small as necessary. A low-accuracy solver is hence applicable as preconditioner in an iterative scheme.

It is important to mention that the methodology proposed in this paper does not exclude the application of the aforementioned coarsening strategies for reducing the computational cost of the RBF-based mesh deformation technique; both methods are complementary and can be used together. We also want to emphasize that it is not the aim of this paper to judge whether RBF interpolation is better (or worse) than any other mesh deformation approach, but rather presents a methodology to improve its computational efficiency. It is up to each individual user/researcher to assess which technique is best suited for the specific problem under concern. If the RBF-based technique is chosen, however, the proposed methodology provides a significant speed-up and can be of large practical value.

The outline of this paper is as follows. [Section 2](#) briefly recalls the methodology of mesh deformation based on RBF interpolation. The inverse fast multipole method is subsequently introduced in [Section 3](#), focusing on the main features for achieving a solver with linear complexity. Numerical benchmarks are presented in [Section 4](#) to demonstrate the efficacy of the proposed strategy, while two engineering applications involving the open-source CFD code SU2 [52, 53] are finally discussed in [Section 5](#).

2. Mesh deformation based on radial basis function interpolation

Mesh deformation based on RBF interpolation is a point-by-point scheme in which displacements within a domain are obtained based on displacements imposed on its boundary. In the following, the number of nodes on the boundary and inside the domain are denoted as N_b and N_d , respectively. The displacement $u_i(\mathbf{x}_k)$ in the direction i ($i = x, y, z$) at a node $\mathbf{x}_k = \{x_k, y_k, z_k\}^T$ inside the domain is written as a weighted sum of radial basis functions $\phi(r)$ and a low-order polynomial $p(\mathbf{x})$:

$$u_i(\mathbf{x}_k) = \sum_{j=1}^{N_b} \alpha_j \phi(\|\mathbf{x}_k - \mathbf{x}_j\|) + p(\mathbf{x}_k) \quad \text{for } k = 1 \dots N_d \quad (1)$$

¹The term *fill-in* is used here to denote the non-zero matrix entries.

²The term *elimination* refers to removing variables from a system of equations and updating the coefficients of the remaining variables appropriately.

The polynomial $p(\mathbf{x})$ is usually expressed as:

$$p(\mathbf{x}) = \sum_{l=1}^{n_p} \beta_l q_l(\mathbf{x}) \quad (2)$$

where $q_l(\mathbf{x})$ are n_p polynomials with degree less or equal than the degree of $p(\mathbf{x})$. In order to obtain the weights α_j and the polynomial coefficients β_l , interpolation conditions on the boundary are used:

$$u_i(\mathbf{x}_j) = \bar{u}_i(\mathbf{x}_j) \quad \text{for } j = 1 \dots N_b \quad (3)$$

where $\bar{u}_i(\mathbf{x}_j)$ is the prescribed displacement in the direction i at boundary node $\mathbf{x}_j$. In addition, the following constraints are typically enforced:

$$\sum_{j=1}^{N_b} \alpha_j q_l(\mathbf{x}_j) = 0 \quad \text{for } l = 1 \dots n_p \quad (4)$$

Eq. (4) can be interpreted as an orthogonality condition on the weights α_j such that the discrepancy between the prescribed displacements and the polynomial interpolation is minimized.

Combining Eq. (3) and (4) leads to a linear system of equations:

$$\begin{bmatrix} \Phi_b & \mathbf{P}_b \\ \mathbf{P}_b^T & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{Bmatrix} = \begin{Bmatrix} \bar{\mathbf{u}}_{bi} \\ \mathbf{0} \end{Bmatrix} \quad (5)$$

with $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ containing the weights α_j and polynomial coefficients β_l , respectively. Φ_b is a $(N_b \times N_b)$ -matrix whose entries are $\Phi_b(j, k) = \phi(\|\mathbf{x}_j - \mathbf{x}_k\|)$, while $\mathbf{P}_b$ is a $(N_b \times n_p)$ -matrix. The unknowns $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are thus given by:

$$\boldsymbol{\alpha} = \Phi_b^{-1} (\bar{\mathbf{u}}_{bi} - \mathbf{P}_b \boldsymbol{\beta}) \quad (6)$$

$$\boldsymbol{\beta} = (\mathbf{P}_b^T \Phi_b^{-1} \mathbf{P}_b)^{-1} \mathbf{P}_b^T \Phi_b^{-1} \bar{\mathbf{u}}_{bi} \quad (7)$$

Note that the interpolation procedure has to be performed for each spatial dimension i separately. The mesh deformation strategy is illustrated by means of a small 2D example, consisting of a rectangle of 1.5×0.5 located in the center of a rectangular domain of 15×5 . The domain is discretized with triangles, resulting in the initial mesh shown in Figure 1a. A unit translation in both directions is imposed on the inner rectangular block, while the nodes at the outer boundary of the computational domain are fixed. This leads to the deformed mesh depicted in Figure 1b.

A variety of radial basis functions $\phi(r)$ are available in the literature [54, 55], where a distinction between functions with compact and global support is usually made. The former are exactly zero at some distance

Figure 1: (a) The initial mesh of a rectangle of 1.5×0.5 located in the center of a rectangular domain of 15×5 . A unit translation in both directions is imposed on the green nodes, while the red nodes are fixed. The displacements of the purple nodes are obtained through RBF interpolation, resulting in the deformed mesh shown in (b).

away from the origin (i.e., outside their support radius), while the latter are always non-zero. Functions with global support lead in general to more accurate solutions [15] but also give rise to a dense matrix Φ_b in Eq. (5), resulting in a more challenging problem from a computational point of view (compared to functions with local support, leading to a sparse matrix). The more challenging problem of functions with global support is therefore considered throughout this paper.

Some commonly used radial basis functions with global support are listed in Table 1. For many of these functions, the addition of a linear polynomial in Eq. (1) suffices (i.e., $n_p = 4$) [56], implying that row j of the matrices $\mathbf{P}_b$ and $\mathbf{P}_d$ is given by $[1 \ x_j \ y_j \ z_j]$.

Radial basis function	$\phi(r)$
Gaussian	$e^{-a^2 r^2}$
Thin plate spline	$r^2 \log(r)$
Multiquadratic biharmonic	$\sqrt{a^2 + r^2}$
Inverse multiquadratic biharmonic	$\sqrt{\frac{1}{a^2 + r^2}}$

Table 1: Commonly used radial basis functions with global support $\phi(r)$ (with shape parameter a).

It is clear from the discussion above that any mesh deformation algorithm based on RBF interpolation consists of two main steps: (a) the solution of Eq. (6) and (7) for obtaining the appropriate RBF weights and (b) the evaluation of the displacements at all nodes inside the domain through Eq. (1).

Subproblem (a) involves the solution of the dense system of equations $\Phi_b \mathbf{x} = \mathbf{b}$ for various right hand sides $\mathbf{b}$. If a direct solver is employed for doing this, the computation time scales unfavorably as $\mathcal{O}(N_b^3)$. Iterative solvers might be an appealing alternative but suffer from the fact that the RBF-matrix Φ_b is often ill-conditioned (hence necessitating a large number of iterations), especially for large problems. This requires the incorporation of a preconditioning strategy, which is a difficult problem. Furthermore, they do not lead to a factorization of the matrix (as opposed to direct methods), implying that the algorithm needs to be restarted if the right hand side of the system is modified.

Solving $\Phi_b \mathbf{x} = \mathbf{b}$ in subproblem (a) is thus the bottleneck that forms the main limitation for applying

RBF interpolation to large 3D problems. The present paper therefore aims at alleviating this drawback through the use of a fast solution technique with only $\mathcal{O}(N_b)$ cost and memory requirement, as will be discussed in Section 3. It is furthermore emphasized that the solution of subproblem (b) leads to a computational cost scaling as $\mathcal{O}(N_b N_d)$ if a naive implementation is employed for Eq. (1). This matrix-vector product can be accelerated to $\mathcal{O}(N_d)$, however, through the use of fast summation schemes such as the fast multipole method [57].

3. The inverse fast multipole method (IFMM)

3.1. $\mathcal{H}^2$ -matrices

The development of a fast direct method for solving Eq. (5) is based on the observation that the dense RBF-matrix Φ_b can be represented in a hierarchical format. More precisely, the original dense matrix is subdivided into a hierarchy of smaller block matrices and most of these blocks are replaced by low-rank approximations. This can be done due to the fact that Φ_b is given by a smooth kernel. Several categories of hierarchical matrices (with increasing complexity) have been developed in the past decades, see for example [58, 59] for a detailed overview of various hierarchical structures³. The most general category encompasses so-called $\mathcal{H}$ - and $\mathcal{H}^2$ -matrices, in which only interactions between non-neighboring clusters of points are assumed to be low-rank ($\mathcal{H}^2$ -matrices can be seen as an algebraic generalization of the well-known fast multipole method). Because of its computational efficiency, this framework is considered here for general 3D problems.

The different character of neighboring and non-neighboring interactions is illustrated in the following using a simple 3D example. Figure 2 shows three clusters of points ($\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$) in the cubes $[0, 1]^3$, $[1, 2]^3$, and $[2, 3]^3$, respectively; each cluster contains N points. The matrices characterizing the interaction between points of clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ are denoted as $\Phi_{b_{ij}}$. Figure 3 shows the singular values σ_k/σ_0 of the matrices $\Phi_{b_{11}}$, $\Phi_{b_{12}}$, and $\Phi_{b_{13}}$, for the case of the multiquadratic biharmonic radial basis function $\phi(r) = \sqrt{a^2 + r^2}$ (with $a = 10^{-3}$)⁴ and for three different values of N . It is clear from Figure 3a that the matrix $\Phi_{b_{11}}$ (characterizing the self-interaction of cluster $\mathcal{C}_1$) should be treated as a full-rank matrix, as the singular values decay very slowly. A much faster decay is observed for the matrix $\Phi_{b_{12}}$ in Figure 3b, implying that this matrix – representing a neighboring interaction – could potentially be considered as low-rank. The rank that is required for a given relative accuracy ε increases with N , however, hampering the computational efficiency of algorithms that treat these interactions as low-rank (e.g., if the matrix is represented as a hierarchically off-diagonal low-rank (HODLR) [60] or a hierarchically semi-separable (HSS)

³Traditionally, hierarchical matrix representations have been exploited to accelerate matrix-vector multiplications, paving the way for the application of iterative solvers (e.g., CG, GMRES, ...). In contrast, a direct solver is presented here.

⁴The value of the parameter a is chosen to ensure that the matrix $\Phi_{b_{11}}$ is reasonably conditioned.

matrix [61, 62]). The only cases in which the decay of singular values remains nearly independent of N (and the matrix is thus truly low-rank) are non-neighboring interactions, as is illustrated in Figure 3c for $\Phi_{b_{13}}$. In $\mathcal{H}$ - and $\mathcal{H}^2$ -matrices, only non-neighboring interactions such as $\Phi_{b_{13}}$ are assumed to be low-rank; neighboring- and self-interactions are treated as full rank matrices.

Figure 2: Three clusters of points: cluster $\mathcal{C}_1$ in the cube $[0, 1]^3$ (blue), $\mathcal{C}_2$ in the cube $[1, 2]^3$ (orange), and $\mathcal{C}_3$ in the cube $[2, 3]^3$ (yellow). Each cluster contains N points.

It is important to note that the decay of singular values depends on the type of kernel. This is illustrated in Figure 4, showing the singular values σ_k/σ_0 of the matrices $\Phi_{b_{11}}$, $\Phi_{b_{12}}$, and $\Phi_{b_{13}}$ for the case of a Gaussian kernel $\phi(r) = e^{-a^2 r^2}$ (with $a = 5$). This example illustrates once more that the rank is (nearly) independent of N for the non-neighboring interaction $\Phi_{b_{13}}$, while this is not the case for $\Phi_{b_{12}}$.

3.2. Extended sparsification

The inverse fast multipole method (IFMM) has recently been developed as one of the first fast (approximate) direct solvers that achieves a linear complexity $\mathcal{O}(N_b)$ for systems of equations involving $\mathcal{H}^2$ -matrices [50, 51]. The solver is inexact, although the error can be controlled and made as small as needed. The two main ideas of this method are summarized in this and the next subsection; the reader is referred to [50, 51] for a detailed overview of the methodology.

A first step is to exploit the hierarchical structure of the matrix in order to convert the original dense system into an extended sparse system, as solving the latter is more efficient from a computational point of view. The sparsification procedure is illustrated for a 2D domain containing the boundary nodes of the mesh shown in Figure 1. This domain is hierarchically subdivided into clusters $\mathcal{C}_i^{(l)}$ using a quadtree

Figure 3: Relative singular values σ_k / σ_0 for matrices (a) $\Phi_{b_{11}}$, (b) $\Phi_{b_{12}}$, and (c) $\Phi_{b_{13}}$, with subscripts referring to the clusters of points in Figure 2. The matrix size varies from $N = 250$ (red line) to $N = 500$ (green line) and $N = 1000$ (blue line). The multiquadratic biharmonic radial basis function $\phi(r) = \sqrt{a^2 + r^2}$ with $a = 10^{-3}$ is employed.

Figure 4: Relative singular values σ_k / σ_0 for matrices (a) $\Phi_{b_{11}}$, (b) $\Phi_{b_{12}}$, and (c) $\Phi_{b_{13}}$, with subscripts referring to the clusters of points in Figure 2. The matrix size varies from $N = 250$ (red line) to $N = 500$ (green line) and $N = 1000$ (blue line). The Gaussian radial basis function $\phi(r) = e^{-a^2 r^2}$ with $a = 5$ is employed.

decomposition in Figure 5 (with super- and subscripts $^{(l)}$ and $_i$ referring to the hierarchical level and the cluster index at that level, respectively) until the leaf level $l = L$ is reached (with $L = 3$ in Figure 5).

This allows identifying cluster pairs $\mathcal{C}_i^{(l)}$ and $\mathcal{C}_j^{(l)}$ that are well-separated (non-neighboring), implying that the corresponding matrix block in Φ_b can be replaced by a low-rank approximation of rank r , i.e., $\Phi_{b_{ij}}^{(l)} \simeq \mathbf{U}_i^{(l)} \mathbf{K}_{ij}^{(l)} \mathbf{V}_j^{(l)\top}$ (see Figure 6b). The rank r is supposed to be considerably smaller than the dimensions of the matrix block under concern. The matrices $\mathbf{U}_i^{(l)}$ and $\mathbf{V}_j^{(l)\top}$ are denoted as the inter- and antepolation operators of clusters $\mathcal{C}_i^{(l)}$ and $\mathcal{C}_j^{(l)}$ ⁵. Examples of such cluster pairs at level $l = 2$ and $l = 3$ are highlighted in yellow and blue in Figure 5, respectively. Note that a nested structure is employed, i.e., the low-rank basis at a certain hierarchical level is constructed using the low-rank basis at the child level. Interactions between neighboring cluster pairs (including self interactions), on the other hand, are treated as full rank matrices. The resulting matrix structure for the case of Figure 5 is shown in Figure 6a⁶. The nested structure of the $\mathcal{H}^2$ -matrix is not visible in this figure.

Figure 5: Quadtree decomposition of a 2D domain into clusters $\mathcal{C}_i^{(l)}$. A neighboring cluster pair at level $l = 3$ is indicated in orange, a well-separated cluster pair at level $l = 3$ is indicated in blue, and a well-separated cluster pair at level $l = 2$ is indicated in yellow.

The dense matrix Φ_b can now recursively be decomposed into a sparse matrix (representing the “near field” interactions at the hierarchical level under concern) and a low-rank approximation (representing the “far field” interactions at that level and all higher levels), starting from the leaf level. This is done in a multilevel fashion. For the example of Figure 5, this decomposition at the leaf level reads as:

$$\Phi_b = \mathbf{S}^{(3)} + \mathbf{U}^{(3)} \mathbf{K}^{(3)} \mathbf{V}^{(3)\top} \quad (8)$$

The matrix $\mathbf{S}^{(3)}$ in Eq. (8) is a sparse matrix containing the full rank interactions of leaf clusters with their neighbors (including the self-interactions), while the matrix $\mathbf{U}^{(3)} \mathbf{K}^{(3)} \mathbf{V}^{(3)\top}$ represents the dense fill-ins arising from far field interactions. This decomposition is pictorially depicted in Figure 7.

⁵Interpolation = from coarse to fine. Antepolation = from fine to coarse.

⁶The different sizes of the matrix blocks are an indication of the fact that the clusters contain a different amount of nodes.

Figure 6: (a) Approximation of the dense matrix by an $\mathcal{H}^2$ -matrix. Red and green blocks correspond to full and low-rank matrices (at several levels), respectively. (b) Low rank approximation of an interaction between well-separated clusters: $\Phi_{b_{ij}}^{(l)} \simeq \mathbf{U}_i^{(l)} \mathbf{K}_{ij}^{(l)} \mathbf{V}_j^{(l)\top}$.

Figure 7: Decomposition of the $\mathcal{H}^2$ -matrix Φ_b at the leaf level L (with $L = 3$): $\Phi_b = \mathbf{S}^{(3)} + \mathbf{U}^{(3)} \mathbf{K}^{(3)} \mathbf{V}^{(3)\top}$.

The matrix $\mathbf{K}^{(3)}$ in Eq. (8) is representative for all far field interactions. It is still rather dense and is therefore further decomposed using a similar decomposition as for Φ_b :

$$\Phi_b = \mathbf{S}^{(3)} + \mathbf{U}^{(3)} \left(\mathbf{S}^{(2)} + \mathbf{U}^{(2)} \mathbf{K}^{(2)} \mathbf{V}^{(2)\top} \right) \mathbf{V}^{(3)\top} \quad (9)$$

This decomposition at level $l = 2$ is illustrated in Figure 8. The sparse matrix $\mathbf{S}^{(2)}$ in Eq. (9) represents near field interactions at level $l = 2$ (that have not been accounted for through $\mathbf{S}^{(3)}$), while $\mathbf{K}^{(2)}$ represents the remaining far field interactions. As no well-separated clusters exist at level $l = 1$, the matrix $\mathbf{K}^{(2)}$ can not be further decomposed.

Figure 8: Decomposition of the matrix $\mathbf{K}^{(3)}$ at level l (with $l = 2$): $\mathbf{K}^{(3)} = \mathbf{S}^{(2)} + \mathbf{U}^{(2)} \mathbf{K}^{(2)} \mathbf{V}^{(2)\top}$.

Eq. (9) is still a dense system, but can now be converted into an extended sparse system by introducing auxiliary variables at each level that contains well-separated clusters (i.e., $l \geq 2$). More specifically, variables $\mathbf{y}^{(l)}$ are defined:

$$\mathbf{y}^{(3)} = \mathbf{V}^{(3)\top} \mathbf{x} \quad (10)$$

$$\mathbf{y}^{(2)} = \mathbf{V}^{(2)\top} \mathbf{y}^{(3)} \quad (11)$$

This allows simplifying Eq. (9) into:

$$\mathbf{S}^{(3)} \mathbf{x} + \mathbf{U}^{(3)} \left(\mathbf{S}^{(2)} \mathbf{y}^{(3)} + \mathbf{U}^{(2)} \mathbf{K}^{(2)} \mathbf{y}^{(2)} \right) = \mathbf{b} \quad (12)$$

Furthermore, variables $\mathbf{z}^{(l)}$ are identified at each level:

$$\mathbf{z}^{(2)} = \mathbf{K}^{(2)} \mathbf{y}^{(2)} \quad (13)$$

$$\mathbf{z}^{(3)} = \mathbf{S}^{(2)} \mathbf{y}^{(3)} + \mathbf{U}^{(2)} \mathbf{z}^{(2)} \quad (14)$$

The original equation $\Phi_b \mathbf{x} = \mathbf{b}$ thus becomes:

$$\mathbf{S}^{(3)} \mathbf{x} + \mathbf{U}^{(3)} \mathbf{z}^{(3)} = \mathbf{b} \quad (15)$$

Eq. (10), (11), (13), (14), and (15) provide an extended sparse system of equations:

$$\begin{bmatrix} \mathbf{S}^{(3)} & \mathbf{U}^{(3)} & & & \\ \mathbf{V}^{(3)\top} & & -\mathbf{I} & & \\ & -\mathbf{I} & \mathbf{S}^{(2)} & \mathbf{U}^{(2)} & \\ & & \mathbf{V}^{(2)\top} & & -\mathbf{I} \\ & & & -\mathbf{I} & \mathbf{K}^{(2)} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{z}^{(3)} \\ \mathbf{y}^{(3)} \\ \mathbf{z}^{(2)} \\ \mathbf{y}^{(2)} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \quad (16)$$

The extended sparse matrix in Eq. (16) (indicated as $\mathbf{E}$ in the following) contains matrices from the decompositions introduced in Eq. (8) and (9) and exhibits a symmetric pattern in terms of fill-in, as is shown in Figure 9. The Gaussian elimination and substitution process are thus performed on this sparse system rather than on the original dense system. Note that the dimensions of $\mathbf{S}^{(2)}$ and $\mathbf{K}^{(2)}$ are small compared to those of $\mathbf{S}^{(3)}$ (as they characterize low-rank approximations), implying that $\mathbf{E}$ is only moderately larger than the original dense matrix Φ_b .

This sparsification procedure is a very general technique that is not restricted to 2D problems with three hierarchical levels. Any dense $\mathcal{H}^2$ -matrix can be recursively decomposed into a sparse matrix and a low-rank approximation, starting from the leaf level, to obtain an extended sparsified matrix.

3.3. Compression of fill-ins

Transforming the dense $\mathcal{H}^2$ -matrix Φ_b into an extended sparse matrix is a crucial but insufficient step to obtain a fast solver. Indeed, feeding the matrix $\mathbf{E}$ to a conventional sparse solver leads to the creation of dense fill-ins throughout the Gaussian elimination phase, which is harmful for the computational cost of the algorithm. In order to preserve the initial sparsity pattern, a second key idea is therefore introduced: fill-ins corresponding to interactions between well-separated clusters are compressed into a low-rank representation and are subsequently redirected through other operators. For example, fill-ins $\mathbf{E}'_{ij}$ and $\mathbf{E}'_{ji}$ arising between well-separated clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ are compressed as:

$$\mathbf{E}'_{ij} \simeq \mathbf{U}'_i \mathbf{K}'_{ij} \mathbf{V}'_j{}^\top \quad (17)$$

$$\mathbf{E}'_{ji} \simeq \mathbf{U}'_j \mathbf{K}'_{ji} \mathbf{V}'_i{}^\top \quad (18)$$

where a prime denotes a fill-in. The superscripts referring to the hierarchical level are omitted for sake of readability. It is assumed in the following that Eq. (17) and (18) represent partial singular value decompo-

Figure 9: A pictorial representation of the extended sparse matrix $\mathbf{E}$ of Eq. (16).

sitions (SVD), implying that the matrices $\mathbf{U}'$ and $\mathbf{V}'$ are orthonormal and that $\mathbf{K}'$ are diagonal matrices containing the singular values (i.e., $\mathbf{K}'_{ij} = \Sigma'_{ij}$ and $\mathbf{K}'_{ji} = \Sigma'_{ji}$). It is also assumed that a criterion is available for determining an appropriate rank in Eq. (17) and (18). These low-rank approximations allow for the calculation of new inter- and anteroposition operators for clusters $\mathcal{C}_i$ and $\mathcal{C}_j$. For example, a new interpolation operator $\hat{\mathbf{U}}_i$ is obtained by concatenating the existing scaled interpolation operator $\mathbf{U}_i \Sigma_{\mathbf{U}_i}$ (where $\Sigma_{\mathbf{U}_i}$ contains the weights associated with each column of $\mathbf{U}_i$) and the scaled fill-in operator $\mathbf{U}'_i \Sigma'_{ij}$ and subsequently applying a low-rank recompression (thin SVD):

$$[\mathbf{U}_i \Sigma_{\mathbf{U}_i} | \mathbf{U}'_i \Sigma'_{ij}] \simeq \hat{\mathbf{U}}_i \hat{\Sigma}_{\mathbf{U}_i} \Psi_i \quad (19)$$

$\Psi_i = [\psi_i | \psi'_i]$ represents a fat, row orthogonal, matrix. Eq. (19) allows expressing the existing and fill-in operators $\mathbf{U}_i$ and $\mathbf{U}'_i$ in terms of the new basis $\hat{\mathbf{U}}_i$:

$$\mathbf{U}_i \simeq \hat{\mathbf{U}}_i \hat{\Sigma}_{\mathbf{U}_i} \psi_i \Sigma_{\mathbf{U}_i}^{-1} = \hat{\mathbf{U}}_i \mathbf{r}_i \quad (20)$$

$$\mathbf{U}'_i \simeq \hat{\mathbf{U}}_i \hat{\Sigma}_{\mathbf{U}_i} \psi'_i \Sigma'^{-1}_{ij} = \hat{\mathbf{U}}_i \mathbf{r}'_i \quad (21)$$

The weights $\widehat{\Sigma}_{\mathbf{U}_i}$ of the new interpolation basis $\widehat{\mathbf{U}}_i$ are stored, as they are needed in potential future recompressions. A new anterpolation operator $\widehat{\mathbf{V}}_i$ is obtained in a similar way, such that:

$$\mathbf{V}_i \simeq \widehat{\mathbf{V}}_i \mathbf{t}_i \quad (22)$$

$$\mathbf{V}'_i \simeq \widehat{\mathbf{V}}_i \mathbf{t}'_i \quad (23)$$

The same strategy is applicable for obtaining new operators $\widehat{\mathbf{U}}_j$ and $\widehat{\mathbf{V}}_j$ for $\mathcal{C}_j$. As a result, the updated interaction between $\mathcal{C}_i$ and $\mathcal{C}_j$ can be written as:

$$\mathbf{U}_i \mathbf{K}_{ij} \mathbf{V}_j^T + \mathbf{U}'_i \mathbf{K}'_{ij} \mathbf{V}'_j{}^T = \widehat{\mathbf{U}}_i \mathbf{r}_i \mathbf{K}_{ij} \mathbf{t}_j^T \widehat{\mathbf{V}}_j^T + \widehat{\mathbf{U}}_i \mathbf{r}'_i \mathbf{K}'_{ij} \mathbf{t}'_j{}^T \widehat{\mathbf{V}}_j^T \quad (24)$$

$$= \widehat{\mathbf{U}}_i \left(\mathbf{r}_i \mathbf{K}_{ij} \mathbf{t}_j^T + \mathbf{r}'_i \mathbf{K}'_{ij} \mathbf{t}'_j{}^T \right) \widehat{\mathbf{V}}_j^T \quad (25)$$

Eq. (25) demonstrates that the fill-ins $\mathbf{E}'_{ij}$ and $\mathbf{E}'_{ji}$ can be folded into the existing operators $\mathbf{K}_{ij}$ and $\mathbf{K}_{ji}$, which have to be updated as follows:

$$\mathbf{K}_{ij} \leftarrow \mathbf{r}_i \mathbf{K}_{ij} \mathbf{t}_j^T + \mathbf{r}'_i \mathbf{K}'_{ij} \mathbf{t}'_j{}^T \quad (26)$$

$$\mathbf{K}_{ji} \leftarrow \mathbf{r}_j \mathbf{K}_{ji} \mathbf{t}_i^T + \mathbf{r}'_j \mathbf{K}'_{ji} \mathbf{t}'_i{}^T \quad (27)$$

Note that all the others operators associated with $\mathcal{C}_i$ and $\mathcal{C}_j$ have to be updated in a similar way (see [50, 51] for details). The crucial benefit of this strategy, consisting of folding the fill-in into existing operators, is that the amount of additional fill-in is minimized, ensuring that the sparsity pattern of $\mathbf{E}$ is preserved throughout the elimination phase.

This leads to an algorithm with linear complexity $\mathcal{O}(N_b)$, under the assumption that the rank of the compressed matrix blocks remains bounded independent of the matrix size (which is indeed expected to be the case, as only interactions between well-separated clusters are replaced by low-rank approximations in an $\mathcal{H}^2$ -matrix).

4. Numerical benchmarks

The IFMM algorithm has been implemented in C++, using the Eigen-library [63] for performing dense linear algebra operations. A synthetic example is considered in this section to demonstrate the efficiency of the IFMM for the solution of $\Phi_b \mathbf{x} = \mathbf{b}$ in Eq. (6) and (7). We solely focus on obtaining the RBF weights in this example; the evaluation of the displacements at all nodes inside the domain through Eq. (1) is not discussed here. All computations have been performed on Intel® Xeon® E5-2650 v2 (2.60 GHz) CPUs.

The example involves a spherical object with radius $r_i = 3$ embedded within a spherical domain with radius $r_o = 6$. The computational domain hence consists of a spherical shell with inner and outer radius

r_i and r_o , respectively; this domain is discretized with tetrahedrons using the mesh generator **Gmsh** [64] (Figure 10). A uniform translation of 0.50 in all directions is imposed on the nodes of the inner sphere, while the nodes at the outer boundary of the domain are fixed. A uniform octree decomposition is employed for obtaining a hierarchical clustering of the boundary nodes. The initial low-rank approximations in the RBF-matrix Φ_b are created by means of interpolation based on Chebyshev polynomials [57], followed by an additional truncated singular value decomposition to reduce the rank. A relative accuracy of 10^{-3} is used in this SVD compression, as well as in the low-rank compressions defined in Eq. (17), (18), and (19). The number of Chebyshev nodes (in each direction) is denoted as n ; the total number of nodes employed for constructing the initial low-rank approximations hence equals n^3 .

Figure 10: Initial mesh consisting of a spherical object with radius $r_i = 3$ embedded within a spherical domain with radius $r_o = 6$. The total number of boundary nodes equals $N_b = 46k$.

4.1. The IFMM as an approximate direct solver

4.1.1. Linear scaling of the computation time

The inverse fast multipole method is employed in this subsection as a direct solver for the systems $\Phi_b \mathbf{x} = \mathbf{b}$ arising in Eq. (6) and (7). The multiquadratic biharmonic radial basis function with global support $\phi(r) = \sqrt{a^2 + r^2}$ is used for the RBF interpolation, where a value of 10^{-3} has been assigned to the shape parameter a . Other kernels will be investigated in subsection 4.1.3.

Figure 11a shows the total time taken to obtain the weights α and β as a function of the number of boundary nodes N_b , for $n = 4$ and $n = 5$. Superimposed is the time required by a classical direct solver (based on a partially pivoted LU factorization). While the computation time scales as $\mathcal{O}(N_b^3)$ for the latter case, a linear complexity $\mathcal{O}(N_b)$ is achieved with the IFMM. The computation time grows if n is increased,

but the linear trend remains. The kinks in the curves of the IFMM are caused by varying the number of levels in the octree. This novel methodology hence clearly outperforms the direct LU factorization, especially for large problems. For the largest problem under concern ($N_b = 544k$), the IFMM takes 388 s (for $n = 4$), which is only a fraction of the (estimated) 9.5×10^6 s for the LU factorization of this problem. The IFMM is slightly slower for very small problems ($N_b < 4k$), however, due to the overhead associated with initializing the algorithm (construction of an octree decomposition, low-rank approximations, ...).

Figure 11: (a) Total CPU time and (b) the relative error $\|\gamma_{LU} - \gamma_{IFMM}\|_2 / \|\gamma_{LU}\|_2$ as a function of the number of boundary nodes N_b . A relative accuracy of 10^{-3} is used in the low-rank compressions.

The accuracy of the IFMM is assessed in Figure 11b, which depicts the relative error $\|\gamma_{LU} - \gamma_{IFMM}\|_2 / \|\gamma_{LU}\|_2$, with $\gamma = \{\alpha; \beta\}$. The error has only been calculated in the range where γ_{LU} could be computed within a reasonable time. The figure clearly shows a decrease of the error as n is increased, as the low-rank approximations become more accurate. The error increases with the problem size, however, which can be attributed to the inherent ill-conditioning of the RBF matrix. It is well known that the condition number κ of the RBF matrix grows with the problem size, see for example [55, 65]. This is also illustrated for the case under concern in Figure 12, showing κ as a function of the number of boundary nodes N_b .

The errors shown in Figure 11b are reasonably small, but one might not be satisfied with the obtained accuracy. In order to further reduce the error (e.g., to achieve 10 or more digits of accuracy), the number of Chebyshev nodes needs to be increased significantly, which leads to a strong increase of the computational cost (even if the linear complexity is preserved). It is therefore more efficient to apply a low accuracy IFMM solver as a preconditioner in an iterative method rather than using a highly accurate direct IFMM solver, as will be illustrated in Subsection 4.2.

4.1.2. Comparison with a coarsening strategy

As mentioned in the introduction, an alternative approach for reducing the computational cost is to use a coarsening strategy for selecting a small subset of boundary nodes [31, 32]. This approach is illustrated in the following paragraphs for comparison, using a greedy method proposed by Rendall and Allen [30] for the

Figure 12: Condition number κ as a function of the number of boundary nodes N_b .

selection of the boundary nodes. The method starts with an initial (arbitrary) selection of nodes, calculates the interpolation error at the unused nodes, and adds the node(s) where the largest error(s) is (are) observed to the list of selected nodes⁷. This procedure is repeated until the maximum error drops below the desired tolerance (e.g., 10^{-3}). The error E at an unused boundary node $\mathbf{x}_k$ is defined as:

$$E(\mathbf{x}_k) = \sqrt{(u_x(\mathbf{x}_k) - \bar{u}_x(\mathbf{x}_k))^2 + (u_y(\mathbf{x}_k) - \bar{u}_y(\mathbf{x}_k))^2 + (u_z(\mathbf{x}_k) - \bar{u}_z(\mathbf{x}_k))^2} \quad (28)$$

where $u_i(\mathbf{x}_k)$ and $\bar{u}_i(\mathbf{x}_k)$ indicate the computed and prescribed displacement in the direction i at a node $\mathbf{x}_k$, respectively.

A problem with $N_b = 46k$ boundary nodes is considered here (9k nodes on the inner sphere, 37k nodes on the outer sphere), with the same deformation pattern as before. Figure 13a shows the maximum interpolation error on the unused boundary nodes ($\max_k E(\mathbf{x}_k)$) as a function of the number of selected boundary nodes (one node at a time is added to the list of selected nodes). Convergence of the error is relatively fast, and only 1114 boundary nodes are needed if a tolerance of 10^{-3} is specified, demonstrating that a coarsening approach is very efficient in this specific case with a smooth surface deformation.

One could therefore argue that the results presented in Figure 11a are of limited practical value, as it is not necessary to solve a $46k \times 46k$ dense system, but only a $1k \times 1k$ system for which an LU-factorization is very cheap. This might be true for the specific example under concern but it does not hold in general, however, especially if a more complex displacement pattern is imposed. For example, assume that the following radial displacements are prescribed on the boundary of the inner sphere (the nodes on the outer boundary remain fixed):

$$u_r(\theta, \phi) = A \cos(k_\theta \theta) \cos(k_\phi \phi) \quad (29)$$

⁷The methodology that we use here is very simple and demonstrates the main idea of mesh reduction methods. More complicated (and more efficient) algorithms exist, however, see for example [31, 32].

Figure 13: Maximum error on the unused boundary nodes as a function of the number of selected boundary nodes, for displacements on the inner sphere corresponding to (a) a uniform translation in all directions and (b) radial displacements given by Eq. (29).

with θ and ϕ representing the polar and azimuthal angle, respectively, and $A = 0.50$ the amplitude of the deformation. k_θ and k_ϕ are wavenumbers determining the wavelengths of the deformation in each direction. Figure 14a and 14b illustrate the radial displacements $u_r(\theta, \phi)$ imposed on the inner sphere for the cases of $k_\theta = k_\phi = 4$ and $k_\theta = k_\phi = 8$, respectively. Increasing the values of k_θ and k_ϕ clearly results in a more oscillatory deformation field (smaller wavelengths).

Figure 14: Radial displacements $u_r(\theta, \phi)$ imposed on the inner sphere for (a) $k_\theta = k_\phi = 4$ and (b) $k_\theta = k_\phi = 8$.

Figure 13b shows the maximum interpolation error as a function of the number of selected boundary nodes, for the two deformation cases depicted in Figure 14. It is clear that a much larger subset of nodes is now required for an accurate representation of the mesh deformation (compare with Figure 13a): up to 7k and 9k nodes, respectively. In the latter case, this means that almost all the nodes on the inner sphere are selected. Application of the IFMM will hence provide a significant speed-up for solving the resulting $9k \times 9k$ dense system. Furthermore, the coarsening strategy in itself becomes computationally expensive in these cases, as we need to perform the RBF-interpolation (computing the RBF-weights for the current

selection of nodes and subsequently the interpolation error at all unused nodes) at every iteration step. Solving thousands of RBF-interpolation problems (that become larger and larger) with an LU-factorization is more expensive than applying the IFMM to the full size system.

In summary, this example illustrates that a coarsening strategy is not always sufficient to obtain a substantial reduction in computation time, especially if the imposed deformations are not smooth. This clearly reveals the benefit of the fast linear solver proposed in this paper.

Note that the greedy technique employed in this paper is not necessarily the most efficient one, as only one point at a time is added to the list of selected nodes, and a full dense solve is performed at every iteration step. Adding multiple points at a time, or delaying the full dense solves by single points additions and corrections, are alternative approaches that are able to make the greedy approach faster. More details about such strategies can be found, for example, in [30, 31].

4.1.3. Parametric study

The results presented so far have been obtained with the multiquadratic biharmonic RBF. In the following, the efficiency and accuracy of the IFMM are also assessed for the other kernels listed in Table 1.

Figure 15a shows the computation time as a function of N_b for the Gaussian, thin plate spline, multiquadratic biharmonic, and inverse multiquadratic biharmonic radial basis functions, and this for $n = 4$ and $n = 5$ (i.e., the number of Chebyshev nodes in each direction). The computation time is almost independent of the kernel; the application of the IFMM results in a linear trend for all considered kernels. The relative error for each of these cases (with respect to an LU factorization) is depicted in Figure 15b. Compared to the multiquadratic biharmonic RBF, considerably smaller errors are obtained for the Gaussian and the inverse multiquadratic biharmonic RBF (for the same value of n), while the errors for the thin plate spline are of the same order of magnitude. These results demonstrate that the IFMM is applicable for a wide range of radial basis functions with global support.

4.2. The IFMM as a preconditioner in an iterative solver

As indicated at the end of subsection 4.1.1, it might be more efficient to apply a low accuracy IFMM solver as a preconditioner in an iterative method rather than using a highly accurate direct IFMM solver. In this subsection, Eq. (5) is solved iteratively for α and β using a non-restarted GMRES-algorithm [66, 67] with a tolerance $\varepsilon = 10^{-8}$ for the relative residual. The black-box FMM [57] is employed for a fast evaluation of the matrix-vector products within GMRES. It is desirable to accelerate the convergence of the iterative solver through the incorporation of a suitable preconditioner. The optimal preconditioner is the inverse of the coefficient matrix:

$$\mathbf{M} = \begin{bmatrix} \Phi_b^{-1} \left(\mathbf{I} - \mathbf{P}_b (\mathbf{P}_b^T \Phi_b^{-1} \mathbf{P}_b)^{-1} \mathbf{P}_b^T \Phi_b^{-1} \right) & \Phi_b^{-1} \mathbf{P}_b (\mathbf{P}_b^T \Phi_b^{-1} \mathbf{P}_b)^{-1} \\ (\mathbf{P}_b^T \Phi_b^{-1} \mathbf{P}_b)^{-1} \mathbf{P}_b^T \Phi_b^{-1} & -(\mathbf{P}_b^T \Phi_b^{-1} \mathbf{P}_b)^{-1} \end{bmatrix} \quad (30)$$

Figure 15: (a) Total CPU time and (b) the relative error $\|\gamma_{LU} - \gamma_{IFMM}\|_2 / \|\gamma_{LU}\|_2$ as a function of the number of boundary nodes N_b , for the Gaussian (GSN), thin plate spline (TPS), multiquadratic biharmonic (MQB), and inverse multiquadratic biharmonic (IMQB) radial basis functions.

A good approximation of this optimal preconditioner is obtained by employing the IFMM for solving the systems $\Phi_b \mathbf{x} = \mathbf{b}$ in Eq. (30). Note that it is not necessary to assemble the preconditioner $\mathbf{M}$ explicitly; only the effect of $\mathbf{M}$ on a vector needs to be evaluated throughout the iterative process.

In a first example, a problem with $N_b = 46k$ boundary nodes is considered. The multiquadratic biharmonic RBF with $a = 10^{-3}$ is used once more, and a uniform translation of 0.50 in all directions is imposed on the nodes of the inner sphere. Figure 16a shows the relative residual as a function of the number of GMRES iterations for several preconditioning approaches. Without preconditioner, 266 iterations are required to achieve convergence, resulting in a total computation time of 705 s. This is already a substantial reduction with respect to the time required for an LU factorization (i.e., 5945 s, see Figure 11a). The application of the IFMM as preconditioner leads to a much faster convergence of GMRES, reducing the number of iterations to 9 for $n = 4$ and even down to 3 for $n = 5$. The computation times for the considered preconditioning strategies are compared in Figure 16b, where a decomposition is made into the time required to construct the preconditioner (i.e., the elimination phase in the IFMM) and the actual iteration time (i.e., the substitution phase in the IFMM, together with all GMRES-related computations). This figure illustrates the trade-off between a preconditioner's accuracy (and thus the number of iterations) and its construction cost. For the example under concern, the use of four Chebyshev nodes per dimension leads to the smallest overall computation time. The results are summarized in Table 2.

	# iterations	Total CPU time [s]	$\ \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}\ _2 / \ \mathbf{b}\ _2$
No precon	266	705	9.9×10^{-9}
IFMM ($n = 4$)	9	89	8.8×10^{-9}
IFMM ($n = 5$)	3	120	8.2×10^{-9}

Table 2: Overview of the results for various preconditioning strategies for $N_b = 46k$.

Figure 16: (a) Relative residual as a function of the iteration step and (b) total CPU time for $N_b = 46k$.

Next, a much larger problem with $N_b = 325k$ boundary nodes is tackled, which leads to a more ill-conditioned RBF-matrix Φ_b . Figure 17a demonstrates that the unpreconditioned GMRES-solver needs a very large number of iterations to converge. Application of the IFMM as preconditioner accelerates the convergence, although the use of four Chebyshev nodes per direction is not as efficient as in the previous example. The additional cost associated with increasing the number of Chebyshev nodes from $n = 4$ to $n = 5$ clearly pays off in this case, as it results in a remarkable reduction of the number of iterations as well as of the overall computation time (see Figure 17b and Table 3).

Figure 17: (a) Relative residual as a function of the iteration step and (b) total CPU time for $N_b = 325k$.

	# iterations	Total CPU time [s]	$\ \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}\ _2 / \ \mathbf{b}\ _2$
No precon	552	10905	9.9×10^{-9}
iFMM ($n = 4$)	181	6428	9.5×10^{-9}
iFMM ($n = 5$)	10	1237	9.8×10^{-9}

Table 3: Overview of the results for various preconditioning strategies for $N_b = 325k$.

5. Applications

The synthetic benchmarks considered in the previous section demonstrate the numerical efficiency of the IFMM as a direct solver or as a preconditioner. Two applications are considered in this section to illustrate the method’s applicability to engineering problems, using the open-source CFD code SU2 [52, 53]. Incorporating the RBF-based mesh deformation strategy into the SU2-code is minimally intrusive, as it only requires knowledge of the physical coordinates of the grid points and the imposed displacements on the boundaries.

5.1. Deformation of the ONERA M6 wing

The first application involves the three-dimensional ONERA M6 wing, a classic validation case for CFD codes. The wing is discretized with almost 3k nodes on its boundary and is embedded in a semi-spherical computational domain, as is shown in Figure 18a. A close-up near the wing is depicted in Figure 18b. The outer boundary of the computational domain contains 19k nodes. The volume discretization of the computational domain leads to a total of 545k tetrahedral elements (96k nodes).

Figure 18: (a) Wireframe representation of the computational domain and (b) close-up near the ONERA M6 wing.

A lateral deflection of 0.25 m is imposed (in one step) on the wingtip, which leads to the deformed wing shape illustrated in Figure 19. The volume mesh needs to be deformed accordingly. This example only focuses on this single mesh deformation step and does not involve a fluid computation.

Results obtained with two mesh deformation strategies are discussed in the following. On the one hand, a mesh deformation technique based on a linear elasticity analogy is used, in which the stiffness of each element is chosen inversely proportional to its volume. This is the standard deformation approach in SU2. The linear elasticity analogy leads to a sparse system of equations (involving all the elements in the computational domain) that is solved by means of an iterative solver (FGMRES with a symmetric Gauss-Seidel preconditioner). This solver fails to converge to the specified residual of 10^{-8} within 500 iterations (i.e., the standard maximum number of iterations in SU2), leading to elements with negative

Figure 19: (a) Front and (b) side view of the undeformed (gray) and deformed (black) ONERA M6 wing.

volume. Increasing this limit allows the solver to converge after 638 iterations. The total computation time for this deformation approach is 606 s.

On the other hand, RBF-based mesh deformation is applied as an alternative, using the multiquadric biharmonic radial basis function with global support $\phi(r) = \sqrt{a^2 + r^2}$ with a shape parameter $a = 5 \times 10^{-2}$. Application of a standard direct solver (LU factorization) for the determination of the RBF weights requires 483 s. The use of GMRES (with a tolerance $\varepsilon = 10^{-8}$ for the relative residual) without preconditioner does not lead to converge to the desired residual within 500 iterations. Fast convergence with GMRES is achieved, however, if the IFMM is employed as preconditioner (using five Chebyshev nodes per dimension and a relative accuracy of 10^{-3} for the low-rank compressions). The number of iterations is limited to 36 and the computation time is only 13 s. Once the RBF weights have been determined, the black-box FMM (described in [57]) is employed for a fast evaluation of Eq. (1) to obtain the mesh movement of the internal nodes. This requires an additional 30 s. A coarsening approach might be a valuable alternative to speed up the RBF-interpolation in this specific example, given the fact that a rather smooth deformation is applied to the wing.

The results are summarized in Table 4, illustrating that the overall computation time for the RBF-based mesh deformation approach is thus drastically reduced to only 43 s through the application of the IFMM as preconditioner in an iterative solver. In this particular case, the RBF-based approach becomes an order of magnitude faster than the standard technique available in SU2. Other elasticity/spring analogy methods might be more efficient than the one employed here, however.

The mesh quality of the deformed mesh is similar for both of the aforementioned mesh deformation methods. The deformed mesh near the wing is visualized in Figure 20 and 21. If the deformation imposed on the wingtip is twice as large, the linear elasticity technique fails in deforming the mesh appropriately, as it leads to negative elements. Application of the RBF-based method, on the other hand, ensures that no negative elements are created, although it is necessary to adjust the shape parameter to $a = 5 \times 10^{-2}$

	Total CPU time [s]
Linear elasticity	606
RBF (LU factorization)	513
RBF (GMRES)	(No convergence)
RBF (GMRES + IFMM)	43

Table 4: Overview of the results for various mesh deformation strategies applied to the ONERA M6 wing. For the RBF-based method, GMRES without preconditioner does not converge within 500 iterations.

in order to achieve this (i.e., a ‘flatter’ radial basis function). This leads to a larger number of GMRES iterations (95 vs. 36) and a larger overall computation time (76 s vs. 43 s). Note that more advanced spring analogy methods than the one available in **SU2** (such as those described in [7, 8, 13]) are better suited to handle large deformations and are potentially able to avoid the reported breakdown.

Figure 20: Wireframe representation of the initial (black) and deformed (red) mesh near the ONERA M6 wing.

5.2. Pitching and plunging of a NACA 0012 airfoil

A 2D case study of a pitching and plunging airfoil is considered in this subsection; this example involves an actual fluid computation. [Figure 22](#) shows a NACA 0012 airfoil with a chord length of 1 m and the surrounding mesh of triangular elements. The mesh consists of $N_b = 8k$ boundary nodes (i.e., situated on the airfoil and the outer boundary of the computational domain) and $N_d = 343k$ domain nodes, resulting in a total of 680k elements. The problem size is rather small in order to be able to perform the fluid computations on a single core. A combined pitching and vertical plunging motion is prescribed on the airfoil, while the nodes on the outer boundary of the domain remain fixed. The pitching angle and plunging amplitude equal 25° and 0.75 m, respectively, and both motions have a period of 0.05 s. The Euler equations are employed for the unsteady flow calculations by means of **SU2**, using a second order dual time stepping scheme with a time step $\Delta t = 0.002$ s (i.e., 25 simulations per period).

Figure 21: Deformed mesh near the ONERA M6 wing.

Figure 22: Triangular mesh around a NACA 0012 airfoil: (a) entire mesh and (b) close-up near the airfoil.

As in Subsection 5.1, results obtained with two mesh deformation strategies are discussed in the following, i.e., the linear elasticity analogy and the RBF-based mesh deformation technique. For the latter, we once again consider three approaches for obtaining the RBF-weights: a standard LU factorization, GMRES without preconditioner, and GMRES with the IFMM as preconditioner. The same parameters as in the previous subsection are used. Figure 23 shows the pressure field around the airfoil at three different time steps, i.e., at $t = 7\Delta t$, $t = 13\Delta t$, and $t = 19\Delta t$. As can be observed, similar results are obtained with both mesh deformation methods⁸. The mesh quality is assessed in Figure 24, depicting the average and maximum element aspect ratios as a function of the simulation step (an aspect ratio close to 1 indicates a good mesh quality). It is clear that the quality of the elements is well maintained with both methods, with a slightly better outcome for the RBF-based technique.

Figure 23: Pressure field around the NACA 0012 airfoil at (a) $t = 7\Delta t$, (b) $t = 13\Delta t$, and (c) $t = 19\Delta t$, using the linear elasticity equations (left) and RBF interpolation (right) for the mesh deformation.

The time required for performing the mesh deformation at every iteration step is summarized in Table 5, for the considered numerical techniques. The overall time spent in deforming the mesh equals 3700 s for the

⁸For the RBF-method, results obtained with GMRES and the IFMM as preconditioner are shown.

Figure 24: Average (dashed line) and maximum (solid line) element aspect ratio as a function of the simulation step, using the linear elasticity equations (red) and RBF interpolation (black) for the mesh deformation. An aspect ratio close to 1 indicates a good mesh quality.

linear elasticity approach, while application of the RBF-method with a standard LU factorization requires 2750 s. Although an LU factorization is affordable in this rather small example (with only 8k boundary nodes), its cubing scaling makes this approach not suitable for larger problems. The use of GMRES without preconditioner is significantly slower, leading to a computation time of 4300 s. Applying the IFMM as preconditioner results once more in a speed-up, limiting the time to 2500 s, and hence becoming the fastest of all approaches. The relative speed-up of the RBF-method by means of the IFMM is expected to be more significant for larger problems, cfr. the results presented in [Section 4](#). As the RBF-method also leads to elements with a slightly higher quality, the flow solver needs less iterations to converge, leading to a (modest) speed-up for that part of the calculations with respect to the linear elasticity technique as well. As mentioned in [Subsection 5.1](#), other elasticity/spring analogy methods than the one available in SU2 might be more efficient, however.

	Total CPU time [s]
Linear elasticity	3700
RBF (LU factorization)	2750
RBF (GMRES)	4300
RBF (GMRES + IFMM)	2500

Table 5: Overview of the results for various mesh deformation strategies applied to the NACA 0012 airfoil.

6. Conclusions

In this paper, the inverse fast multipole method has been presented as a technique that drastically reduces the computational cost associated with mesh deformation based on radial basis function interpolation. This technique aims at accelerating the underlying dense linear algebra operations, in contrast to other

developments in this field that try to reduce the number of boundary nodes (and hence the computational cost) using coarsening strategies [30, 31, 33]. Both strategies can be combined, however: first apply a coarsening technique to select a subset of boundary nodes, and subsequently use the IFMM to speed up the solution of the resulting (smaller) dense system of equations. Reducing the computational cost for solving a dense linear system from $\mathcal{O}(N_b^3)$ to $\mathcal{O}(N_b)$ remains significant, even if it is only applied to a subset of the surface points.

The dense RBF-matrix is represented as an $\mathcal{H}^2$ -matrix and its hierarchical structure is exploited to obtain a fast approximate direct solver with linear complexity $\mathcal{O}(N_b)$. This enables an efficient solution of the dense system of equations for obtaining the RBF weights. The method relies on two key ideas, i.e., performing an extended sparsification and minimizing the fill-in through low-rank compressions.

Numerical benchmarks have been performed to confirm the linear scaling of the algorithm, while it has also been demonstrated that the IFMM solver performs extremely well as a preconditioner in an iterative algorithm. Two case studies involving a wing and an airfoil have furthermore been discussed to illustrate the method’s applicability to engineering problems. The proposed methodology can straightforwardly be integrated in existing codes and is easily applicable to more complicated problems (see for example [33] for a case in which RBF-based mesh deformation is applied to the simulation of flows around flapping wings).

In conclusion, the IFMM allows to fully exploit the benefits of mesh deformation techniques based on RBF interpolation (i.e., high mesh quality preservation, robustness, easy implementation, ...) at an affordable computational cost. Important aspects to consider in future work include the parallelization of the IFMM-algorithm and its extension to adaptive hierarchical decompositions with a varying number of levels to cope even better with unstructured grids [68, 69].

Acknowledgments

The authors would like to thank Heather Kline and Thomas Economon at Stanford University for their assistance in obtaining the results presented in Section 5.

The first author is a post-doctoral fellow of the Research Foundation Flanders (FWO) and a Francqui Foundation fellow of the Belgian American Educational Foundation (BAEF). The financial support is gratefully acknowledged.

Part of this research was done at Stanford University, and was supported in part by the U.S. Army Research Laboratory, through the Army High Performance Computing Research Center, Cooperative Agreement W911NF-07-0027. This material is also based upon work supported by the Department of Energy under Award Number DE-NA0002373-1.

References

- [1] D. Sieger, S. Gaulik, J. Achenbach, S. Menzel, M. Botsch, Constrained space deformation techniques for design optimization, *Computer-Aided Design* 72 (2016) 40–51.
- [2] A. Johnson, T. Tezduyar, Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces, *Computer Methods in Applied Mechanics and Engineering* 119 (1–2) (1994) 73–94.
- [3] F. Blom, Considerations on the spring analogy, *International Journal for Numerical Methods in Fluids* 32 (6) (2000) 647–668.
- [4] A. de Boer, A. van Zuijlen, H. Bijl, Review of coupling methods for non-matching meshes, *Computer Methods in Applied Mechanics and Engineering* 196 (8) (2007) 1515–1525.
- [5] J. Batina, Unsteady Euler airfoil solutions using unstructured dynamic meshes, *AIAA Journal* 28 (8) (1990) 1381–1388.
- [6] C. Farhat, C. Degand, B. Koobus, M. Lesoinne, Torsional springs for two-dimensional dynamic unstructured fluid meshes, *Computer Methods in Applied Mechanics and Engineering* 163 (1998) 231–245.
- [7] D. Zeng, C. Ethier, A semi-torsional spring analogy model for updating unstructured meshes in 3D moving domains, *Finite Elements in Analysis and Design* 41 (11–12) (2005) 1118–1139.
- [8] G. Markou, Z. Mouroutis, D. Charmpis, M. Papadrakakis, The ortho-semi-torsional (OST) spring analogy method for 3D mesh moving boundary problems, *Computer Methods in Applied Mechanics and Engineering* 196 (4–6) (2007) 747–765.
- [9] C. Bottasso, D. Detomi, R. Serra, The ball-vertex method: a new simple spring analogy method for unstructured dynamic meshes, *Computer Methods in Applied Mechanics and Engineering* 194 (39–41) (2005) 4244–4264.
- [10] G. Chiandussi, G. Buggeda, E. Oñate, A simple method for automatic update of finite element meshes, *Communications in Numerical Methods in Engineering* 16 (1) (2000) 1–19.
- [11] R. Löhner, C. Yang, Improved ALE mesh velocities for moving boundaries, *Communications in Numerical Methods in Engineering* 12 (10) (1996) 599–608.
- [12] B. Helenbrook, Mesh deformation using the biharmonic operator, *International Journal for Numerical Methods in Engineering* 56 (7) (2003) 1007–1021.
- [13] R. Dwight, Robust mesh deformation using the linear elasticity equations, in: H. Deconinck, E. Dick (Eds.), *Computational Fluid Dynamics 2006: Proceedings of the Fourth International Conference on Computational Fluid Dynamics, ICCFD*, Ghent, Belgium, July 2006, Springer Berlin Heidelberg, 2009, pp. 401–406.
- [14] X. Liu, N. Qin, H. Xia, Fast dynamic grid deformation based on delaunay graph mapping, *Journal of Computational Physics* 211 (2) (2006) 405–423.
- [15] A. de Boer, M. van der Schoot, H. Bijl, Mesh deformation based on radial basis function interpolation, *Computers and Structures* 85 (11–14) (2007) 784–795.
- [16] O. Estruch, O. Lehmkuhl, R. Borrell, C. Pérez Segarra, A. Oliva, A parallel radial basis function interpolation method for unstructured dynamic meshes, *Computers and Fluids* 80 (2013) 44–54.
- [17] J. Witteveen, Explicit and robust inverse distance weighting mesh deformation for CFD, in: *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, Orlando, USA, 2010.
- [18] E. Luke, E. Collins, E. Blades, A fast mesh deformation method using explicit interpolation, *Journal of Computational Physics* 231 (2) (2012) 586–601.
- [19] E. Lefrançois, A simple mesh deformation technique for fluidstructure interaction based on a submesh approach, *International Journal for Numerical Methods in Engineering* 75 (9) (2008) 1085–1101.
- [20] D. Martineau, J. Georgala, A mesh movement algorithm for high quality generalised meshes, in: *42nd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, USA, 2004.
- [21] D. Kholodar, S. Morton, R. Cummings, Deformation of unstructured viscous grids, in: *43rd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, USA, 2005.

- [22] B. Prananta, I. Tjatra, S. Spekrijse, J. Kok, J. Meijer, Static aeroelastic simulation of military aircraft configuration in transonic flow, Tech. rep., NLR-TP-2001-346 (2001).
- [23] S. Spekrijse, B. Prananta, J. Kok, A simple, robust and fast algorithm to compute deformations of multi-block structured grids, Tech. rep., NLR-TP-2002-105 (2002).
- [24] M. Potsdam, G. Guruswamy, A parallel multiblock mesh movement scheme for complex aeroelastic applications, in: 39th AIAA Aerospace Sciences Meeting and Exhibit, Reno, USA, 2001.
- [25] U. Cella, M. Biancolini, Aeroelastic analysis of aircraft wind-tunnel model coupling structural and fluid dynamic codes, AIAA Journal of Aircraft 49 (2) (2012) 407–414.
- [26] M. Biancolini, I. Viola, M. Riotte, Sails trim optimisation using CFD and RBF mesh morphing, Computers and Fluids 93 (2014) 46–60.
- [27] M. Biancolini, Mesh morphing and smoothing by means of radial basis functions (RBF): A practical example using Fluent and RBF-Morph, in: J. Leng, W. Sharrock (Eds.), Handbook of Research on Computational Science and Engineering: Theory and Practice, 2012, pp. 347–380.
- [28] RBF4AERO: Reshaping the future of aircraft design, <http://www.rbf4aero.eu/>.
- [29] S. Jakobsson, O. Amoignon, Mesh deformation using radial basis functions for gradient-based aerodynamic shape optimization, Computers and Fluids 36 (6) (2007) 1119–1136.
- [30] T. Rendall, C. Allen, Efficient mesh motion using radial basis functions with data reduction algorithms, Journal of Computational Physics 228 (17) (2009) 6231–6249.
- [31] T. Rendall, C. Allen, Parallel efficient mesh motion using radial basis functions with application to multi-bladed rotors, International Journal for Numerical Methods in Engineering 81 (1) (2010) 89–105.
- [32] Y. Wang, N. Qin, N. Zhao, Delaunay graph and radial basis function for fast quality mesh deformation, Journal of Computational Physics 294 (2015) 149–172.
- [33] F. Bos, B. van Oudheusden, H. Bijl, Radial basis function based mesh deformation applied to simulation of flow around flapping wings, Computers and Fluids 79 (2013) 167–177.
- [34] G. Wang, H. Mian, Z.-Y. Ye, J.-D. Lee, Improved point selection method for hybrid-unstructured mesh deformation using radial basis functions, AIAA Journal 53 (4) (2015) 1016–1025.
- [35] A. Michler, Aircraft control surface deflection using RBF-based mesh deformation, International Journal for Numerical Methods in Engineering 88 (10) (2011) 986–1007.
- [36] G. Anjos, N. Borhani, N. Mangiavacchi, J. Thome, A 3D moving mesh finite element method for two-phase flows, Journal of Computational Physics 270 (2014) 366–377.
- [37] O. Livne, G. Wright, Fast multilevel evaluation of smooth radial basis function expansions, Electronic Transactions on Numerical Analysis 23 (2006) 263–287.
- [38] N. Gumerov, R. Duraiswami, Fast radial basis function interpolation via preconditioned Krylov iteration, SIAM Journal on Scientific Computing 29 (5) (2007) 1876–1899.
- [39] M. Bonnet, Boundary integral equation methods for solids and fluids, John Wiley & Sons, Chichester, United Kingdom, 1995.
- [40] R. Kalman, Mathematical description of linear dynamical systems, SIAM Journal on Control and Optimization 1 (2) (1963) 152–192.
- [41] P.-G. Martinsson, V. Rokhlin, A fast direct solver for boundary integral equations in two dimensions, Journal of Computational Physics 205 (1) (2005) 1–23.
- [42] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, T. Pals, A fast solver for HSS representations via sparse matrices, SIAM Journal on Matrix Analysis and Applications 29 (1) (2007) 67–81.
- [43] J. Xia, S. Chandrasekaran, M. Gu, X. Li, Fast algorithms for hierarchically semiseparable matrices, Numerical Linear

- Algebra with Applications 17 (6) (2010) 953–976.
- [44] K. Ho, L. Greengard, A fast direct solver for structured linear systems by recursive skeletonization, *SIAM Journal on Scientific Computing* 34 (5) (2012) 2507–2532.
 - [45] A. Gillman, P.-G. Martinsson, An $O(N)$ algorithm for constructing the solution operator to 2D elliptic boundary value problems in the absence of body loads, *Advances in Computational Mathematics* 40 (4) (2014) 773–796.
 - [46] E. Corona, P.-G. Martinsson, D. Zorin, A $O(N)$ direct solver for integral equations on the plane, *Applied and Computational Harmonic Analysis* 38 (2) (2015) 284–317.
 - [47] K. Ho, L. Ying, Hierarchical interpolative factorization for elliptic operators: Integral equations, *Communications on Pure and Applied Mathematics* (2015) 10.1002/cpa.21577.
 - [48] W. Hackbusch, A sparse matrix arithmetic based on $\mathcal{H}$ -matrices. Part I: Introduction to $\mathcal{H}$ -matrices, *Computing* 62 (2) (1999) 89–108.
 - [49] W. Hackbusch, B. N. Khoromskij, R. Kriemann, Hierarchical matrices based on a weak admissibility criterion, *Computing* 73 (2004) 207–243.
 - [50] S. Ambikasaran, E. Darve, The Inverse Fast Multipole Method, *arXiv* (2014) 1407.1572.
 - [51] P. Coulier, H. Pouransari, E. Darve, The inverse fast multipole method: using a fast approximate direct solver as a preconditioner for dense linear systems, *arXiv* (2015) 1508.01835.
 - [52] F. Palacios, M. R. Colonno, A. C. Aranake, A. Campos, S. R. Copeland, T. D. Economon, A. K. Lonkar, T. W. Lukaczyk, T. W. R. Taylor, J. J. Alonso, Stanford University Unstructured (SU2): An open-source integrated computational environment for multi-physics simulation and design, in: 51st AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, Grapevine, USA, 2013.
 - [53] F. Palacios, T. D. Economon, A. C. Aranake, S. R. Copeland, A. K. Lonkar, T. W. Lukaczyk, D. E. Manosalvas, K. R. Naik, A. S. Padron, B. Tracey, A. Variyar, J. J. Alonso, Stanford University Unstructured (SU2): Open-source analysis and design technology for turbulent flows, in: 52nd AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, National Harbor, USA, 2014.
 - [54] M. D. Buhmann, *Radial basis functions: theory and implementations*, Cambridge University Press, Cambridge, UK, 2003.
 - [55] H. Wendland, Error estimates for interpolation by compactly supported radial basis functions of minimal degree, *Journal of Approximation Theory* 93 (2) (1998) 258–272.
 - [56] A. Beckert, H. Wendland, Multivariate interpolation for fluid-structure-interaction problems using radial basis functions, *Aerospace Science and Technology* 5 (2) (2001) 125–134.
 - [57] W. Fong, E. Darve, The black-box fast multipole method, *Journal of Computational Physics* 228 (23) (2009) 8712–8725.
 - [58] M. Bebendorf, *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*, 1st Edition, Springer Publishing Company, 2008.
 - [59] S. Börm, L. Grasedyck, W. Hackbusch, Introduction to hierarchical matrices with applications, *Engineering Analysis with Boundary Elements* 27 (5) (2003) 405–422.
 - [60] A. Aminfar, S. Ambikasaran, E. Darve, A fast block low-rank dense solver with applications to finite-element matrices, *arXiv* (2014) 1403.5337.
 - [61] M. Chandrasekaran, S. Gu, T. Pals, A fast ULV decomposition solver for hierarchically semi-separable representations, *SIAM Journal on Matrix Analysis and Applications* 28 (3) (2006) 603–622.
 - [62] Z. Sheng, P. Dewilde, S. Chandrasekaran, Algorithms to solve hierarchically semi-separable systems, in: D. Alpay, V. Vinnikov (Eds.), *System Theory, the Schur Algorithm and Multidimensional Analysis*, Vol. 176 of *Operator Theory: Advances and Applications*, Birkhäuser Basel, 2007, pp. 255–294.
 - [63] G. Guennebaud, B. Jacob, et al., *Eigen v3*, <http://eigen.tuxfamily.org> (2010).
 - [64] C. Geuzaine, J.-F. Remacle, *Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities*,

- 1
2
3
4
5 International Journal for Numerical Methods in Engineering 79 (11) (2009) 1309–1331.
6
7 [65] J. Boyd, K. Gildersleeve, Numerical experiments on the condition number of the interpolation matrices for radial basis
8 functions, Applied Numerical Mathematics 61 (4) (2011) 443–459.
9 [66] Y. Saad, M. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM
10 Journal on Scientific and Statistical Computing 7 (3) (1986) 856–869.
11 [67] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, SIAM Journal on Scientific and Statistical Computing
12 14 (2) (1993) 461–469.
13 [68] L. Ying, G. Biros, D. Zorin, A kernel-independent adaptive fast multipole algorithm in two and three dimensions, Journal
14 of Computational Physics 196 (2) (2004) 591–626.
15 [69] H. Pouransari, E. Darve, Optimizing the adaptive fast multipole method for fractal sets, SIAM Journal on Scientific
16 Computing 37 (2) (2015) A1040–A1066.
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65