An Efficient Trivariate Algorithm for Tetrahedral Shepard Interpolation

An Efficient Trivariate Algorithm for Tetrahedral Shepard Interpolation

R. Cavoretto · A. De Rossi · F. Dell'Accio · F. Di Tommaso

Abstract In this paper we present a trivariate algorithm for fast computation of tetrahedral Shepard interpolants. Though the tetrahedral Shepard method achieves an approximation order better than classical Shepard formulas, it requires to detect suitable configurations of tetrahedra whose vertices are given by the set of data points. In doing that, we propose the use of a fast searching procedure based on the partitioning of domain and nodes in cubic blocks. This allows us to find the nearest neighbor points associated with each ball that need to be used in the 3D interpolation scheme. Numerical experiments show good performance of our interpolation algorithm.

Keywords scattered data interpolation · tetrahedral Shepard operator · fast algorithms · approximation algorithms

Mathematics Subject Classification (2000) 65D05 · 65D15 · 41A05

1 Introduction

Suppose that discrete values of a function f are given at certain data points or nodes $X_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ in a compact convex domain $\Omega \subset \mathbb{R}^2$. If the data points are scattered, that is they have not any structure, the triangular Shepard method [14] can be applied efficiently [6] to approximate and interpolate the target function $f : \Omega \to \mathbb{R}$. This scheme has been introduced by Little in 1983 in light of some

R. Cavoretto · A. De Rossi
Department of Mathematics "Giuseppe Peano", University of Torino, via Carlo Alberto 10, 10123 Torino, Italy
Tel.: +39 0116702830 and +39 0116702837
Fax: +39 0116702878
E-mail: roberto.cavoretto@unito.it, alessandra.derossi@unito.it

F. Dell'Accio · F. Di Tommaso
Department of Mathematics and Computer Science, University of Calabria, via P. Bucci, Cubo 30A, 87036 Rende (CS), Italy
Tel.: +39 0984 496490 and +39 0984 496493
Fax: +39 0984 496484
E-mail: francesco.dellaccio@unical.it, ditommaso@mat.unical.it

drawbacks of the more known Shepard method (see [16] for the original paper or [8] to get acquainted with some strategies to overcome them) and uses the same idea of combining in a convex way some values obtainable from the data $f(\boldsymbol{x}_i)$.

More precisely, the Shepard method is a simple and fast scheme for interpolating scattered data in $\mathbb{R}^d$ that combines weighted inverse distance functions with function values $f(\boldsymbol{x}_i)$, but reproduces only constant polynomials and has flat spots or cusps in the neighbourhood of all data points. The triangular Shepard method is a scheme for interpolating scattered data in $\mathbb{R}^2$ that surpasses the Shepard's method greatly in esthetic behavior and provides an interpolant with linear precision without derivative data [14]. However it is more complex than the Shepard method and presupposes an appropriate triangulation of the node set. In fact, it combines basis functions which are the normalization of the product of the inverse distances from the vertices of each triangle with linear combinations of the values $f(\boldsymbol{x}_i)$ at the vertices of those triangles. Each linear combination is chosen in order to interpolate these data and therefore it has a simple expression in terms of barycentric coordinates and is numerically stable within the triangle, where the influence of the basis element is greater. The requirement of a triangulation of the node set $X_n \subset \mathbb{R}^2$, e.g. a Delaunay triangulation, is a disadvantage with respect to the original Shepard method [16], but recent researches have demonstrated that the triangulation can be organized in an efficient way by reducing by 1/3 the number of triangles [10]. We are talking about the so called *compact triangulation* which allows the triangles to overlap or be disjoint. These triangulations are determined by minimizing the bound of the error of the linear interpolant on the vertices of the triangle, chosen in a set of nearby nodes. For such kind of triangulations the block-based partitioning structure procedure introduced in [7] can be easily applied to make the method very fast [6].

The need of scattered data interpolation methods in the multivariate framework and, in particular, in the trivariate case, motivates the generalization of the triangular Shepard method to the tetrahedral one. In analogy with the 2D case, the 3D operator is a linear combination of basis functions based on the vertices of tetrahedra with Lagrange linear interpolants on these vertices as coefficients. In order to well define this operator, a set of tetrahedra whose vertices cover $X_n \subset \mathbb{R}^3$ is needed. This can be determined by standard routines for constructing a 3D Delaunay triangulation, which could be very expensive in general situations, since it could require a computation cost of $O(n^2)$ [12]. However, by adapting the strategy used in [10], we can produce 3D compact triangulations and reduce until to $O(n \log n)$ the computational cost in case of general configurations of nodes.

A very important point in the implementation of local high-dimensional interpolants, as the tetrahedral Shepard method, is the construction of fast partitioning and searching routines, which allow us to efficiently organize the set of data points. This request derives from the need of detecting suitable configurations of tetrahedra whose vertices are given by the interpolation nodes. It is therefore important for each point to find the nearest neighbors so as to efficiently compute the tetrahedral Shepard interpolant. In the current literature, the most advanced techniques suited for this purpose are known as kd-trees (see [2, 11]). In this paper, we propose the use of a fast searching procedure based on the partitioning of the domain and of its interpolation points in cubic blocks. Acting in this way, by recursively calling a sorting routine, we can partition and arrange all the nodes in the different cube-shaped blocks. Then, once the data points are stored in such blocks, an optimized

searching technique is applied to detect the nearest neighbor points, thus enabling us to carry out a suitable choice of tetrahedra for 3D interpolation. Similar techniques were also studied in [7,4,5] in the context of partition of unity methods combined with the use of local radial basis functions, and suitably adapted to 2D interpolation via triangular Shepard interpolants [6]. Note that in this work we present the tetrahedral Shepard method and the related theoretical results for a generic domain $\Omega \subset \mathbb{R}^3$. However, for our purposes and for the sake of clarity, we restrict our attention to the unit cube domain $\Omega = [0,1]^3 \subset \mathbb{R}^3$ when we describe the algorithms and discuss our numerical results.

The paper is organized as follows. In Section 2 we present the tetrahedral Shepard method for trivariate interpolation, giving a bound for the error and analyzing the convergence. Section 3 is devoted to describe our efficient procedures used to identify and search the nearest neighbor points in the 3D interpolation scheme. Besides that, a complexity analysis is also provided. In Section 4 we give a pseudo-code of the complete interpolation algorithm. In Section 5 we propose a series of numerical experiments to show the performance of our tetrahedral Shepard algorithm. Finally, Section 6 deals with conclusions and future work.

2 Trivariate Interpolation on Tetrahedra

2.1 Tetrahedral Shepard Method

Let $X_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be a set of data points or nodes of $\mathbb{R}^3$ with an associated set of function data $F_n = \{f_1, \ldots, f_n\}$ and $T = \{t_1, \ldots, t_m\}$ be a set of tetrahedra with vertices in X_n. Let us denote by $V_j = \{\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4}\}$ the set of vertices of t_j, $j = 1, \ldots, m$ and let us assume that the set $\{V_j\}_{j=1,\ldots,m}$ constitutes a cover of X_n, that is

$$\bigcup_{j=1}^{m} V_j = X_n.$$

To each tetrahedra t_j we can associate the set of barycentric coordinates of a point $\boldsymbol{x} \in \mathbb{R}^3$, that is

$$\lambda_{j,j_1}(\boldsymbol{x}) = \frac{V(\boldsymbol{x}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4})}{V(\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4})}, \quad \lambda_{j,j_2}(\boldsymbol{x}) = \frac{V(\boldsymbol{x}_{j_1}, \boldsymbol{x}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4})}{V(\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4})},$$

$$\lambda_{j,j_3}(\boldsymbol{x}) = \frac{V(\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}, \boldsymbol{x}_{j_4})}{V(\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4})}, \quad \lambda_{j,j_4}(\boldsymbol{x}) = \frac{V(\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x})}{V(\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4})},$$

where $V(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{v}, \boldsymbol{z})$ is 6 times the signed volume of the tetrahedra of vertices $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{v}, \boldsymbol{z}$. The linear polynomial $L_j(\boldsymbol{x})$, which interpolates the data at the vertices of the tetrahedra t_j, has a very simple expression in terms of barycentric coordinates

$$L_j(\boldsymbol{x}) = \sum_{k=1}^{4} \lambda_{j,j_k}(\boldsymbol{x}) f_{j_k}, \quad j = 1, \ldots, m. \tag{1}$$

The tetrahedral basis functions are a normalization of the product of the inverse distances from the vertices of the tetrahedra t_j

$$B_{\mu,j}(\boldsymbol{x}) = \frac{\prod_{\ell=1}^{4} \|\boldsymbol{x} - \boldsymbol{x}_{j_\ell}\|_2^{-\mu}}{\sum_{k=1}^{m} \prod_{\ell=1}^{4} \|\boldsymbol{x} - \boldsymbol{x}_{k_\ell}\|_2^{-\mu}}, \qquad j = 1, \ldots, m, \quad \mu > 0. \tag{2}$$

and the tetrahedral Shepard method is defined by

$$\mathcal{T}_\mu[f](\boldsymbol{x}) = \sum_{j=1}^{m} B_{\mu,j}(\boldsymbol{x})\, L_j(\boldsymbol{x}). \tag{3}$$

Here $\|\cdot\|_2$ denotes the Euclidean distance in $\mathbb{R}^3$. Similarly to the triangular Shepard basis functions, the tetrahedral ones form a partition of unity and allow the interpolation of functional and derivative values. Below we formalize these statements without proofs which can be easily obtained in analogy with [10, Proposition 2.1].

Proposition 1 *The tetrahedral basis function $B_{\mu,j}(\boldsymbol{x})$ and its gradient (that exists for $\mu > 1$) vanish at all nodes $\boldsymbol{x}_i \in X_n$ that are not a vertex of the corresponding tetrahedron t_j. That is,*

$$B_{\mu,j}(\boldsymbol{x}_i) = 0, \tag{4}$$
$$\nabla B_{\mu,j}(\boldsymbol{x}_i) = 0, \quad \mu > 1, \tag{5}$$

for any $j = 1, \ldots, m$ and $i \notin \{j_1, j_2, j_3, j_4\}$. Moreover, they form a partition of unity, that is

$$\sum_{j=1}^{m} B_{\mu,j}(\boldsymbol{x}) = 1 \tag{6}$$

and consequently, for each $i = 1, \ldots, n$,

$$\sum_{j \in J_i} B_{\mu,j}(\boldsymbol{x}_i) = 1, \tag{7}$$
$$\sum_{j \in J_i} \nabla B_{\mu,j}(\boldsymbol{x}_i) = 0, \quad \mu > 1, \tag{8}$$

where $J_i = \big\{k \in \{1, \ldots, m\} : i \in \{k_1, k_2, k_3, k_4\}\big\}$ is the set of tetrahedra which have $\boldsymbol{x}_i$ as a vertex.

Above properties imply that the operator $\mathcal{T}_\mu$ satisfies the following ones.

Proposition 2 *The operator $\mathcal{T}_\mu$ is an interpolation operator, that is,*

$$\mathcal{T}_\mu[f](\boldsymbol{x}_i) = f_i, \qquad i = 1, \ldots, n,$$

and reproduces polynomials of degree less than or equal to 1.

Proof If $\boldsymbol{x}_i$ is a vertex of tetrahedron t_j (i.e., $i \in \{j_1, j_2, j_3, j_4\}$), then $L_j(\boldsymbol{x}_i) = f_i$ by (1), otherwise $B_{\mu,j}(\boldsymbol{x}_i) = 0$ by Proposition 1. Using (7) we then have

$$\mathcal{T}_\mu[f](\boldsymbol{x}_i) = \sum_{j=1}^{m} B_{\mu,j}(\boldsymbol{x}_i) L_j(\boldsymbol{x}_i) = \sum_{j \in J_i} B_{\mu,j}(\boldsymbol{x}_i) f_i = f_i.$$

The basis functions $B_{\mu,j}$ are a partition of unity and the polynomials $L_j(\boldsymbol{x})$, for $j = 1, \ldots, m$, reproduce linear polynomials. As a consequence the operator $\mathcal{T}_\mu$ reproduces linear polynomials.

□

2.2 Bound for the Error and Analysis of Convergence

The procedure to select the compact 3D-triangulation (by tetrahedra) of the node set X_n strongly affects the results of the analysis of the convergence of the operator $\mathcal{T}_\mu[f](\boldsymbol{x})$ which is the object of this section.

In order to determine the approximation order of the tetrahedral operator, we denote by $\Omega \subset \mathbb{R}^3$ a compact convex domain containing X_n and by $C^{1,1}(\Omega)$ the class of differentiable functions $f : \Omega \to \mathbb{R}$ whose partial derivative of order 1 are Lipschitz-continuous, equipped with the seminorm

$$\|f\|_{1,1} = \sup \left\{ \frac{|D^\nu f(\boldsymbol{u}) - D^\nu f(\boldsymbol{v})|}{\|\boldsymbol{u} - \boldsymbol{v}\|_2} : \boldsymbol{u}, \boldsymbol{v} \in \Omega, \boldsymbol{u} \neq \boldsymbol{v}, \|\nu\|_2 = 1 \right\}, \tag{9}$$

where $|\cdot|$ denotes the absolute value of the argument. We also denote by $\boldsymbol{e}_{j_k,j_\ell} = \boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_\ell}$, with $\ell, k = 1, 2, 3, 4$, the edge vectors of the tetrahedron t_j. Then, the following result holds.

Proposition 3 *Let $f \in C^{1,1}(\Omega)$ and $t_j \in T$ a tetrahedron of vertices $\boldsymbol{x}_{j_1}$, $\boldsymbol{x}_{j_2}$, $\boldsymbol{x}_{j_3}$, $\boldsymbol{x}_{j_4}$. Then, for all $\boldsymbol{x} \in \Omega$ we have*

$$|f(\boldsymbol{x}) - L_j(\boldsymbol{x})| \leq ||f||_{1,1} \left(3 \, ||\boldsymbol{x} - \boldsymbol{x}_{j_1}||_2^2 + \frac{27}{2} C_j h_j \, ||\boldsymbol{x} - \boldsymbol{x}_{j_1}||_2 \right), \tag{10}$$

where $h_j = \max\limits_{k,\ell=1,2,3,4} \|\boldsymbol{e}_{j_k,j_\ell}\|_2$ and C_j is a constant which depends only on the shape of the tetrahedron t_j.

Proof Let us expand $f(\boldsymbol{x}_{j_k})$, $k = 2, 3, 4$, by the second order Taylor polynomial at $\boldsymbol{x}_{j_1}$ with integral remainder

$$\begin{aligned} f(\boldsymbol{x}_{j_k}) = f(\boldsymbol{x}_{j_1}) + \nabla f(\boldsymbol{x}_{j_1}) \cdot (\boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_1}) \\ + \|\boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_1}\|_2^2 \int_0^1 \frac{\partial^2 f(\boldsymbol{x}_{j_1} + t(\boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_1}))}{\partial \boldsymbol{\nu}_{j_k}^2} (1-t) \, dt, \end{aligned} \tag{11}$$

where $\frac{\partial^2}{\partial \boldsymbol{\nu}_{j_k}^2}$, $k = 2, 3, 4$, is the second order directional derivatives along the unit vectors $\boldsymbol{\nu}_{j_k} = \frac{\boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_1}}{\|\boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_1}\|_2}$, $k = 2, 3, 4$. Substituting (11) in (1) we get

$$\begin{aligned} L_j(\boldsymbol{x}) = f(\boldsymbol{x}_{j_1}) + \sum_{k=2}^{4} \lambda_{j,j_k}(\boldsymbol{x}) \nabla f(\boldsymbol{x}_{j_1}) \cdot (\boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_1}) \\ + \sum_{k=2}^{4} \lambda_{j,j_k}(\boldsymbol{x}) \|\boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_1}\|_2^2 \int_0^1 \frac{\partial^2 f(\boldsymbol{x}_{j_1} + t(\boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_1}))}{\partial \boldsymbol{\nu}_{j_k}^2} (1-t) \, dt \end{aligned}$$

and then

$$L_j(\boldsymbol{x}) = T[f, \boldsymbol{x}_{j_1}](\boldsymbol{x}) + \delta_j(\boldsymbol{x}),$$

where

$$T_1[f, \boldsymbol{x}_{j_1}](\boldsymbol{x}) = f(\boldsymbol{x}_{j_1}) + \sum_{k=2}^{4} \lambda_{j,j_k}(x) \nabla f(\boldsymbol{x}_{j_1}) \cdot (\boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_1})$$

is the first order Taylor polynomial of f at $\boldsymbol{x}_{j_1}$ and

$$\delta_j(\boldsymbol{x}) = \sum_{k=2}^{4} \lambda_{j,j_k}(\boldsymbol{x}) \|\boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_1}\|_2^2 \int_0^1 \frac{\partial^2 f(\boldsymbol{x}_{j_1} + t(\boldsymbol{x}_{j_k} - \boldsymbol{x}_{j_1}))}{\partial \nu_{j_k}^2} (1-t)\, dt.$$

By the triangle inequality we have

$$|f(\boldsymbol{x}) - L_j(\boldsymbol{x})| \leq |f(\boldsymbol{x}) - T_1[f, \boldsymbol{x}_{j_1}](\boldsymbol{x})| + |\delta_j(\boldsymbol{x})|$$

,where we bound the remainder term in Taylor expansion in a standard way

$$|f(\boldsymbol{x}) - T_1[f, \boldsymbol{x}_{j_1}](\boldsymbol{x})| \leq 3 ||f||_{1,1} ||\boldsymbol{x} - \boldsymbol{x}_{j_1}||_2^2$$

and, to bound the term $|\delta_j(\boldsymbol{x})|$, we recall that

$$|\lambda_{j,j_i}(\boldsymbol{x})| \leq \frac{h_j^2}{|V(\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4})|} ||\boldsymbol{x} - \boldsymbol{x}_{j_1}||_2, \quad i = 2, 3, 4.$$

Then, by setting

$$C_j = \frac{h_j^3}{|V(\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4})|} \tag{12}$$

we have

$$|\delta_j(\boldsymbol{x})| \leq \frac{27}{2} ||f||_{1,1} C_j h_j ||\boldsymbol{x} - \boldsymbol{x}_{j_1}||_2.$$

Now we prove that the constant C_j depends only on the shape of t_j. To this aim we recall the Cayley-Menger formula [17] for the computation of the volume of t_j

$$V(\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4})^2 = \frac{1}{288} \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & \|\boldsymbol{e}_{j_1,j_2}\|_2^2 & \|\boldsymbol{e}_{j_1,j_3}\|_2^2 & \|\boldsymbol{e}_{j_1,j_4}\|_2^2 \\ 1 & \|\boldsymbol{e}_{j_2,j_1}\|_2^2 & 0 & \|\boldsymbol{e}_{j_2,j_3}\|_2^2 & \|\boldsymbol{e}_{j_2,j_4}\|_2^2 \\ 1 & \|\boldsymbol{e}_{j_3,j_1}\|_2^2 & \|\boldsymbol{e}_{j_3,j_2}\|_2^2 & 0 & \|\boldsymbol{e}_{j_3,j_4}\|_2^2 \\ 1 & \|\boldsymbol{e}_{j_4,j_1}\|_2^2 & \|\boldsymbol{e}_{j_4,j_2}\|_2^2 & \|\boldsymbol{e}_{j_4,j_3}^2\|_2 & 0 \end{vmatrix}$$

and, since $\|\boldsymbol{e}_{j_k,j_\ell}\|_2 = \alpha_{j_k,j_\ell} h_j$ with $\alpha_{j_k,j_\ell} \in \mathbb{R}$, it follows that

$$\begin{aligned} V(\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4})^2 = \\ = h_j^6 \Bigg[& \frac{1}{144} \left(\alpha_{j_1,j_2}^2 + \alpha_{j_1,j_3}^2 + \alpha_{j_1,j_4}^2 + \alpha_{j_2,j_3}^2 + \alpha_{j_2,j_4}^2 + \alpha_{j_3,j_4}^2\right) \times \\ & \times \left(\alpha_{j_1,j_2}^2 \alpha_{j_3,j_4}^2 + \alpha_{j_1,j_3}^2 \alpha_{j_2,j_4}^2 + \alpha_{j_1,j_4}^2 \alpha_{j_2,j_3}^2\right) + \\ & - \frac{1}{72} \alpha_{j_1,j_2}^2 \alpha_{j_3,j_4}^2 \left(\alpha_{j_1,j_2}^2 + \alpha_{j_3,j_4}^2\right) - \frac{1}{72} \alpha_{j_1,j_3}^2 \alpha_{j_2,j_4}^2 \left(\alpha_{j_1,j_3}^2 + \alpha_{j_2,j_4}^2\right) + \\ & - \frac{1}{72} \alpha_{j_1,j_4}^2 \alpha_{j_2,j_3}^2 \left(\alpha_{j_1,j_4}^2 + \alpha_{j_2,j_3}^2\right) - 144 \left(\alpha_{j_1,j_2}^2 \alpha_{j_1,j_3}^2 \alpha_{j_2,j_3}^2 - \right. \\ & \left. + \alpha_{j_1,j_2}^2 \alpha_{j_1,j_4}^2 \alpha_{j_2,j_4}^2 + \alpha_{j_1,j_3}^2 \alpha_{j_1,j_4}^2 \alpha_{j_3,j_4}^2 + \alpha_{j_2,j_3}^2 \alpha_{j_2,j_4}^2 \alpha_{j_3,j_4}^2\right) \Bigg]. \end{aligned}$$

As a consequence

$$V\left(\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \boldsymbol{x}_{j_3}, \boldsymbol{x}_{j_4}\right) = \alpha_j h_j^3,$$

where

$$\begin{aligned}\alpha_j = \Bigg[& \frac{1}{144}\left(\alpha_{j_1,j_2}^2 + \alpha_{j_1,j_3}^2 + \alpha_{j_1,j_4}^2 + \alpha_{j_2,j_3}^2 + \alpha_{j_2,j_4}^2 + \alpha_{j_3,j_4}^2\right) \\ & \left(\alpha_{j_1,j_2}^2\alpha_{j_3,j_4}^2 + \alpha_{j_1,j_3}^2\alpha_{j_2,j_4}^2 + \alpha_{j_1,j_4}^2\alpha_{j_2,j_3}^2\right) + \\ & -\frac{1}{72}\alpha_{j_1,j_2}^2\alpha_{j_3,j_4}^2\left(\alpha_{j_1,j_2}^2 + \alpha_{j_3,j_4}^2\right) - \frac{1}{72}\alpha_{j_1,j_3}^2\alpha_{j_2,j_4}^2\left(\alpha_{j_1,j_3}^2 + \alpha_{j_2,j_4}^2\right) \\ & -\frac{1}{72}\alpha_{j_1,j_4}^2\alpha_{j_2,j_3}^2\left(\alpha_{j_1,j_4}^2 + \alpha_{j_2,j_3}^2\right) - 144\left(\alpha_{j_1,j_2}^2\alpha_{j_1,j_3}^2\alpha_{j_2,j_3}^2 + \right. \\ & \left.\alpha_{j_1,j_2}^2\alpha_{j_1,j_4}^2\alpha_{j_2,j_4}^2 + \alpha_{j_1,j_3}^2\alpha_{j_1,j_4}^2\alpha_{j_3,j_4}^2 + \alpha_{j_2,j_3}^2\alpha_{j_2,j_4}^2\alpha_{j_3,j_4}^2\right)\Bigg]^{1/2},\end{aligned}$$

and this proves that the constant C_j in (12) is independent from h_j.

□

To determine the approximation order of the operator $\mathcal{T}_\mu[f]$ we need some additional notations and definitions. Let $||\cdot||_\infty$ be the maximum norm and $B_r(\boldsymbol{y}) = \left\{\boldsymbol{x} \in \mathbb{R}^3 : ||\boldsymbol{x} - \boldsymbol{y}||_\infty \leq r\right\}$ be the closed cube of center $\boldsymbol{y}$ and radius r. With $\mathcal{V}(t)$ we denote the set of vertices of the tetrahedron t and we define

$$h' = \inf\left\{r > 0 : \forall \boldsymbol{x} \in \Omega \exists t \in T : B_r(\boldsymbol{x}) \cap \mathcal{V}(t) \neq \emptyset\right\}, \tag{13}$$

$$h'' = \inf\left\{r > 0 : \forall t \in T \exists \boldsymbol{x} \in \Omega : t \subset B_r(\boldsymbol{x})\right\} \tag{14}$$

and

$$h = \max\left\{h', h''\right\}. \tag{15}$$

Finally, we set

$$M = \sup_{\boldsymbol{x} \in \Omega} \sharp\left\{t \in T : B_h(\boldsymbol{x}) \cap \mathcal{V}(t) \neq \emptyset\right\}, \tag{16}$$

where $\sharp$ denotes the cardinality operator. M is the maximum number of tetrahedra with at least a vertex in some cube with edge length $2h$.

Theorem 1 *Let Ω be a compact convex domain which contains X_n, $f \in C^{1,1}(\Omega)$ and $\mu > 5/4$. Then*

$$|f(\boldsymbol{x}) - \mathcal{T}_\mu[f](\boldsymbol{x})| \leq CM\,||f||_{1,1}\,h^2 \tag{17}$$

for any $\boldsymbol{x} \in \Omega$, with C a positive constant which depends on T and μ and M defined in (16).

Proof The proof follows by Proposition 3 and by [9, Theorem 3.1] by setting $s = 3$, $p = 1$ and $\sigma = 4$.

□

In order to implement the tetrahedral Shepard method (3) we need to identify a 3D-triangulation of the node set X_n. The right hand side of the bound (17) suggests to use triangulations with a smaller number of – as regular as possible – small tetrahedra (see equations (15), (16) and (12)). In analogy with the case of the triangular Shepard method, the 3D-triangulation can be organized in an efficient way by allowing tetrahedra to overlap or be disjoint. In the following section we describe in details a fast algorithm to detect and select subsets of nearby nodes, among them we select the vertices of the tetrahedra.

3 Fast Algorithms for Detection and Search of Points

In this section we present the algorithms that are used to implement our tetrahedral Shepard method (3). The latter requires indeed to find suitable configurations of tetrahedra (whose vertices are given by the data points), which allow us to locally reduce the bound of the error in (10) for the local linear interpolants in (1). It is therefore important to efficiently determine the nearest neighbor points associated with each node because this guarantees to reduce the local error, then enabling us to compute the related tetrahedral basis functions in (2). For this reason, we need to use a fast searching technique for the computation of the nearest neighbor points, which in our case is based on two previous stages consisting in partitioning and localizing the interpolation nodes. More precisely, these algorithms consist of three stages and briefly can be sketched as follows:

- `Stage 1:` for each interpolation point we define a ball centred at that point, which is used for the localization of the nearest neighbor points;
- `Stage 2:` we construct a partitioning structure that partition the domain and the interpolation points in cube-shape blocks;
- `Stage 3:` combining the two previous phases, we can quickly find the nearest neighbor points by examining only a limited number of cubic blocks.

Similar computational procedures were first introduced to efficiently solve interpolation and differential problems in the context of radial basis function partition of unity methods [7,4,5], and then suitably modified for classical and triangular Shepard-type methods defined on plane, sphere and other manifolds [1, 6]. Although such fast algorithms can be applied to 2D and 3D generic domains, for the sake of brevity and clarity, here we describe our localizing and searching techniques focusing on the unit cube, that is the domain $\Omega = [0,1]^3 \subset \mathbb{R}^3$.

3.1 Description of the Algorithms

`Stage 1: localization phase.` We define a ball of radius

$$\delta = \frac{\sqrt{3}}{d},$$

where

$$d = \left\lfloor \left(\frac{n}{8}\right)^{1/3} \right\rfloor, \tag{18}$$

and each ball is centred at a data point belonging to Ω. We remark that the larger (smaller) the value of d is, the finer (coarser) the structure is. The value (18) is suitably chosen extending the definition contained in [7].

`Stage 2: partitioning phase.` To find the closest points that belong to the different balls and then apply our tetrahedral interpolation method, we construct a structure based on a partition of the domain Ω in cube-shaped blocks. Such technique leads to an effective searching procedure which turns out to be quite efficient from a computational point of view. To be more precise, this structure

allows us to partition the domain Ω with b^3 cubic blocks, where b denotes the number of blocks along one side of the unit cube defined by

$$b = \left\lceil \frac{1}{\delta} \right\rceil.$$

From this assumption we get the side of each cubic block is equal to the ball radius. This choice enables us to examine in the searching procedure only a small number of blocks, so as to importantly reduce the computational cost as compared to the most advanced searching techniques, as for instance the kd-trees [2,18]. This benefit is indeed proved by the fact that our searching process is carried out in constant time, i.e. $O(1)$. Further, in this partitioning phase we number the cube-shaped blocks from 1 to b^3. As a consequence, by repeatedly using a *quicksort* routine, we can split the set X_n in the b^3 cubic blocks, so that we can easily construct the subsets X_{n_k}, $k = 1, \ldots, b^3$, where X_{n_k} contains the points belonging to twenty-seven blocks: the k-th block and its twenty-six neighboring blocks. In such framework, we are able to obtain a fast searching procedure to detect the interpolation points nearest to each of nodes. In Figure 1 we show an example of domain partitioning in cube-shaped blocks along with a set of scattered data points.

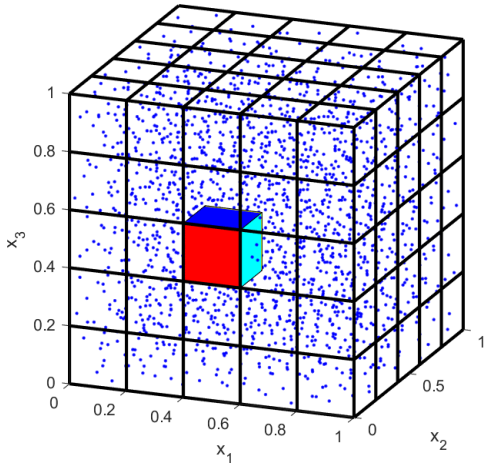

Fig. 1: Example of 3D block-based partitioning structure with a set of interpolation nodes (blue dots) contained in $\Omega = [0,1]^3 \subset \mathbb{R}^3$.

`Stage 3: searching phase.` After arranging the interpolation points in the b^3 cubic blocks, we need to answer the following two queries, known respectively as *containing query* and *range search*, i.e.

a) given a point that belongs to the domain Ω, return the k-th cubic block containing that point;

b) given a set X_n of points and a ball, determine all points included in that ball.

Thus, given a data point in the ball, solving the containing query problems consists in finding the index of the k-th cubic block containing such point. As a result, denoted rispectively by k_i, $i = 1, 2, 3$, the index associated with the x_i-axis, we can detect the k-th block by means of the following rule:

$$k = (k_1 - 1)\, b^2 + (k_2 - 1)\, b + k_3.$$

After answering the query a), for each given ball the searching procedure enables us to find all interpolation points lying in the balls, and so we also reply to the query b). Then, among all points belonging to each ball, in the definition of the tetrahedral Shepard method – and specifically in the selection of suitable configurations of tetrahedra – we only consider the n_w nearest neighbor points. In particular, assuming that the node ball belongs to the k-th cubic block, the search routine examines all data points lying in the k-th block and in its twenty-six neighboring blocks. Obviously, if a block lies on the boundary of the cubic domain Ω, the partitioning structure and the search process allows us to further reduce the number of neighboring blocks that need to be examined.

3.2 Computational Complexity

The localization phase described in `Stage 1` is basically a sort of data pre-processing that is not involved in computational cost. As a result, we focus our attention on `Stage 2` where we partition the n interpolation points in cube-shaped blocks. In the assessment of the total complexity we should also take into account the cost associated with the storing of the evaluation points. However, for simplicity and brevity, we here refer to the interpolation data points only.

As illustrated in `Stage 2`, the partitioning structure in b^3 cubic blocks is based on the use of a quicksort routine, i.e. recursive calls to the MATLAB `sortrows.m` routine. It has $O(n \log n)$ time complexity and requires $O(\log n)$ space, n being the number of points that need to be sorted. This approach enables us to partition the unit cube Ω, organizing the points in a block-based structure. Therefore, we can estimate that the cost of the second phase is given by $O\left(2n \log n + 6n\right)$.

Finally, as regards the search technique of `Stage 3` we need to apply again a quicksort routine to sort distances of points within each ball. Since the data point distribution is supposed to be quite uniform (and so it is in the twenty-seven neighboring cube-shaped blocks), we can estimate the complexity of this search phase is $O(1)$.

4 Trivariate Interpolation Algorithm

In this section we describe in a pseudo-code the interpolation algorithm, which performs the tetrahedral Shepard method (3) using the block-based partitioning structure and the related searching procedure. Here, for shortness, the presentation is carried out by assuming that the domain $\Omega = [0, 1]^3 \subset \mathbb{R}^3$. It is however possible to extend the algorithm to generic domain as in [7].

`INPUTS:` n, number of data; $X_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, set of data points; $F_n = \{f_1, \ldots, f_n\}$, set of data values; n_e, number of evaluation points; $Z_{n_e} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n_e}\}$, set of evaluation points; n_w, number of nearest neighbor points for the selection of suitable configurations of tetrahedra.

`OUTPUTS:` $E_{n_e} = \{\mathcal{T}_\mu[f](\boldsymbol{z}_1), \ldots, \mathcal{T}_\mu[f](\boldsymbol{z}_{n_e})\}$, set of approximated values.

`Step 1:` For each point $\boldsymbol{x}_i$, $i = 1, \ldots, n$, construct a ball of radius

$$\delta = \frac{\sqrt{3}}{d}, \quad \text{with} \quad d = \left\lfloor \left(\frac{n}{8}\right)^{1/3} \right\rfloor.$$

`Step 2:` Compute the number b of blocks (along one side of the unit cube Ω) defined by

$$b = \left\lceil \frac{1}{\delta} \right\rceil.$$

`Step 3:` Build the partitioning structure on the domain Ω and split the set X_n of interpolation nodes in b^3 cubic blocks.

`Step 4:` For each ball (or data point), solve the containing query and the range search problems to detect all nodes X_{n_k}, $k = 1, \ldots, b^3$, belonging to the twenty-seven neighboring blocks.

`Step 5:` For each data point $\boldsymbol{x}_i \in X_n$, fix the set $\mathcal{N}(\boldsymbol{x}_i) \subset X_n$ of the n_w nearest neighbors to $\boldsymbol{x}_i$ ordered with respect to the increasing distances from $\boldsymbol{x}_i$. Set t_k, $k = 1, \ldots, \frac{n_w(n_w-1)(n_w-2)}{6}$ one of the tetrahedra with a vertex in $\boldsymbol{x}_i$ and other three vertices in $\mathcal{N}(\boldsymbol{x}_i)$ and set t_i that one for which the quantity $C_k h_k$, in equation (12), is minimum.

`Step 6:` Set $T = \{t_i\}_{i=1}^n$, where equal tetrahedra are identified.

`Step 7:` Compute the local basis function $B_{\mu,j}(\boldsymbol{z})$, $j = 1, \ldots, m$, at each evaluation point $\boldsymbol{z} \in Z_{n_e}$.

`Step 8:` Compute the linear interpolants $L_j[f](\boldsymbol{z})$, $j = 1, \ldots, m$, at each evaluation point $\boldsymbol{z} \in Z_{n_e}$.

`Step 9:` Apply the tetrahedral Shepard method (3) and evaluate the trivariate interpolant at the evaluation points $\boldsymbol{z} \in Z_{n_e}$.

5 Numerical Results

In this section we illustrate the performance of our trivariate interpolation algorithm, which is implemented in MATLAB. All the numerical experiments have been carried out on a laptop with an Intel(R) Core i7 6500U CPU 2.50GHz processor and 8.00GB RAM.

In the following presentation of our results we discuss about several tests carried out, which refer to solving very large interpolation problems by means of the tetrahedral Shepard method (3). This analysis is therefore realized by taking two different distributions of irregularly distributed (or scattered) data points

contained in the unit cube $\Omega = [0,1]^3 \subset \mathbb{R}^3$, and considering a number n of interpolation nodes that varies from 10 000 to 80 000. More precisely, as interpolation points we focus on a few sets of low discrepancy Halton points generated through the MATLAB function `haltonset` called with the setting `haltonset(2,'Skip',1)`, and pseudo-random points obtained by using the `rand` MATLAB command. In addition, the interpolation errors are computed on a grid consisting of $n_e = 21 \times 21 \times 21$ evaluation points, while we fix the value $n_w = 13$ which guarantees a sufficient number of tetrahedra for an effective choice of the triangulation T.

In the various experiments we thus analyze the performance of our interpolation algorithm assuming the data values are given by the following four trivariate test functions [15]:

$$\begin{aligned}
f_1(x_1,x_2,x_3) &= \frac{3}{4}\exp\left(-\frac{(9x_1-2)^2+(9x_2-2)^2+(9x_3-2)^2}{4}\right) \\
&+ \frac{3}{4}\exp\left(-\frac{(9x_1+1)^2}{49}-\frac{9x_2+1}{10}-\frac{9x_3+1}{10}\right) \\
&+ \frac{1}{2}\exp\left(-\frac{(9x_1-7)^2+(9x_2-3)^2+(9x_3-5)^2}{4}\right) \\
&- \frac{1}{5}\exp\left(-(9x_1-4)^2-(9x_2-7)^2-(9x_3-5)^2\right), \\
f_2(x_1,x_2,x_3) &= \frac{\tanh(9x_3-9x_1-9x_2)+1}{9}, \\
f_3(x_1,x_2,x_3) &= \frac{1}{9}\sqrt{64-81\left((x_1-0.5)^2+(x_2-0.5)^2+(x_3-0.5)^2\right)}-0.5, \\
f_4(x_1,x_2,x_3) &= \frac{1}{1+50((x_1-0.5)^2+(x_2-0.5)^2+(x_3-0.5)^2)}.
\end{aligned}$$

These functions are usually used in approximation schemes to test and validate new methods and algorithms (see e.g. [3,13]).

As a measure of the quality/accuracy of our results, we compute the Maximum Absolute Error (MAE) and the Root Mean Square Error (RMSE), whose formulas are respectively given by

$$\mathrm{MAE} = ||f - \mathcal{T}_\mu[f]||_\infty = \max_{1\le i\le n_e} |f(\boldsymbol{z}_i) - \mathcal{T}_\mu[f](\boldsymbol{z}_i)|$$

and

$$\mathrm{RMSE} = \frac{1}{\sqrt{n_e}}||f - \mathcal{T}_\mu[f]||_2 = \sqrt{\frac{1}{n_e}\sum_{i=1}^{n_e} |f(\boldsymbol{z}_i) - \mathcal{T}_\mu[f](\boldsymbol{z}_i)|^2},$$

where $\boldsymbol{z}_i \in Z_{n_e}$ is an evaluation point belonging to the domain Ω.

In Tables 1–2 we report MAEs and RMSEs that decrease when the number n of interpolation points increases. Comparing then the errors obtained by using the two data distributions, we can note that a (slightly) better accuracy is achieved whenever we employ Halton nodes. This fact is basically due to greater level of regularity of Halton points than pseudo-random MATLAB nodes. Analyzing the error behavior with the test functions f_1, f_2, f_3 and f_4, we get similar results in terms of accuracy of the interpolation scheme, although the function f_1

results in a slightly lessen precision. In order to test the quadratic approximation order of the tetrahedral Shepard method stated in Theorem 1 we have considered a set of Halton nodes with increasing resolution. Table 3 lists the number of interpolation points, the number of tetrahedra and the maximum edge length $h_T = \max\{h_1, h_2, \ldots, h_m\}$. In this experiment the number n of nodes is chosen so that at each step the value h_T is halved.

	f_1		f_2		f_3		f_4	
n	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
10 000	6.23e-2	2.98e-3	2.18e-2	1.97e-3	1.03e-2	1.12e-3	4.14e-2	2.04e-3
20 000	3.11e-2	1.76e-3	2.17e-2	1.28e-3	4.86e-3	6.92e-4	4.87e-2	1.37e-3
40 000	2.02e-2	1.22e-3	1.92e-2	9.40e-4	2.57e-3	4.65e-4	3.71e-2	1.11e-3
80 000	9.46e-3	7.58e-4	9.13e-3	6.07e-4	1.87e-3	2.92e-4	2.85e-2	6.24e-4

Table 1: MAE and RMSE computed on Halton points.

	f_1		f_2		f_3		f_4	
n	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
10 000	6.78e-2	3.89e-3	4.51e-2	2.68e-3	1.37e-2	1.60e-3	8.34e-2	3.02e-3
20 000	3.99e-2	2.51e-3	2.07e-2	1.65e-3	8.18e-3	1.00e-3	4.20e-2	1.78e-3
40 000	3.45e-2	1.59e-3	1.75e-2	1.09e-3	5.30e-3	6.37e-4	3.81e-2	1.26e-3
80 000	1.56e-2	1.05e-3	1.10e-2	7.27e-4	2.66e-3	4.11e-4	2.51e-2	7.71e-4

Table 2: MAE and RMSE computed on pseudo-random MATLAB points.

With regard to the efficiency of our trivariate algorithm in Table 4 we show the CPU times computed in seconds, comparing the performance of our searching procedure based on the partitioning of domain and data points in cubic blocks (t_{fast}) with a standard implementation of the algorithm where one computes all the distances between the interpolation nodes ($t_{standard}$). From this study we highlight a remarkable enhancement in terms of computational efficiency when the

n	m	h_T
100	66	5.3968e-1
600	404	2.7502e-1
4850	3066	1.3721e-1
47007	29151	6.7123e-2
500000	290932	3.4831e-2

Table 3: The number m of tetrahedra and the maximum edge length h_T for different sets of n Halton points.

new partitioning and searching techniques are applied. Furthermore, we observe that the computation of errors and execution times via the standard procedure is not allowed by MATLAB for $n > 30\,000$ points, since it is very expensive from a computational standpoint and memory required does not turn out to be sufficient to complete the entire process. In the tables we denote this drawback with the symbol –. On the contrary, the tetrahedral Shepard interpolant (3) can be applied successfully – without any particular issues and in an efficient way – when the block-based technique is used. Then, in order to emphasize the high efficiency of our new algorithm, we also compute the CPU time ratios between t_{fast} and $t_{standard}$, i.e. $t_{fast}/t_{standard}$. Though this comparison is only possible until $n = 30\,000$ points, these results are enough to point out a considerable saving of time above all when the number of interpolation nodes tends to become larger and larger.

	Halton points			MATLAB points		
n	$t_{standard}$	t_{fast}	$t_{fast}/t_{standard}$	$t_{standard}$	t_{fast}	$t_{fast}/t_{standard}$
10 000	40.2	27.0	0.67	42.1	27.1	0.64
20 000	317.3	54.3	0.17	320.0	55.6	0.17
30 000	877.8	103.8	0.12	902.1	102.8	0.11
40 000	–	146.4	–	–	151.5	–
50 000	–	221.0	–	–	233.6	–
60 000	–	322.1	–	–	330.3	–
70 000	–	442.4	–	–	454.2	–
80 000	–	605.0	–	–	636.3	–

Table 4: Comparison of CPU times obtained by using our new fast procedures (t_{fast}) and the standard ones ($t_{standard}$).

Moreover, to make even clearer the obtained results concerning the efficiency, in Figure 2 we graphically compare the CPU times obtained by using the fast algorithm, for both Halton and pseudo-random MATLAB data points. In this study we thus report the execution times for f_1 only, because the time behavior with other test functions is similar.

Finally, in Figure 3 we show the log-log-plot of the MAE over the maximum edge length h_T for the four test functions and the set of tetrahedra given in Table 3.

6 Conclusions and Future Work

In this article we proposed a new trivariate algorithm to efficiently interpolate irregularly distributed or scattered data points through the tetrahedral Shepard method. Since this interpolation scheme needs to find suitable tetrahedra associated with the nodes, we considered a fast searching procedure based on the partitioning of domain/nodes in cube-shaped blocks. Such a technique turned out

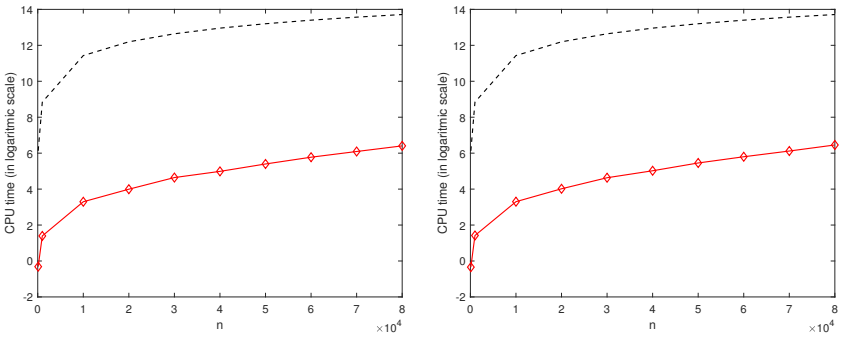

Fig. 2: CPU times (in seconds) in logarithmic scale computed on Halton (left) and pseudo-random MATLAB (right) points for f_1 by using the fast algorithm (the dashed line is the logarithmic plot of $n \log n$).

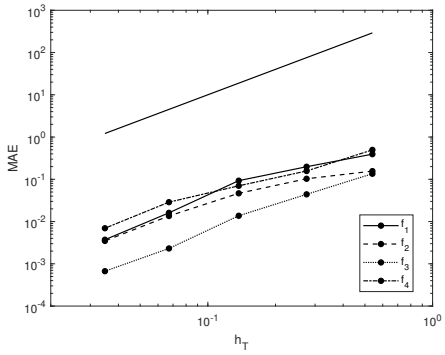

Fig. 3: Log-log-plot of the approximation error MAE over the maximum edge length h_T for the four test functions and the set of tetrahedra in Table 3. As a reference the solid line indicates a quadratic trend.

to be computationally more efficient than a standard implementation of the interpolation algorithm. Numerical experiments showed good performance of our procedures, which enabled us to quickly deal with a large number of points, whereas standard routines were not able to solve the approximation problems in all considered cases.

As future work we propose to study a new triangular/tetrahedral Shepard method, which can be applied on the sphere $\mathbb{S}^2$ or other manifolds (see e.g. [1, 19]).

Acknowledgements This work was partially supported by the INdAM-GNCS 2018 research project "Methods, algorithms and applications of multivariate approximation" and by the 2018 project "Mathematics for applications" funded by the Department of Mathematics "Giuseppe Peano" of the University of Torino. This research has been accomplished within RITA (Research ITalian network on Approximation). All the authors are members of the INdAM Research group GNCS.

References

1. Allasia, G., Cavoretto, R., De Rossi, A.: Hermite-Birkhoff interpolation on scattered data on the sphere and other manifolds. Appl. Math. Comput. **318**, 35–50 (2018)
2. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J. ACM **45**, 891–923 (1998)
3. Bozzini, M., Rossini, M.: Testing methods for 3D scattered data interpolation. Monogr. Real Acad. Ci. Exact. Fis.-Quim. Nat. Zaragoza **20**, 111–135 (2002)
4. Cavoretto, R., A. De Rossi: Adaptive meshless refinement schemes for RBF-PUM collocation. Appl. Math. Lett. **90**, 131–138 (2019)
5. Cavoretto, R., De Rossi, A.: Error indicators and refinement strategies for solving Poisson problems through a RBF partition of unity collocation scheme. Appl. Math. Comput. **369**, 124824 (2020)
6. Cavoretto, R., De Rossi, A., Dell'Accio, F., Di Tommaso, F.: Fast computation of triangular Shepard interpolants. J. Comput. Appl. Math. **354**, 457–470 (2019)
7. Cavoretto, R., De Rossi, A., Perracchione, E.: Efficient computation of partition of unity interpolants through a block-based searching technique. Comput. Math. Appl. **71**, 2568–2584 (2016)
8. Dell'Accio, F., Di Tommaso, F.: Scattered data interpolation by Shepard's like methods: Classical results and recent advances. Dolomites Res. Notes Approx. **9**, 32–44 (2016)
9. Dell'Accio, F., Di Tommaso, F.: Rate of convergence of multinode Shepard operators. Dolomites Res. Notes Approx. **12**, 1–6 (2019)
10. Dell'Accio, F., Di Tommaso, F., Hormann, K.: On the approximation order of triangular Shepard interpolation. IMA J. Numer. Anal. **36**, 359–379 (2016)
11. Fasshauer, G.E.: Meshfree Approximation Methods with MATLAB. World Scientific Publishing Co., Inc., Singapore (2007)
12. Golin, M.J., Na, H.S.: On the average complexity of 3d-voronoi diagrams of random points on convex polytopes. Computational Geometry **25**(3), 197 – 231 (2003). DOI https://doi.org/10.1016/S0925-7721(02)00123-2. URL `http://www.sciencedirect.com/science/article/pii/S0925772102001232`
13. Lazzaro, D., Montefusco, L.B.: Radial basis functions for the multivariate interpolation of large scattered data sets. J. Comput. Appl. Math. **140**, 521–536 (2002)
14. Little, F.F.: Convex combination surfaces. In: R.E. Barnhill, W. Boehm (eds.) Surfaces in Computer Aided Geometric Design, vol. 1479, pp. 99–108. North-Holland (1983)
15. Renka, R.J.: Multivariate interpolation of large sets of scattered data. ACM Trans. Math. Softw. **14**, 139–148 (1988)
16. Shepard, D.: A two-dimensional interpolation function for irregularly-spaced data. In: Proceedings of the 1968 23rd ACM National Conference, ACM '68, pp. 517–524. ACM, New York, NY, USA (1968)
17. Uspensky, J.V.: Theory of equations / J.V. Uspensky. McGraw-Hill New York (1948)
18. Wendland, H.: Scattered Data Approximation. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press (2005)
19. Zhang, M., Liang, X.Z.: On a Hermite interpolation on the sphere. Appl. Numer. Math. **61**, 666–674 (2011)