SIAM J. SCI. COMPUT.
Vol. 37, No. 4, pp. A1891–A1908

A TRIVARIATE INTERPOLATION ALGORITHM USING A CUBE-PARTITION SEARCHING PROCEDURE*

ROBERTO CAVORETTO† AND ALESSANDRA DE ROSSI†

Abstract. In this paper we propose a fast algorithm for trivariate interpolation, which is based on the partition of unity method for constructing a global interpolant by blending local radial basis function interpolants and using locally supported weight functions. The partition of unity algorithm is efficiently implemented and optimized by connecting the method with an effective cube-partition searching procedure. More precisely, we construct a cube structure, which partitions the domain and strictly depends on the size of its subdomains, so that the new searching procedure and, accordingly, the resulting algorithm enable us to efficiently deal with a large number of nodes. Complexity analysis and numerical experiments show high efficiency and accuracy of the proposed interpolation algorithm.

Key words. meshless approximation, fast algorithms, partition of unity methods, radial basis functions, scattered data

AMS subject classifications. 65D05, 65D15, 65D17

DOI. 10.1137/140989157

1. Introduction.

The problem of constructing fast algorithms for multivariate approximation of scattered data points has recently interested many researchers, who work in various areas of applied mathematics and scientific computing such as interpolation, approximation theory, neural networks, computer aided geometric design, and machine learning, to name a few. So we often need to have numerical algorithms, which allow us to efficiently deal with a large number of points, not only in one or two dimensions but also in higher dimensions, as usually occurs in several applications (see, e.g., [18, 32] and references therein).

Though there exist several numerical algorithms and alternative techniques for bivariate interpolation to scattered data, the problem of efficiently approximating many thousands or millions of three-dimensional (3D) data does not seem to be much considered in the literature, with the exception of a few cases such as in [3, 15, 17, 23, 27, 30]; a comparison of radial basis function (RBF) methods in the 3D setting can be found in [5].

Since mesh-based methods require some sort of an underlying computational mesh, i.e., any triangulation of the domain, their construction is a rather difficult task, already in two dimensions, where the mesh generation turns out usually to be one of the most time consuming parts. For this reason, in the following we focus on a *meshfree* or *meshless* approximation. More precisely, here we consider the partition of unity method, which involves the use of RBFs as local approximants and of locally supported weight functions (see [31]). Further details on the origin of the partition of unity method can be found in [2, 24]. Moreover, some other examples of local

*Submitted to the journal's Methods and Algorithms for Scientific Computing section September 29, 2014; accepted for publication (in revised form) May 21, 2015; published electronically August 4, 2015.

http://www.siam.org/journals/sisc/37-4/98915.html

†Department of Mathematics, "G. Peano", University of Torino, I–10123 Torino, Italy (roberto.cavoretto@unito.it, alessandra.derossi@unito.it). The research of the first author was partially supported by the University of Torino via grant "Approssimazione di dati sparsi e sue applicazioni." The research of the second author was partially supported by the GNCS-INdAM.

approaches involving modified Shepard's methods and different searching procedures can be found in [1, 4, 11, 22, 23, 26, 27, 30].

Starting from the previous work [10], where an efficient algorithm with a new cell-based searching procedure is presented for bivariate interpolation of large scattered data sets, in this paper we directly extend it to trivariate case, obtaining in this way a new fast algorithm for interpolation, which can briefly be summarized in three stages as follows:

(i) partition the domain into a suitable number of cubes;
(ii) consider an optimized cube-partition searching procedure establishing the minimal number of cubes to be examined, in order to localize the subset of nodes belonging to each subdomain;
(iii) apply the partition of unity method combined with local RBFs.

In particular, the algorithm is characterized by the construction of a *cube-partition searching procedure*, whose origin comes from the repeated use of a *quicksort* routine with respect to different directions, which enables us to pass from unordered to ordered data structures. Moreover, this technique is strictly related to the construction of a partition of the domain in cubes and depends on the size of its subdomains, thus producing a nearest neighbor searching procedure, which is particularly efficient in local interpolation methods. Complexity analysis and numerical experiments show efficiency and accuracy of the algorithm for cube domains. The code implemented in C/C++ language is available online at [13].

The paper is organized as follows. In section 2 we recall some theoretical results, giving a general description of the partition of unity method, which makes use of RBFs as local approximants. In section 3, we present in detail the cube-partition algorithm for trivariate interpolation, which is efficiently implemented and optimized by using a nearest neighbor searching procedure. Computational complexity and storage requirements of the interpolation algorithm are analyzed as well. In section 4, we show numerical results concerning efficiency and accuracy of the partition of unity algorithm. Finally, section 5 deals with conclusions and future work.

2. Partition of unity interpolation. Let $\mathcal{X}_n = \{\boldsymbol{x}_i, i = 1, 2, \ldots, n\}$ be a set of distinct data points or nodes, arbitrarily distributed in a domain $\Omega \subseteq \mathbb{R}^N$, $N \geq 1$, with an associated set $\mathcal{F}_n = \{f_i, i = 1, 2, \ldots, n\}$ of data values or function values, which are obtained by sampling some (unknown) function $f : \Omega \to \mathbb{R}$ at the nodes, i.e., $f_i = f(\boldsymbol{x}_i)$, $i = 1, 2, \ldots, n$.

The basic idea of the partition of unity interpolation is to start with a partition of the open and bounded domain $\Omega \subseteq \mathbb{R}^N$ into d subdomains Ω_j such that $\Omega \subseteq \bigcup_{j=1}^{d} \Omega_j$ with some mild overlap among the subdomains. Associated with these subdomains we choose a partition of unity, i.e., a family of compactly supported, nonnegative, continuous functions W_j with $\operatorname{supp}(W_j) \subseteq \Omega_j$ such that

$$\sum_{j=1}^{d} W_j(\boldsymbol{x}) = 1. \tag{2.1}$$

For each subdomain Ω_j we consider a local approximant R_j and form then the global approximant

$$\mathcal{I}(\boldsymbol{x}) = \sum_{j=1}^{d} R_j(\boldsymbol{x}) W_j(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega. \tag{2.2}$$

Here $R_j : \Omega_j \to \mathbb{R}$ defines a RBF interpolant of the form

$$R_j(\boldsymbol{x}) = \sum_{i=1}^{\bar{n}_j} c_i^{(j)} \phi(||\boldsymbol{x} - \boldsymbol{x}_i^{(j)}||_2),$$

where $\phi : [0, \infty) \to \mathbb{R}$ represents a RBF, $|| \cdot ||_2$ denotes the Euclidean norm, and $\bar{n}_j$ indicates the number of data points in Ω_j, i.e., the points $\boldsymbol{x}_i^{(j)} \in \mathcal{X}_j = \mathcal{X}_n \cap \Omega_j$. Furthermore, R_j satisfies the interpolation conditions

$$R_j(\boldsymbol{x}_i^{(j)}) = f_i^{(j)}, \quad i = 1, 2, \ldots, \bar{n}_j. \tag{2.3}$$

Note that if the local approximants satisfy the interpolation conditions (2.3), then the global approximant also interpolates at this node, i.e.,

$$\mathcal{I}(\boldsymbol{x}_i^{(j)}) = f_i^{(j)}, \quad i = 1, 2, \ldots, \bar{n}_j.$$

Solving the jth interpolation problem (2.3) leads to a system of linear equations of the form

$$\begin{bmatrix} \phi(||\boldsymbol{x}_1^{(j)} - \boldsymbol{x}_1^{(j)}||_2) & \phi(||\boldsymbol{x}_1^{(j)} - \boldsymbol{x}_2^{(j)}||_2) & \cdots & \phi(||\boldsymbol{x}_1^{(j)} - \boldsymbol{x}_{\bar{n}_j}^{(j)}||_2) \\ \phi(||\boldsymbol{x}_2^{(j)} - \boldsymbol{x}_1^{(j)}||_2) & \phi(||\boldsymbol{x}_2^{(j)} - \boldsymbol{x}_2^{(j)}||_2) & \cdots & \phi(||\boldsymbol{x}_2^{(j)} - \boldsymbol{x}_{\bar{n}_j}^{(j)}||_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi(||\boldsymbol{x}_{\bar{n}_j}^{(j)} - \boldsymbol{x}_1^{(j)}||_2) & \phi(||\boldsymbol{x}_{\bar{n}_j}^{(j)} - \boldsymbol{x}_2^{(j)}||_2) & \cdots & \phi(||\boldsymbol{x}_{\bar{n}_j}^{(j)} - \boldsymbol{x}_{\bar{n}_j}^{(j)}||_2) \end{bmatrix} \begin{bmatrix} c_1^{(j)} \\ c_2^{(j)} \\ \vdots \\ c_{\bar{n}_j}^{(j)} \end{bmatrix} = \begin{bmatrix} f_1^{(j)} \\ f_2^{(j)} \\ \vdots \\ f_{\bar{n}_j}^{(j)} \end{bmatrix},$$

or simply

$$\Phi^{(j)} \boldsymbol{c}^{(j)} = \boldsymbol{f}^{(j)}. \tag{2.4}$$

In particular, the interpolation problem is well-posed, i.e., a solution to the problem exists and is unique, if and only if the matrix $\Phi^{(j)}$ is nonsingular. A sufficient condition to have nonsingularity is that the corresponding matrix is positive definite. In fact, if the matrix $\Phi^{(j)}$ is positive definite, then all its eigenvalues are positive and therefore $\Phi^{(j)}$ is nonsingular (see, e.g., [18]).

Though the theory of RBFs is here considered, for brevity we do not report basic definitions and theorems, referring to [6, 18, 21, 32] for a more detailed analysis. Then, we give the following definition (see [31]).

DEFINITION 2.1. *Let $\Omega \subseteq \mathbb{R}^N$ be a bounded set. Let $\{\Omega_j\}_{j=1}^d$ be an open and bounded covering of Ω. This means that all Ω_j are open and bounded and that $\Omega \subseteq \bigcup_{j=1}^d \Omega_j$. Set $\delta_j = \operatorname{diam}(\Omega_j) = \sup_{\boldsymbol{x},\boldsymbol{y} \in \Omega_j} ||\boldsymbol{x} - \boldsymbol{y}||_2$. We call a family of nonnegative functions $\{W_j\}_{j=1}^d$ with $W_j \in C^k(\mathbb{R}^N)$ a k-stable partition of unity with respect to the covering $\{\Omega_j\}_{j=1}^d$ if*

(1) $\operatorname{supp}(W_j) \subseteq \Omega_j$;
(2) $\sum_{j=1}^d W_j(\boldsymbol{x}) \equiv 1$ *on* Ω;
(3) *for every $\beta \in \mathbb{N}_0^N$ with $|\beta| \leq k$ there exists a constant $C_\beta > 0$ such that*

$$||D^\beta W_j||_{L_\infty(\Omega_j)} \leq \frac{C_\beta}{\delta_j^{|\beta|}}$$

for all $1 \leq j \leq d$.

In agreement with the statements in [31], we require additional regularity assumptions on the *covering* $\{\Omega_j\}_{j=1}^d$.

DEFINITION 2.2. *Suppose that* $\Omega \subseteq \mathbb{R}^N$ *is bounded and* $\mathcal{X}_n = \{\boldsymbol{x}_i, i = 1, 2, \ldots, n\} \subseteq \Omega$ *are given. An open and bounded covering* $\{\Omega_j\}_{j=1}^d$ *is called regular for* $(\Omega, \mathcal{X}_n)$ *if the following properties are satisfied:*

(a) *for each* $\boldsymbol{x} \in \Omega$*, the number of subdomains* Ω_j *with* $\boldsymbol{x} \in \Omega_j$ *is bounded by a global constant* K*;*
(b) *each subdomain* Ω_j *satisfies an interior cone condition;*
(c) *the local fill distances* $h_{\mathcal{X}_j,\Omega_j}$*, where* $\mathcal{X}_j = \mathcal{X}_n \cap \Omega_j$*, are uniformly bounded by the global fill distance* $h_{\mathcal{X}_n,\Omega}$*, i.e.,*

$$h_{\mathcal{X}_n,\Omega} = \sup_{\boldsymbol{x}\in\Omega} \min_{\boldsymbol{x}_i\in\mathcal{X}_n} ||\boldsymbol{x} - \boldsymbol{x}_i||_2.$$

Property (a) is required to ensure that the sum in (2.2) is actually a sum over at most K summands. Since K is independent of n, unlike d, which should be proportional to n, this is essential to avoid losing convergence orders. It is crucial for an efficient evaluation of the global interpolant that only a constant number of local approximants has to be evaluated. In such a way, it should be possible to locate those K indices in constant time. Properties (b) and (c) are important for employing the estimates on RBF interpolants (see [32]).

Moreover, we are able to formulate the following theorem, which yields the polynomial precision and controls the growth of error estimates, denoting by $\pi_s^N := \pi_s(\mathbb{R}^N)$ the set of polynomials of degree at most s (see, e.g., [32]).

THEOREM 2.3. *Suppose that* $\Omega \subseteq \mathbb{R}^N$ *is compact and satisfies an interior cone condition with angle* $\theta \in (0, \pi/2)$ *and radius* $r > 0$*. Let* $s \in \mathbb{N}$ *be fixed and there exist constants* $h_0, C_1, C_2 > 0$ *depending only on* N, θ, r *such that* $h_{\mathcal{X}_n,\Omega} \leq h_0$*. Then, for all* $\mathcal{X}_n = \{\boldsymbol{x}_i, i = 1, 2, \ldots, n\} \subseteq \Omega$ *and all* $\boldsymbol{x} \in \Omega$*, there exist functions* $u_j : \Omega \to \mathbb{R}$*,* $j = 1, 2, \ldots, n$*, such that*

(1) $\sum_{j=1}^n u_j(\boldsymbol{x})p(\boldsymbol{x}_j) = p(\boldsymbol{x})$ *for all* $p \in \pi_s(\mathbb{R}^N)$*;*
(2) $\sum_{j=1}^n |u_j(\boldsymbol{x})| \leq C_1$*;*
(3) $u_j(\boldsymbol{x}) = 0$ *provided that* $||\boldsymbol{x} - \boldsymbol{x}_j||_2 > C_2 h_{\mathcal{X}_n,\Omega}$*.*

Therefore, after defining the space $C_\nu^k(\mathbb{R}^N)$ of all functions $f \in C^k$ whose derivatives of order $|\beta| = k$ satisfy $D^\beta f(\boldsymbol{x}) = \mathcal{O}(||\boldsymbol{x}||_2^\nu)$ for $||\boldsymbol{x}||_2 \to 0$, we consider the following convergence result (see, e.g., [18, 32]).

THEOREM 2.4. *Let* $\Omega \subseteq \mathbb{R}^N$ *be open and bounded and suppose that* $\mathcal{X}_n = \{\boldsymbol{x}_i, i = 1, 2, \ldots, n\} \subseteq \Omega$*. Let* $\phi \in C_\nu^k(\mathbb{R}^N)$ *be a strictly conditionally positive definite function of order* m*. Let* $\{\Omega_j\}_{j=1}^d$ *be a regular covering for* $(\Omega, \mathcal{X}_n)$ *and let* $\{W_j\}_{j=1}^d$ *be* k*-stable for* $\{\Omega_j\}_{j=1}^d$*. Then the error between* $f \in \mathcal{N}_\phi(\Omega)$*, where* $\mathcal{N}_\phi$ *is the native space of* ϕ*, and its partition of unity interpolant* (2.2) *can be bounded by*

$$|D^\beta f(\boldsymbol{x}) - D^\beta \mathcal{I}(\boldsymbol{x})| \leq C h_{\mathcal{X}_n,\Omega}^{(k+\nu)/2-|\beta|} |f|_{\mathcal{N}_\phi(\Omega)}$$

for all $\boldsymbol{x} \in \Omega$ *and all* $|\beta| \leq k/2$*.*

Comparing this convergence result with the global error estimates (see, e.g., [32]), we note that the partition of unity preserves the local approximation order for the global fit. This means that we can efficiently compute large RBF interpolants by solving small RBF interpolation problems (in parallel as well) and then glue them together with the global partition of unity $\{W_j\}_{j=1}^d$. In other words, the partition of unity approach is a simple and effective technique to decompose a large problem into many small problems while at the same time ensuring that the accuracy obtained for

the local fits is carried over to the global one. In particular, the partition of unity method can be thought as a Shepard's type interpolation with higher-order data, since local approximations R_j instead of data values f_j are used.

Finally, we remark that, among several weight functions $\bar{W}_j(\mathbf{x})$ in (2.2), a possible choice is given by Shepard's weight

$$W_j(\boldsymbol{x}) = \frac{\bar{W}_j(\boldsymbol{x})}{\sum_{k=1}^{d} \bar{W}_k(\boldsymbol{x})}, \qquad j = 1, 2, \ldots, d, \tag{2.5}$$

where $\bar{W}_j$ is the inverse of the Euclidean norm $\|\cdot - \boldsymbol{x}_j\|_2$. It constitutes a partition of unity as in (2.1).

3. Cube-partition algorithm. In this section we propose a new algorithm for trivariate interpolation of large scattered data sets lying on the domain $\Omega = [0,1]^3 \subset \mathbb{R}^3$. This algorithm, which is based on the partition of unity method for constructing a global interpolant by blending RBFs as local approximants and using locally supported weight functions, is efficiently implemented and optimized by connecting the interpolation method with an effective cube-partition searching procedure. More precisely, the considered approach is characterized by the construction of a *cube-based structure*, which partitions the domain Ω in cubes and strictly depends on the size of its subdomains. This technique is a direct extension in the 3D case of the square-partition searching procedure presented in [10] for bivariate interpolation, which we briefly recall in subsection 3.1.

Note that the paper [10] follows preceding works, where efficient searching procedures based on the partition of the domain in strips or spherical zones are considered (see [1, 7, 8, 10]).

3.1. Review of the 2D square-partition searching procedure. The construction of the 2D searching procedure described in [10] is obtained by making a partition of the bivariate domain in square cells. They are achieved generating two orthogonal families of parallel strips (see Figure 1). This approach is combinated with the repeated use of a *quicksort* routine with respect to different directions. At first, we make a sorting along the y-axis on all the points, constructing then a first family of strips parallel to the x-axis. Afterward, we order the points contained in each strip with respect to the x-axis direction, and finally we build the second family of strips parallel to the y-axis. The outcome is a square-based structure, which allows us to pass from unordered to ordered data structures. Following this idea, we can suitably split up the original data set in ordered and well-organized data subsets. More precisely, we may act as follows:

(i) organize all the data by means of a *quicksort$_y$ procedure* applied along the y-axis (the subscript denotes the sorting direction);
(ii) consider a first family of q strips, parallel to the x-axis, and order the points of each strip by using a *quicksort$_x$ procedure*;
(iii) create a second family of q strips, parallel to the y-axis, which orthogonally intersect the first strip family, thus producing a partition of the bivariate domain in square cells (see Figure 2).

Note that a specific square cell k is denoted by a double index notation in square brackets, i.e., $k = [v, w]$.

In order to obtain an efficient searching technique in the localization of points, we connect the interpolation method with the square-based partition structure, exploiting the data structure and the domain partition previously considered. This result is

obtained assuming that the square side is equal to the subdomain radius. Though this choice might seem to be trivial, in practice such an imposition means that the search of the nearby points, an essential aspect of local methods as the partition of unity method, is limited at most to nine squares: the square on which the considered point lies, and the eight neighboring squares (see Figures 1 and 2). The combination between square cell and subdomain sizes constitutes an *optimal* choice, since it allows us to search the closest points by considering only a very small number of them, i.e., taking those points belonging to one of the nine square cells and a priori ignoring all the other ones. Finally, for all those points belonging to the first and last square cells, namely, the ones located on or close to the boundary of the domain, we reduce the total number of square cells to be examined.

Remark 3.1. Assuming that the square side is equal to the subdomain radius means to limit the search of the nearest neighbor points at most to nine square cells. In fact, considering the sizes of side and radius, none of the other points (i.e., the ones which do not belong to the nine cells) can be one of the "nearest"; in other words, such choice automatically excludes all points that do not belong to the nine cells. On the contrary, if we did not impose the previous assumption, the number of square cells to be examined might become much greater and this would have a significant impact on the efficiency of our searching technique, producing a loss of time. This fact has been analyzed and checked computationally in our preliminary 2D tests, and it turns out to be valid also in the 3D case we will present in the next subsection.

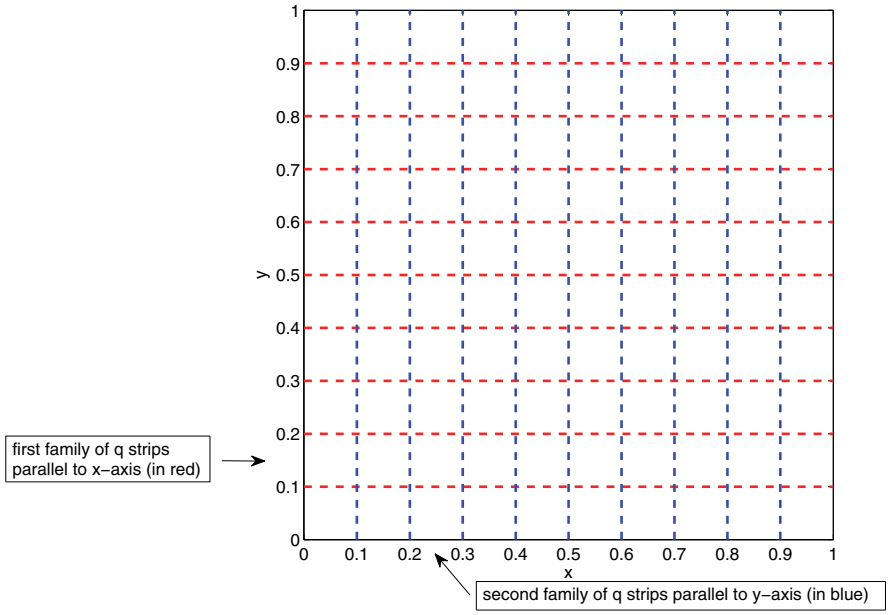

FIG. 1. *Example of orthogonal families of strips.*

3.2. The 3D cube-partition searching procedure. As in the 2D case, the basic idea in constructing the 3D searching procedure comes from the repeated use of a quicksort routine with respect to (three) different directions, i.e., along the z-axis, the y-axis, and the x-axis, enabling us to pass from unordered to ordered data

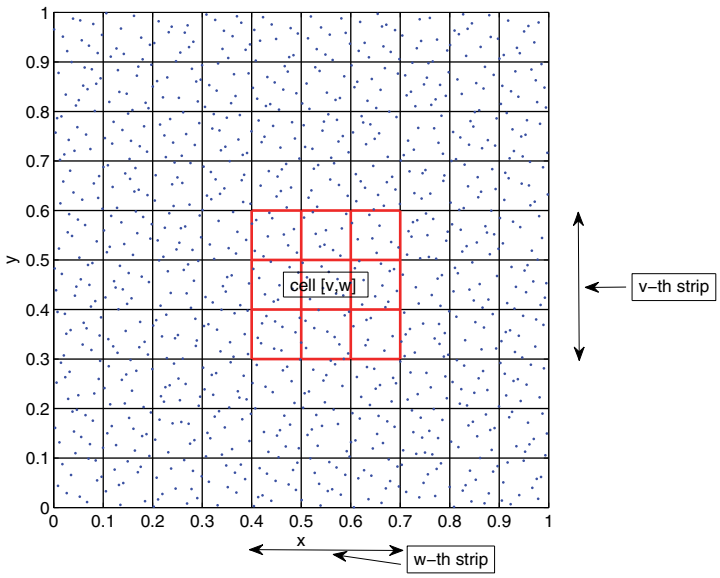

FIG. 2. *Example of square-based structure with a set of scattered data points.*

structures. This process is strictly related to the construction of a partition of the domain, here the unit cube, in smaller cubes. They are obtained generating three orthogonal families of parallelepipeds, while at the same time the original data set is suitably split up in ordered and well-organized data subsets. More precisely, in order to obtain the cube-based structure and then the resulting searching procedure, we may act as follows:

(i) organize all the data by means of a quicksort$_z$ procedure applied along the z-axis;
(ii) consider a first family of q parallelepipeds, parallel to the xy-plane, and order the points of each parallelepiped by using a quicksort$_x$ procedure;
(iii) create a second family of q parallelepipeds, parallel to the yz-plane, which orthogonally intesect the first family, and order the points of each parallelepiped by using a quicksort$_y$ procedure;
(iv) construct a third family of q parallelepipeds, parallel to the xz-plane, which orthogonally intersect the two previous families, thus producing a partition of Ω in cubes (see Figure 3).

Now, exploiting the data structure and the domain partition, we construct an efficient searching technique to be used in the localization of points, effectively connecting the partition of unity scheme with the cube-partition structure. This result is had by assuming that the cube side δ_{cube} is equal to the subdomain radius δ_{subdom}, i.e., taking $\delta_{cube} \equiv \delta_{subdom}$. From this assumption it follows that the search of the nearby points is limited at most to 27 (3^3) cubes: the cube on which the considered point lies, and the 26 neighboring cubes (see Figure 4). From now on, to locate a specific cube k, we define a triple index notation using square brackets, i.e., $k = [u, v, w]$, $u, v, w = 1, 2, \ldots, q$.

We note that the combination between cube and subdomain sizes provides an *optimal* choice, since it allows us to search the closest points only considering a very small number of them (that is, only those points belonging to one of the 27 cubes)

and a priori ignoring all the other points of Ω. Obviously, then, for all those points belonging to cubes close to the boundary of Ω, it will be required a reduction of the total number of cubes to be examined. Further details on this searching procedure are contained in subsection 3.3, where we give a detailed description of the proposed algorithm.

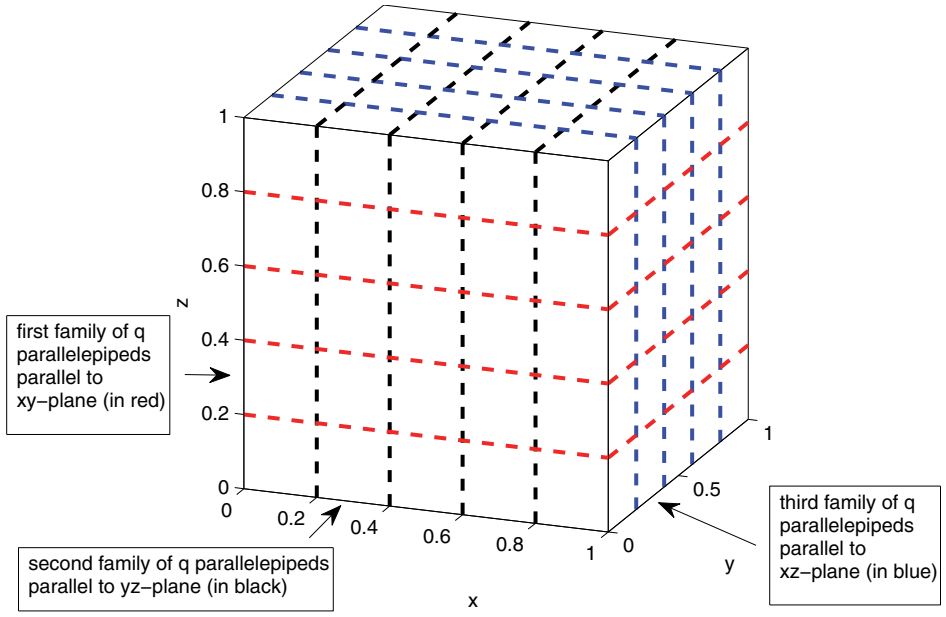

FIG. 3. *Example of orthogonal families of parallelepipeds.*

3.3. Cube algorithm. INPUT: n, number of data; $\mathcal{X}_n = \{(x_i, y_i, z_i), i = 1, 2, \ldots, n\}$, set of data points; $\mathcal{F}_n = \{f_i, i = 1, 2, \ldots, n\}$, set of data values; d, number of subdomains; $\mathcal{C}_d = \{(\bar{x}_i, \bar{y}_i, \bar{z}_i), i = 1, 2, \ldots, d\}$, set of subdomain points (centres); s, number of evaluation points; $\mathcal{E}_s = \{(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i), i = 1, 2, \ldots, s\}$, set of evaluation points.

OUTPUT: $\mathcal{A}_s = \{\mathcal{I}(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i), i = 1, 2, \ldots, s\}$, set of approximated values.

`Stage 1.` The set $\mathcal{X}_n$ of nodes and the set $\mathcal{E}_s$ of evaluation points are ordered with respect to a common direction (e.g., the z-axis), by applying a quicksort$_z$ procedure.

`Stage 2.` For each subdomain point $(\bar{x}_i, \bar{y}_i, \bar{z}_i)$, $i = 1, 2, \ldots, d$, a local spherical subdomain is constructed, whose spherical radius depends on the subdomain number d, i.e.,

$$\delta_{subdom} = \frac{\sqrt{2}}{\sqrt[3]{d}}. \tag{3.1}$$

Although other choices δ_{subdom} are possible, this value is suitably chosen, supposing to have a nearly uniform node distribution and assuming that the ratio $n/d \approx 2^3$.

`Stage 3.` A triple structure of intersecting parallelepipeds is constructed as follows:

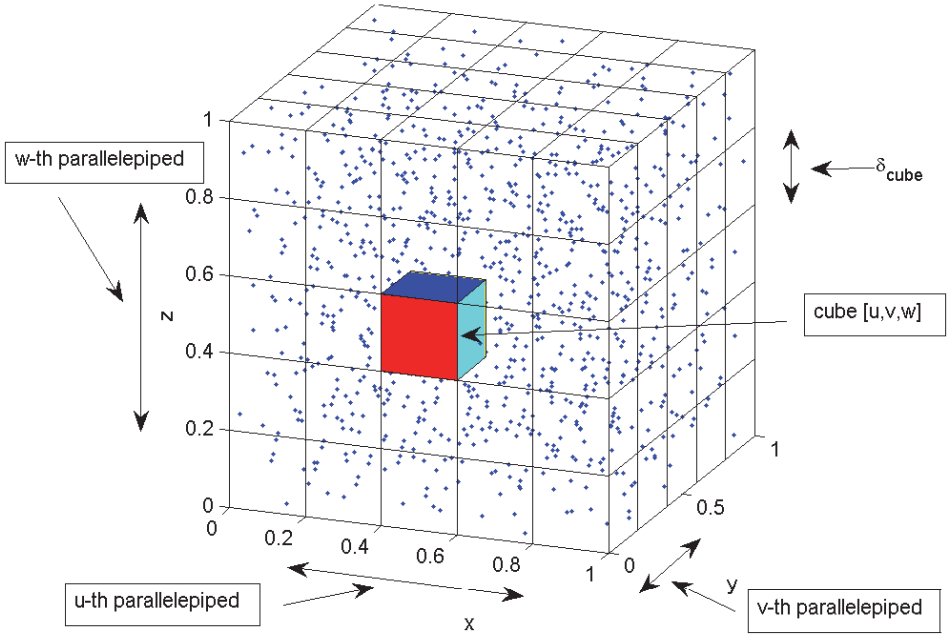

FIG. 4. *Example of cube-based structure with a set of scattered data points.*

(i) a first family of q parallelepipeds, parallel to the xy-plane, is considered taking

$$q = \left\lceil \frac{1}{\delta_{subdom}} \right\rceil, \tag{3.2}$$

and a quicksort$_x$ procedure is applied to order the nodes belonging to each parallelepiped;

(ii) a second family of q parallelepipeds, parallel to the yz-plane, is constructed and a quicksort$_y$ procedure is used to order the nodes belonging to each of the resulting parallelepipeds;

(iii) a third family of q parallelepipeds, parallel to the xz-plane, is considered.

Note that each of the three families of parallelepipeds is ordered and numbered from 1 to q; the choice in (3.2) follows directly from the side length of the domain, i.e., the unit cube, and the subdomain radius δ_{subdom}.

`Stage 4.` The unit cube is partitioned by a cube-based structure consisting of q^3 cubes, whose side length is $\delta_{cube} \equiv \delta_{subdom}$. Then, the sets $\mathcal{X}_n$, $\mathcal{C}_d$, and $\mathcal{E}_s$ are partitioned by the cube structure into q^3 subsets $\mathcal{X}_{n_k}$, $\mathcal{C}_{d_k}$, and $\mathcal{E}_{p_k}$, $k = 1, 2, \ldots, q^3$, where n_k, d_k, and p_k are the number of points in the kth cube.

This stage can be summarized in Algorithm 1.

`Stage 5.` In order to identify the cubes to be examined in the searching procedure, we adopt the following rule, which is composed of three steps:

(1) The cube side δ_{cube} is chosen equal to the subdomain radius δ_{subdom}, i.e., $\delta_{cube} \equiv \delta_{subdom}$, and the ratio between these quantities is denoted by $i^* = \delta_{subdom}/\delta_{cube}$.

(2) The value i^* provides the number j^* of cubes to be examined for each point by the rule $j^* = (2i^* + 1)^3$, which obviously here gives $j^* = 27$. In practice,

ALGORITHM 1. CUBE-PARTITION STRUCTURE.

for each cube $k = [u, v, w]$, $u, v, w = 1, 2, \ldots, q$ **do**
  partition and count the number of points
    $n_k = n_{u,v,w}$ (nodes)
    $d_k = d_{u,v,w}$ (subdomain points)
    $p_k = p_{u,v,w}$ (evaluation points);
  return $(n_k; \mathcal{X}_{n_k}) \wedge (d_k; \mathcal{C}_{d_k}) \wedge (p_k; \mathcal{E}_{p_k})$
end for

this means that the search of the nearby points is limited at most to 27 cubes: the cube on which the considered point lies, and the 26 neighboring cubes.

(3) For each cube $k = [u, v, w]$, $u, v, w = 1, 2, \ldots, q$, a cube-partition searching procedure is considered, examining the points from the cube $[u-i^*, v-i^*, w-i^*]$ to the cube $[u+i^*, v+i^*, w+i^*]$. For the points of the first and last cubes (those close to the boundary of the unit cube), we reduce the total number of cubes to be examined, setting $u - i^* = 1$ and/or $v - i^* = 1$ and/or $w - i^* = 1$ (when $u - i^* < 1$ and/or $v - i^* < 1$ and/or $w - i^* < 1$) and $u + i^* = q$ and/or $v + i^* = q$ and/or $w + i^* = q$ (when $u + i^* > q$ and/or $v + i^* > q$ and/or $w + i^* > q$).

Then, after defining which and how many cubes are to be examined, the cube-partition searching procedure (see Algorithm 2) is applied

- for each subdomain point of $\mathcal{C}_{d_k}$, $k = 1, 2, \ldots, q^3$, to determine all nodes belonging to a subdomain; the number of nodes of the subdomain centered at $(\bar{x}_j, \bar{y}_j, \bar{z}_j)$ is counted and stored in $\bar{n}_j$, $j = 1, 2, \ldots, d$;
- for each evaluation point of $\mathcal{E}_{p_k}$, $k = 1, 2, \ldots, q^3$, in order to find all those belonging to a subdomain of center $(\bar{x}_i, \bar{y}_i, \bar{z}_i)$ and radius δ_{subdom}. The number of subdomains containing the ith evaluation point is counted and stored in r_i, $i = 1, 2, \ldots, s$.

`Stage 6.` A local interpolant R_j, $j = 1, 2, \ldots, d$, is found for each subdomain, solving the linear system (2.4).

`Stage 7.` Local RBF interpolant R_j and weight function W_j, $j = 1, 2, \ldots, d$, are evaluated at each evaluation point.

`Stage 8.` Applying the global interpolant (2.2), one can find approximated values computed at any evaluation point $(\tilde{x}, \tilde{y}, \tilde{z}) \in \mathcal{E}_s$.

Remark 3.2. An important remark concerns the type of partition of unity to cover the domain Ω. Supposing a nearly uniform node distribution, in `Stage 2` we set the ratio between the node number n and the subdomain number d approximately equal to eight, i.e., $n/d \approx 2^3$. However, in general it is also possible to study different strategies of partition of unity, suitably increasing (decreasing) the number of subdomains. From a computational point of view, we would expect to increase (reduce) the number of subdomains, reducing (increasing) at the same time the subdomain size. A change of this type may influence more or less significantly the approximation results in terms of both accuracy and stability (see, e.g., [19, 29]). A further aspect to be considered is the particular shape of the subdomain Ω_j; in this case, here we use a spherical subdomain, but sometimes in applications different shapes could be more valuable and bring any benefit [28].

Remark 3.3. An interesting observation deserving to be considered concerns applicability of our algorithm in the case of nonrectangular domains. Specifically, the

ALGORITHM 2. CUBE-PARTITION SEARCHING PROCEDURE.

for $w = 1, 2, \ldots, q$ **do**
  for $v = 1, 2, \ldots, q$ **do**
    for $u = 1, 2, \ldots, q$ **do**
      set $[first_x, first_y, first_z] = [u - i^*, v - i^*, w - i^*]$
        $[last_x, last_y, last_z] = [u + i^*, v + i^*, w + i^*]$
      if $first_x < 1$ and/or $first_y < 1$ and/or $first_z < 1$ **then**
        set $first_x = 1$ and/or $first_y = 1$ and/or $first_z = 1$
      end if
      if $last_x > q$ and/or $last_y > q$ and/or $last_z > q$ **then**
        set $last_x = q$ and/or $last_y = q$ and/or $last_z = q$
      end if
      for $h = subdom\_bp_{u,v,w}, \ldots, subdom\_ep_{u,v,w}$ **do**
        set $\bar{n}_h = 0$
        for $k = first_z, \ldots, last_z$ **do**
          for $j = first_y, \ldots, last_y$ **do**
            for $i = first_x, \ldots, last_x$ **do**
              for $r = bp_{i,j,k}, \ldots, ep_{i,j,k}$ **do**
                if $(x_r, y_r, z_r) \in I_h((\bar{x}, \bar{y}, \bar{z}); \delta_{subdom})$ **then**
                  set $\bar{n}_h = \bar{n}_h + 1$
                    $STORE_{h,\bar{n}_h}(x_r, y_r, z_r, f_r)$
                end if
              end for
            end for
          end for
        end for
        return $(x, y, z) \in I_h((\bar{x}, \bar{y}, \bar{z}); \delta_{subdom})$
      end for
      for $h = eval\_bp_{u,v,w}, \ldots, eval\_ep_{u,v,w}$ **do**
        set $r_h = 0$
        for $k = first_z, \ldots, last_z$ **do**
          for $j = first_y, \ldots, last_y$ **do**
            for $i = first_x, \ldots, last_x$ **do**
              for $r = subdom\_bp_{i,j,k} \ldots, subdom\_ep_{i,j,k}$ **do**
                if $(\tilde{x}_r, \tilde{y}_r, \tilde{z}_r) \in I_h((\bar{x}, \bar{y}, \bar{z}); \delta_{subdom})$ **then**
                  set $r_h = r_h + 1$
                    $STORE_{h,r_h}(\tilde{x}_r, \tilde{y}_r, \tilde{z}_r)$
                end if
              end for
            end for
          end for
        end for
        return $(\tilde{x}, \tilde{y}, \tilde{z}) \in I_h((\bar{x}, \bar{y}, \bar{z}); \delta_{subdom})$
      end for
    end for
  end for
end for

cube-based structure and the resulting cube-partition searching procedure we present in this paper are constructed for cube domains. Nevertheless, since this algorithm is based on a meshfree method, which does not require any change, in general our searching technique could be applied with suitable adaptations also to other types of domains, like nonrectangular prisms or polyhedra. In fact, though the partitioning structure is generated for its nature on a cube, it is also true that any polyhedra can be inscribed in a cube domain. In this situation, it is possible (or very likely) that a certain number of cells is empty, so they should not be considered in the searching process of the nearest neighbor points. To do this, we need to check whether a cube cell is empty. This control consists in practice in suitably modifying some lines of code in Algorithm 2, excluding the empty cells and allowing us to find all points belonging to each subdomain, as described in this section. Another possibility to deal with a problem of this type is to construct cube cells of variable sizes, suitably increasing their sides when such cells are devoid of points. In this way, we could think to consider an adaptive approach, which enables our searching procedure to effectively work. However, as this topic turns out to be particularly important and not trivial (see, e.g., the recent papers [14, 28]), it will be dealt with in detail in forthcoming research.

3.4. Complexity analysis. The algorithm is based on the construction of a cube-partition searching procedure. It enables us to efficiently determine all points belonging to each subdomain Ω_j, $j = 1, 2, \ldots, d$, so that we can compute local RBF interpolants to be used in the partition of unity scheme. Assuming that the covering $\{\Omega_j\}_{j=1}^d$ is regular and local and the set $\mathcal{X}_n$ of data points is quasi-uniform, we analyze the complexity of this code.

The cube-partition algorithm involves the use of the standard quicksort routine, which requires on average a time complexity $\mathcal{O}(M \log M)$, where M is the number of points to be sorted. Specifically, we have a distribution phase consisting of building the data structure, in which the computational cost has order: $\mathcal{O}(n \log n)$ for the sorting of all n nodes and $\mathcal{O}(s \log s)$ for the sorting of all s evaluation points in `Stage 1`. Then, in `Stage 3` the quicksort routine is repeatedly used with respect to different directions considering a reduced number of points. Since the number of centers in each subdomain Ω_j is bounded by a constant (see Definition 2.2), we need $\mathcal{O}(1)$ space and time for each subdomain to solve the local RBF interpolation problems. In fact, in order to obtain the local RBF interpolants, we have to solve d linear systems of (relatively) small sizes, i.e., $\bar{n}_j \times \bar{n}_j$, with $\bar{n}_j << n$, thus requiring a constant running time $\mathcal{O}(\bar{n}_j^3)$, $j = 1, 2, \ldots, d$, for each subdomain (see `Stage 6`). Then, in `Stages 5, 7,` and `8` we also need a cost of $r_k \cdot \mathcal{O}(\bar{n}_j)$, $j = 1, 2, \ldots, d$, $k = 1, 2, \ldots, s$, for the kth evaluation point of $\mathcal{E}_s$; in other words, we have a constant time to get the value of the global fit (2.2). Finally, the algorithm requires $4n$, $4d$, and $4s$ storage requirements for the data, and $\bar{n}_j$, $j = 1, 2, \ldots, d$, locations for the coefficients of each local RBF interpolant.

In conclusion, we point out that in this paper we actually propose a new space-partitioning data structure based on the construction of the cube structure and the corresponding searching procedure (see subsections 3.2–3.3). This technique is studied to reduce the computational cost compared to the most advanced data structures like kd-trees [32]; in fact, while time complexity of the optimal kd-tree is $\mathcal{O}(3M \log M)$, complexity of the cube-based structure is $\mathcal{O}(2M \log M)$. Moreover, the generation of such a cube-based structure allows us to run the searching procedure in O(1), whereas the same process for kd-trees can be performed in $\mathcal{O}(\log M)$ time.

4. Numerical experiments. In this section we present a few numerical tests to show performance of the cube-partition algorithm, numerically analyzing efficiency and accuracy of the local interpolation scheme on some sets of scattered data. The code available online at [13] is implemented in C/C++ language, while numerical results are carried out on an Intel Core i7-4500U 1.8-GHz processor. In the experiments we consider three node distributions containing $n = (2^k+1)^3$, $k = 4, 5, 6$: (i) uniformly random Halton nodes generated by using the MATLAB program `haltonseq.m` (see [18]); (ii) pseudorandom nodes generated by using the `rand` MATLAB command;[1] and (iii) grid nodes. The cube-partition algorithm is run considering $d = 8^{k-1}$, $k = 4, 5, 6$, subdomain points and $s = 11^3 = 1331$ grid evaluation points, which are contained in the unit cube $\Omega = [0, 1]^3$. Here, for the global interpolant (2.2) we use Shepard's weight (2.5).

The performance of the interpolation algorithm is verified taking the data values by the following two trivariate Franke's test functions (see, e.g., [23, 26])

$$\begin{aligned} f_1(x,y,z) = {} & \frac{3}{4}\exp\left[-\frac{(9x-2)^2+(9y-2)^2+(9z-2)^2}{4}\right] \\ & + \frac{3}{4}\exp\left[-\frac{(9x+1)^2}{49}-\frac{9y+1}{10}-\frac{9z+1}{10}\right] \\ & + \frac{1}{2}\exp\left[-\frac{(9x-7)^2+(9y-3)^2+(9z-5)^2}{4}\right] \\ & - \frac{1}{5}\exp\left[-(9x-4)^2-(9y-7)^2-(9z-5)^2\right], \end{aligned}$$

$$f_2(x,y,z) = \frac{(1.25+\cos(5.4y))\cos(6z)}{6+6\,(3x-1)^2},$$

and using Gaussian C^∞ (G), Matérn C^4 (M4), and Wendland C^4 (W4) as local RBF interpolants,

$$\text{(G)} \qquad \phi(r) = \mathrm{e}^{-\alpha^2 r^2},$$

$$\text{(M4)} \qquad \phi(r) = \mathrm{e}^{-\epsilon r}(\epsilon^2 r^2 + 3\epsilon r + 3),$$

$$\text{(W4)} \qquad \phi(r) = (1-\delta r)_+^6\,(35\delta^2 r^2 + 18\delta r + 3),$$

where $\alpha, \epsilon, \delta \in \mathbb{R}^+$ are the *shape parameters*, $r = ||\cdot||_2$ is the Euclidean distance, and $(\cdot)_+$ denotes the truncated power function. Note that Gaussian C^∞ and Matérn C^4 are globally supported basis functions, whereas Wendland C^4 is a compactly supported one (see [32]).

Some information about the execution of the interpolation algorithm described in section 3 is reported in Table 1, namely, the number q^3 of partitions in cubes of the domain and the CPU times (in seconds) computed on Halton points and obtained by running the cube-partition algorithm. Moreover, since we are interested in pointing out the effectiveness of the proposed algorithm, in Table 1 we also show CPU times obtained by using the same interpolation method presented in section 2, but

[1] To permit the repetition of tests, we used the MATLAB command `rng('default')` before generating such points with `rand`.

without partitioning the domain Ω in cubes and, accordingly, bereft of the corresponding searching procedure; in other words, when the cube-partitioning structure is not considered, the interpolation scheme is simply applied by making a complete search within the domain to find the nearest neighbor points in each subdomain. This analysis emphasizes that the use of a cube structure gives a considerable saving of time, mainly when the number of points to be handled becomes quite large. In particular, as confirmed from Table 1, we remark that the benefit of using the cube-partition searching procedure is more and more significant as the number of interpolation nodes becomes larger and larger.

TABLE 1

Number of partitions in cubes and CPU times (in seconds) computed on Halton points and obtained by running the cube-partition algorithm (t_{cube}), and the corresponding one without a cube structure ($t_{no-cube}$).

n	d	q^3	t_{cube}	$t_{no-cube}$
4913	512	6^3	**1.1**	1.4
35937	4096	12^3	**7.9**	15.5
274625	32768	23^3	**62.7**	525.0

Analyzing the performance of the algorithm, we observe that the cube-partition searching procedure turns out to be powerful and efficient, because CPU times reported in Table 1 are mainly due to solution of d linear systems having matrices with a relatively large number of entries, usually more than a hundred.

Now, in order to investigate accuracy of the method, we compute the root mean square error (RMSE), whose formula is

$$RMSE = \sqrt{\frac{1}{s}\sum_{i=1}^{s} |f(\boldsymbol{x}_i) - \mathcal{I}(\boldsymbol{x}_i)|^2},$$

analyzing its behavior on Halton points by varying the values of the shape parameters for Gaussian, Matérn, and Wendland functions (see Figure 5). These graphs allow us to find the optimal values of α, ϵ, and δ, i.e., those values for which we obtain the smallest RMSEs (see Tables 2 and 3). Note that each evaluation is carried out by choosing equispaced values of the shape parameters, taking $\alpha, \epsilon \in [1, 10]$ and $\delta \in [0.1, 1.9]$. Analyzing error tables and graphs, we can see that Matérn and Wendland functions have a greater stability than RBF Gaussian, but the latter gives us greater accuracy, although its interpolation matrices might be subject to ill-conditioning problems for small values of α. This behavior is what we expect from a theoretical standpoint, but here it is validated by numerical tests. Moreover, we remark that several numerical experiments (not reported here for brevity) have been carried out using other test functions and the results show uniform behavior.

Finally, to show that the CPU times in Table 1 essentially depend on the size of interpolation matrices, we repeat numerical tests fixing a maximum number (i.e., $m_i = m_{max}$, $i = 1, 2, \ldots, d$) of nodes for each subdomain, namely, only considering the m_{max} nodes closest to the subdomain centres. In fact, for example, taking $m_{max} = 50, 70$ (and also m_{max} not fixed) and denoting by $t_{cube}^{m_{max}}$ the corresponding execution times, we get a significant reduction of times, since $t_{cube}^{50} = 0.5$ and $t_{cube}^{70} = 0.6$ for $n = 4913$, $t_{cube}^{50} = 1.9$ and $t_{cube}^{70} = 3.4$ for $n = 35937$, while $t_{cube}^{50} = 14.2$ and $t_{cube}^{70} = 28.1$ for $n = 274625$ (see Table 1 for a comparison). Nevertheless, this reduction expressed

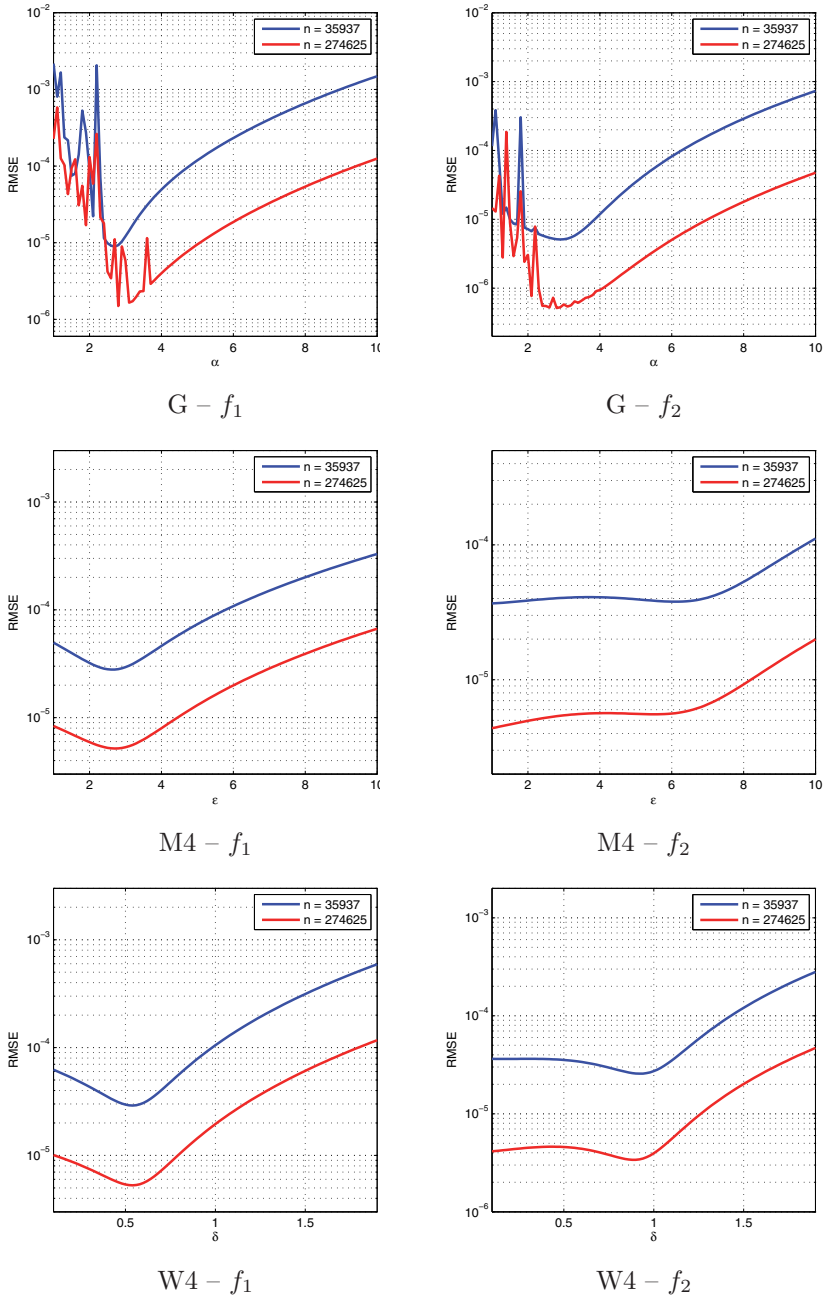

FIG. 5. *RMSEs computed on Halton points and obtained by varying the shape parameters.*

in terms of CPU times is paid, in general, only with a slight loss of accuracy, since the behavior of RMSEs is similar to that shown in Figure 5.

Similarly, we repeat the extensive experiments carried out on Halton points, using MATLAB pseudorandom points. The results obtained essentially show similar behavior in terms of both computational performance and conditioning. For this reason,

TABLE 2
RMSEs computed on Halton points and obtained by using optimal values of α, ϵ, and δ for f_1.

	G		M4		W4	
n	RMSE	α_{opt}	RMSE	ϵ_{opt}	RMSE	δ_{opt}
35937	8.8797E − 6	2.7	2.7905E − 5	2.6	2.9041E − 5	0.54
274625	1.4928E − 6	2.8	5.1734E − 6	2.7	5.2847E − 6	0.54

TABLE 3
RMSEs computed on Halton points and obtained by using optimal values of α, ϵ, and δ for f_2.

	G		M4		W4	
n	RMSE	α_{opt}	RMSE	ϵ_{opt}	RMSE	δ_{opt}
35937	5.1013E − 6	2.9	3.6761E − 5	1.0	2.5677E − 5	0.92
274625	5.1446E − 7	2.8	4.3760E − 6	1.0	3.3941E − 6	0.88

TABLE 4
RMSEs computed on MATLAB *pseudorandom points and obtained by using optimal values of α, ϵ, and δ for f_1.*

	G		M4		W4	
n	RMSE	α_{opt}	RMSE	ϵ_{opt}	RMSE	δ_{opt}
35937	1.5678E − 5	2.7	3.2917E − 5	2.8	3.3926E − 5	0.56
274625	1.8669E − 6	3.0	6.5270E − 6	2.8	6.7059E − 6	0.54

TABLE 5
RMSEs computed on MATLAB *pseudorandom points and obtained by using optimal values of α, ϵ, and δ for f_2.*

	G		M4		W4	
n	RMSE	α_{opt}	RMSE	ϵ_{opt}	RMSE	δ_{opt}
35937	5.4690E − 6	2.2	3.3778E − 5	1.0	2.7932E − 5	0.86
274625	5.9387E − 7	2.8	5.5566E − 6	1.0	4.5822E − 6	0.90

we do not report further tables and graphs about CPU times and errors, respectively, restricting ourselves here to exhibit Tables 4 and 5. In such cases, we observe a slight loss of accuracy due to less uniformity of pseudorandom points compared to the Halton nodes. A degradation of this type has also been noted on the conditioning, but in general it turns out to be absolutely minimal.

In conclusion, in Table 6 we also report the RMSEs obtained by applying the cube-partition algorithm on sets of grid points. In this situation, the interpolation nodes are uniform on the domain Ω; this fact leads to more accurate results and, generally, to less ill-conditioned RBF matrices.

TABLE 6
RMSEs computed on grid points and obtained by using optimal values of α, ϵ, and δ.

n	35937		274625	
	f_1	f_2	f_1	f_2
G	2.4327E − 6	1.5521E − 7	2.6580E − 7	1.3038E − 8
α_{opt}	3.4	3.1	4.4	2.7
M4	1.3052E − 5	3.0937E − 6	2.9642E − 6	6.2521E − 7
ϵ_{opt}	2.0	1.5	4.3	2.2
W4	1.2938E − 5	2.8769E − 6	2.6711E − 6	5.9111E − 7
δ_{opt}	0.48	0.50	0.86	0.46

5. Conclusions and future work. In this paper we propose a new local interpolation algorithm for trivariate interpolation of scattered data points. It is based on the construction of a partition of the domain in cubes, enabling us to optimally implement a cube-partition searching procedure in order to efficiently detect the nodes belonging to each subdomain of the partition of unity method. This technique works well and quickly also when the amount of data to be interpolated is very large. Moreover, the proposed algorithm is flexible, since different choices of local interpolants are allowable, and completely automatic.

As regards research and future work, first of all we are interested in extending the proposed algorithm to higher dimensions. Then, even though the choice of low-order basis functions such as Matérn and Wendland functions gives a good trade-off between stability and accuracy, we are still considering the need dealing with the ill-conditioning problem of high-order basis functions. On the one hand, we might consider suitable preconditioning techniques for RBF interpolation matrices as already done in [9] for RBF collocation matrices; on the other hand, one could study alternative stategies to have a stable evaluation of interpolants via the Hilbert–Schmidt SVD as in [12, 20] or new stable bases as in [16, 25].

Acknowledgments. The authors are very grateful to the editor and the anonymous referees for their detailed and valuable comments, which helped to greatly improve the paper.

REFERENCES

[1] G. ALLASIA, R. BESENGHI, R. CAVORETTO, AND A. DE ROSSI, *Scattered and track data interpolation using an efficient strip searching procedure*, Appl. Math. Comput., 217 (2011), pp. 5949–5966.

[2] I. BABUŠKA AND J. M. MELENK, *The partition of unity method*, Internat. J. Numer. Methods. Engrg., 40 (1997), pp. 727–758.

[3] R. K. BEATSON, W. A. LIGHT, AND S. BILLINGS, *Fast solution of the radial basis function interpolation equations: Domain decomposition methods*, SIAM J. Sci. Comput., 22 (2000), pp. 1717–1740.

[4] M. W. BERRY AND K. S. MINSER, *Algorithm* 798: *High-dimensional interpolation using the modified Shepard method*, ACM Trans. Math. Software, 25 (1999), pp. 353–366.

[5] M. BOZZINI AND M. ROSSINI, *Testing methods for* 3*D scattered data interpolation*, Monogr. Real Acad. Ci. Exact. Fis.-Quim. Nat. Zaragoza, 20 (2002), pp. 111–135.

[6] M. D. BUHMANN, *Radial Basis Functions: Theory and Implementation*, Cambridge Monogr. Appl. Comput. Math. 12, Cambridge University Press, Cambridge, UK, 2003.

[7] R. CAVORETTO AND A. DE ROSSI, *Fast and accurate interpolation of large scattered data sets on the sphere*, J. Comput. Appl. Math., 234 (2010), pp. 1505–1521.

[8] R. Cavoretto and A. De Rossi, *Spherical interpolation using the partition of unity method: An efficient and flexible algorithm*, Appl. Math. Lett., 25 (2012), pp. 1251–1256.
[9] R. Cavoretto, A. De Rossi, M. Donatelli, and S. Serra-Capizzano, *Spectral analysis and preconditioning techniques for radial basis function collocation matrices*, Numer. Linear Algebra Appl., 19 (2012), pp. 31–52.
[10] R. Cavoretto and A. De Rossi, *A meshless interpolation algorithm using a cell-based searching procedure*, Comput. Math. Appl., 67 (2014), pp. 1024–1038.
[11] R. Cavoretto, *A numerical algorithm for multidimensional modeling of scattered data points*, Comput. Appl. Math., 34 (2015), pp. 65–80.
[12] R. Cavoretto, G. E. Fasshauer, and M. McCourt, *An introduction to the Hilbert-Schmidt SVD using iterated Brownian bridge kernels*, Numer. Algorithms, 68 (2015), pp. 393–422.
[13] R. Cavoretto and A. De Rossi, *A trivariate interpolation algorithm using a cube-partition searching procedure*, http://hdl.handle.net/2318/152999.
[14] Y. Chen, S. Gottlied, A. Heryudono, and A. Narayan, *A reduced radial basis function method for partial differential equations on irregular domains*, J. Sci. Comput., to appear.
[15] J. Cherrie, R. Beatson, and G. Newsam, *Fast evaluation of radial basis functions: Methods for generalized multiquadrics in* $\mathbb{R}^n$, SIAM J. Sci. Comput., 23 (2002), pp. 1549–1571.
[16] S. De Marchi and G. Santin, *A new stable basis for radial basis function interpolation*, J. Comput. Appl. Math., 253 (2013), pp. 1–13.
[17] S. Deparis, D. Forti, and A. Quarteroni, *A rescaled localized radial basis function interpolation on non-Cartesian and nonconforming grids*, SIAM J. Sci. Comput., 36 (2014), pp. A2745–A2762.
[18] G. E. Fasshauer, *Meshfree Approximation Methods with* Matlab, World Scientific Publishers, River Edge, NJ, 2007.
[19] G. E. Fasshauer, *Positive definite kernels: Past, present and future*, Dolomites Res. Notes Approx., 4 (2011), pp. 21–63.
[20] G. E. Fasshauer and M. J. McCourt, *Stable evaluation of Gaussian radial basis function interpolants*, SIAM J. Sci. Comput., 34 (2012), pp. A737–A762.
[21] A. Iske, *Scattered data approximation by positive definite kernel functions*, Rend. Sem. Mat. Univ. Pol. Torino, 69 (2011), pp. 217–246.
[22] M. A. Iyer, L. T. Watson, and M. W. Berry, *SHEPPACK: A Fortran* 95 *package for interpolation using the modified Shepard algorithm*, in Proceedings of the Annual Southeast Conference, R. Menezes et al., eds., ACM, New York, 2006, pp. 476–481.
[23] D. Lazzaro and L. B. Montefusco, *Radial basis functions for the multivariate interpolation of large scattered data sets*, J. Comput. Appl. Math., 140 (2002), pp. 521–536.
[24] J. M. Melenk and I. Babuška, *The partition of unity finite element method: Basic theory and applications*, Comput. Methods. Appl. Mech. Engrg., 139 (1996), pp. 289–314.
[25] M. Pazouki and R. Schaback, *Bases for kernel-based spaces*, J. Comput. Appl. Math., 236 (2011), pp. 575–588.
[26] R. J. Renka, *Multivariate interpolation of large sets of scattered data*, ACM Trans. Math. Software, 14 (1988), pp. 139–148.
[27] R. J. Renka, *Algorithm* 661: *QSHEP*3*D: Quadratic Shepard method for trivariate interpolation of scattered data*, ACM Trans. Math. Software, 14 (1988), pp. 151–152.
[28] A. Safdari-Vaighani, A. Heryudono, and E. Larsson, *A radial basis function partition of unity collocation method for convection-diffusion equations arising in financial applications*, J. Sci. Comput., 64 (2015), pp. 341–367.
[29] R. Schaback, *Error estimates and condition numbers for radial basis function interpolation*, Adv. Comput. Math., 3 (1995), pp. 251–264.
[30] W. I. Thacker, J. Zhang, L. T. Watson, J. B. Birch, M. A. Iyer, and M. W. Berry, *Algorithm* 905: *SHEPPACK: Modified Shepard algorithm for interpolation of scattered multivariate data*, ACM Trans. Math. Software, 37 (2010), pp. 1–20.
[31] H. Wendland, *Fast evaluation of radial basis functions: Methods based on partition of unity*, in Approximation Theory X: Wavelets, Splines, and Applications, C. K. Chui, L. L. Schumaker, and J. Stöckler, eds., Vanderbilt University Press, Nashville, TN, 2002, pp. 473–483.
[32] H. Wendland, *Scattered Data Approximation*, Cambridge Monogr. Appl. Comput. Math. 17, Cambridge University Press, Cambridge, UK, 2005.