JOURNAL OF COMPUTATIONAL PHYSICS **132,** 157–166 (1997)
ARTICLE NO. CP965604

A Generalized Particle Search–Locate Algorithm for Arbitrary Grids

Alejandro Allievi* and Rodolfo Bermejo†

*2660 Kent Ave., 200, Montréal, Québec, Canada H35 1M7;
†Departamento de Matemática Aplicada, Facultad de Ciencias Matemáticas, Universidad Complutense de Madrid, Madrid 28040, Spain

Received January 29, 1996; revised September 18, 1996

A generalized iterative algorithm for searching and locating particles in arbitrary meshes is presented. The algorithm uses Newton method to invert a bijective map of the mesh elements onto a reference element, together with a criterion to move from element to element in the mesh. The generality of the method is shown by explicit formulations for linear and quadratic triangular and quadrilateral elements. Numerical examples demonstrate the performance of the method as well as its higher accuracy and versatility. © 1997 Academic Press

1. INTRODUCTION

Computer models based on particle dynamics, known as particle methods, are often used in science and engineering. The book by Hockney and Eastwood [1] gives a comprehensive account of these methods and their application in plasma physics, astrophysics, and molecular dynamics. Numerical algorithms closely related to the idea of particle methods are also becoming prominent in atmospheric sciences and fluid dynamics [2–5]. Such algorithms are known as semi-Lagrangian schemes in meteorology, and characteristic-Galerkin methods in the context of finite elements. Several particle methods have in common at least two interrelated features, namely a searching step followed by an interpolation step. The interrelation arises from the fact that, to perform interpolation at a point located in the interior of a mesh, identification of the host element is generally required.

Particle methods as well as semi-Lagrangian and characteristic-Galerkin methods are computationally useful by virtue of their accuracy and linear unconditional stability. This allows for larger time steps in the integration process and hence, for a given degree of accuracy, a potentially faster and more economical numerical procedure. However, if the mesh is not structured, or is nonuniform in each coordinate direction, these numerical methods may be quite slow. The underlying reason is the poor performance of the algorithms in the searching and interpolation steps. In order to overcome this drawback, there have been a number of researchers who have proposed some schemes to carry out searching and interpolation efficiently. Brackbill and Ruppert [6] and Westermann [7] have designed schemes to perform searching and bilinear interpolation on structured meshes composed of nonuniform quadrilaterals. Löhner and Ambrosiano [8] devised an algorithm for unstructured meshes composed of linear triangles or tetrahedra. Although they differ on implementational aspects, all these works share the idea described in the following.

Given a domain Ω (for simplicity, we assume that Ω is in the plane, i.e., $\Omega \subset \mathbf{R}^2$) on which a mesh has been generated, one wishes to know the element Ω_j of the mesh, where the point $\mathbf{x}_p$ is located. The spatial location $\mathbf{x}_p$ is an interpolation point for a vector or scalar field whose values are known at the mesh-points. The schemes proposed in [6–8] to find the element Ω_j define a one-to-one mapping $\mathbf{F}_j$ from Ω_j to a reference element $\hat{\Omega}$ defined in the plane (p, q). Then, if $\mathbf{x}_p \in \Omega_j$ there exists one and only one point $\overline{\mathbf{p}} = (\bar{p}, \bar{q}) \in \hat{\Omega}$ such that

$$\mathbf{x}_p = \mathbf{F}_j(\overline{\mathbf{p}}), \tag{1}$$

and

$$a_0 \le \bar{p} \le b_0,$$
$$c_0 \le \bar{q} \le d_0,$$

where the real constants a_0, b_0, c_0, and d_0 are known by construction (boldface characters denote vectors or vector-valued functions unless otherwise stated). If $\mathbf{x}_p \notin \Omega_j$, then at least one of the following conditions is satisfied:

$$\mathbf{x}_p \notin \Omega_j \rightarrow \begin{cases} \bar{p} \notin [a_0, b_0], \text{ or} \\ \bar{q} \notin [c_0, d_0], \text{ or} \\ \text{both.} \end{cases} \tag{2}$$

Provided that (2) is true, the schemes have a criterium to choose a neighbour element of Ω_j in order to define a new $\mathbf{F}_j$ and continue the test. Löhner and Ambrosiano [8] use finite element techniques to construct $\mathbf{F}_j$ on linear simplexes. At this point, we should remark that their scheme for simplexes and Westermann's [7] for quadrilaterals build the mapping $\mathbf{F}_j$ using linear and bilinear interpolation, respectively. For cases such as curved simplexes and quadrilaterals these schemes may experience difficulties, or even fail to find the rigth element where the point $\mathbf{x}_p$ is located. This is demonstrated by a numerical example presented in Section 4.

In this paper, we propose a scheme for search and location based on similar ideas as those mentioned above, *but* it represents a generalization of previous works. The salient features of our algorithm are the following:

(a) In contrast with the algorithms proposed in [6–8], ours can be applied to curved elements, both simplexes and quadrilaterals, with higher order interpolation functions to define the mapping $\mathbf{F}_j$.

(b) For linear simplexes, our algorithm coincides with that of Löhner and Ambrosiano [8]. However, unlike Westermann's scheme [7], it does not require the solution of a quadratic equation to determine $(\bar{p}, \bar{q})$ for linear quadrilaterals.

(c) The vectorization of our algorithm can be performed in the same manner as in [7, 8]. Furthermore, it can also be used as the basic scheme in the hierarchy of robust, vectorizable algorithms proposed by Löhner [9].

In addition, implementation of our scheme in any particle method is straightforward, particularly in finite element codes since the concepts presented in this work are standard in finite element technology.

The remainder of the paper is organized as follows. We describe in Section 2 the scheme. In Section 3 we study the convergence. Section 4 is devoted to illustrate the performance of the scheme through some significant tests. Finally, some conclusions are included in Section 5.

2. SEARCH–LOCATE ALGORITHM

For simplicity, we formulate the algorithm in a domain $\Omega \subset \mathbf{R}^2$. Application of the algorithm to a domain $\Omega \subset \mathbf{R}^3$ will be clear from the formulation. We assume that Ω is partitioned into small elements Ω_j; some of them may have curved boundaries, such that given an integer $M \geq 1$,

$$\overline{\Omega} = \bigcup_{j=1}^{M} \Omega_j,$$

where $\overline{\Omega}$ denotes the domain plus its boundary and M is the number of elements that form the numerical mesh. The vertices of Ω_j are the mesh-points. Although, the elements Ω_j can be either triangles or quadrilaterals of any

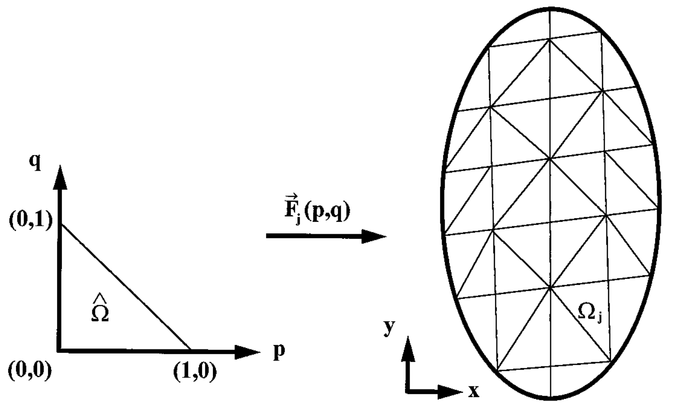


FIG. 1. Mapping from reference element $\hat{\Omega}$ to an element Ω_j of the grid.

size, it is convenient from a numerical point of view to impose some regularity conditions on the mesh elements. We now formulate such conditions:

- First, we assume that the mesh is regular in the sense that there exists a constant $\sigma > 0$, such that if h_j is the largest dimension of Ω_j and ρ_j is the diameter of the largest circle (or sphere) inscribed in Ω_j, then $h_j/\rho_j \leq \sigma$, for any Ω_j. This condition implies that arbitrarily thin elements, or elements with arbitrarily small angles are not allowed.
- Second, let $\Omega_j \neq \Omega_k$, then

$$\Omega_j \cap \Omega_k = \begin{cases} \emptyset, & \text{no intersection, or} \\ \Gamma_{ij}, & \text{one side in common, or} \\ \mathbf{x}_{ij}, & \text{one point in common.} \end{cases}$$

These are standard conditions that any mesh generating algorithm is supposed to satisfy. Next, given a point $\mathbf{x}_p = (x_p, y_p)$ in Ω, we wish to know whether $\mathbf{x}_p$ is in Ω_j. In order to do so, we shall assume that for any Ω_j there exists a one-to-one mapping $\mathbf{F}_j$ from an element of reference $\hat{\Omega}$ to Ω_j. Figure 1 illustrates this idea. Observe that $\hat{\Omega}$ is in the coordinate plane (p, q) such that

$$\begin{aligned} \hat{\Omega} &= \{(p,q) \mid -1 \leq p, q \leq 1\} \quad \text{for quadrilaterals,} \\ \hat{\Omega} &= \{(p,q) \mid 0 \leq p, q \leq 1, 0 \leq 1 - p - q \leq 1\} \\ &\qquad \text{for triangles.} \end{aligned} \tag{3}$$

Following [8], we set

$$\begin{aligned} N_1 &= 1 - p - q, \\ N_2 &= p, \\ N_3 &= q, \end{aligned} \tag{4}$$

so that we can define $\hat{\Omega}$ for triangles

$$\hat{\Omega} = \{(p,q) \mid \max(N_1, N_2, N_3) \le 1 \text{ and } \min(N_1, N_2, N_3) \ge 0\}. \tag{5}$$

Therefore, if for any $j = 1, 2, \ldots, M$, $\mathbf{x}_p = (x_p, y_p) \in \Omega_j$, then there exists $(\bar{p}, \bar{q}) \in \hat{\Omega}$ such that

$$\begin{aligned} G_{1j}(\bar{p}, \bar{q}) &\equiv x_p - F_{1j}(\bar{p}, \bar{q}) = 0, \\ G_{2j}(\bar{p}, \bar{q}) &\equiv y_p - F_{2j}(\bar{p}, \bar{q}) = 0. \end{aligned} \tag{6}$$

On the other hand, if for some j, $\mathbf{x}_p = (x_p, y_p) \notin \Omega_j$, then there is no $(\bar{p}, \bar{q}) \in \hat{\Omega}$ such that (6) holds. Hence, to find $(\bar{p}, \bar{q}) \in \hat{\Omega}$ that satisfies (6) is equivalent to calculating the unique solution of the equation

$$\mathbf{G}_j(\mathbf{p}) = 0.$$

An efficient procedure to solve (6) is the Newton method, for we know that under certain conditions this method has a quadratic rate of convergence. We shall obtain such conditions below. For the time being, we shall assume that if $\mathbf{x}_p \in \Omega_j$, the Newton method will converge to the unique solution $(\bar{p}, \bar{q}) \in \hat{\Omega}$ independently of the initial guess $(p^0, q^0) \in \hat{\Omega}$. A formulation of the Newton method as a search–locate algorithm to solve (6) is as follows:

(ST1) Assume that $\mathbf{x}_p = (x_p, y_p) \in \Omega_j$ and let $(p^0, q^0) \in \hat{\Omega}$ be an initial guess solution. Then for $k = 0, 1, \ldots, N$

$$\begin{Bmatrix} p^{k+1} \\ q^{k+1} \end{Bmatrix} = \begin{Bmatrix} p^{k} \\ q^{k} \end{Bmatrix} - J_j^{-1}(p^k, q^k) \begin{Bmatrix} x_p - F_{1j}(p^k, q^k) \\ y_p - F_{2j}(p^k, q^k) \end{Bmatrix}. \tag{7}$$

Stage **ST1** is the localization stage of our algorithm.

(ST2) If $\mathbf{x}_p = (x_p, y_p) \notin \Omega_j$, the iterate $(p^{k+1}, q^{k+1}) \notin \hat{\Omega}$. Then, select a neighbour element of Ω_j and go to **ST1**. Stage **ST2** is the search stage of our algorithm. Criteria to perform this stage will be given in the following subsections.

In (7), $J_j^{-1}(p^k, q^k)$ is the inverse of the Jacobian matrix $J_j(p^k, q^k)$ of the mapping $\mathbf{F}_j(p, q)$. This Jacobian matrix is given by

$$J_j(p^k, q^k) = -\begin{bmatrix} \dfrac{\partial F_{1j}(p^k, q^k)}{\partial p} & \dfrac{\partial F_{1j}(p^k, q^k)}{\partial q} \\ \dfrac{\partial F_{2j}(p^k, q^k)}{\partial p} & \dfrac{\partial F_{2j}(p^k, q^k)}{\partial q} \end{bmatrix}. \tag{8}$$

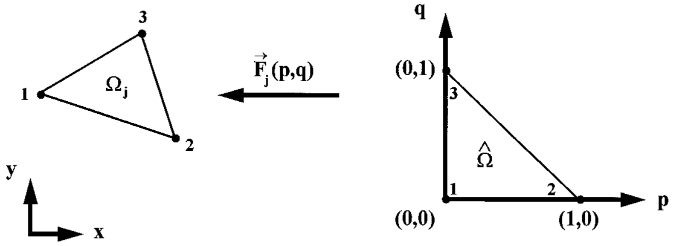


FIG. 2. Mapping for linear triangular elements.

In the following subsections, we focus our attention on the implementation of algorithm (7)–(8) for linear and quadratic simplexes and quadrilaterals. The extension to 3D cases will be clear from the formulation. The mapping $\mathbf{F}_j$ of the reference element $\hat{\Omega}$ onto any $\Omega_j \in \overline{\Omega}$ is defined in this work by the so-called isoparametric technique, extensively used by the finite element community [10].

2.1. *Linear Triangles*

The one-to-one mapping $\mathbf{F}_j : \hat{\Omega} \to \Omega_j$, as shown in Fig. 2, is given as

$$\begin{aligned} F_{1j}(p,q) &\equiv x = \sum_{i=1}^{3} x_i \Phi_i(p,q), \\ F_{2j}(p,q) &\equiv y = \sum_{i=1}^{3} y_i \Phi_i(p,q), \end{aligned} \tag{9}$$

where $\{\Phi_i\}_{i=1}^3$ are the so-called basis functions, whose expressions are

$$\begin{aligned} \Phi_1(p,q) &= 1 - p - q, \\ \Phi_2(p,q) &= p, \\ \Phi_3(p,q) &= q, \end{aligned} \tag{10}$$

and $(x_i, y_i)_{i=1}^3$ denotes the coordinates of the vertices of the triangle Ω_j. Observe that for this particular case the basis functions Φ_i coincide with the functions N_i defined in (4).

Combining (9) and (10), we set

$$\begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \begin{Bmatrix} p \\ q \end{Bmatrix} + \begin{Bmatrix} x_1 \\ y_1 \end{Bmatrix}. \tag{11}$$

Consequently, the mapping $\mathbf{F}_j\,(p, q)$ is an affine transformation of $\hat{\Omega}$ onto Ω_j that maps the vertices of Ω_j. Since the three vertices are not aligned, then the matrix of the transformation can be inverted. Now, if we apply (7) and (8) for the linear triangle and use (9) and (10), we have the procedure

- For $k = 0, 1, 2, \ldots$ set

$$\begin{Bmatrix} p^{k+1} \\ q^{k+1} \end{Bmatrix} = \begin{Bmatrix} p^{k} \\ q^{k} \end{Bmatrix} + \frac{1}{\Delta}\begin{bmatrix} y_3 - y_1 & x_1 - x_3 \\ y_1 - y_2 & x_2 - x_1 \end{bmatrix} \begin{Bmatrix} x_p - \sum_{i=1}^{3} x_i \Phi_i(p^k, q^k) \\ y_p - \sum_{i=1}^{3} y_i \Phi_i(p^k, q^k) \end{Bmatrix}, \tag{12}$$

where $\Delta = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)$.

At this point, it is important to remark that (12) is identical to Eq. (4) of the paper by Löhner and Ambrosiano [8]. Such coincidence is only true for linear triangles, because the matrix in (12) is the inverse matrix of the affine transformation (11) between $\hat{\Omega}$ and Ω_j. Then, if $\mathbf{x}_p \in \Omega_j$, the iterate (p^1, q^1) will be the exact solution $(\bar{p}, \bar{q})$ for any initial guess (p^0, q^0) in $\hat{\Omega}$. Since Ω_j is an arbitrary element, then we have to test whether (p^1, q^1) is in $\hat{\Omega}$, for if this is true then $\mathbf{x}_p \in \Omega_j$. Using (5), $(p^1, q^1) \in \hat{\Omega}$ if

$$\begin{aligned} &\max(N_1^1, N_2^1, N_3^1) \le 1 \\ &\min(N_1^1, N_2^1, N_3^1) \ge 0, \end{aligned} \tag{13}$$

where we have used the notation $N_i^k = N_i(p^k, q^k)$, for $i = 1, 2, 3$. If (13) does not hold, the algorithm chooses a neighbour of Ω_j, according to the selection criterium for simplexes as described in [8], and goes to (9). Such criteria can be expressed as follows:

- Let l = index(min (N_1^k, N_2^k, N_3^k)),
- Choose element Ω_k such that $\Gamma_{jk} = \Omega_j \cap \Omega_k$ is the side of Ω_j opposite to the vertex $\mathbf{x}_l$.

In summary, the search–locate algorithm for simplexes can be divided into the following steps:

Search–Locate Algorithm for Linear Triangles (SALT).

(S1) Given $\mathbf{x}_p$, pick any $\Omega_m \subset \Omega$ and any $\mathbf{p}^0 \in \hat{\Omega}$ to build $\mathbf{F}_m(p^0)$ using (9)–(11).

(S2) Use (12) to find (p^1, q^1).

(S3) Apply (13). If (13) holds STOP, ELSE,

(S4) Apply the selection criterium and GO TO **S1**.

Remarks. (i) If the algorithm is part of an iterative numerical scheme, then it may be advantageous to start **S1** with an Ω_m that is the element where $\mathbf{x}_p$ was located in the previous iteration.

(ii) Sometimes, especially in particle and characteristic-Galerkin methods, the points $\mathbf{x}_p$ may lie outside Ω. This situation can be easily detected by the selection criterion for which one needs to generate a two-dimensional integer array containing the neighbour elements to each element Ω_j. If Ω_j is a boundary element, then some of the entries for Ω_j in such array are zero.

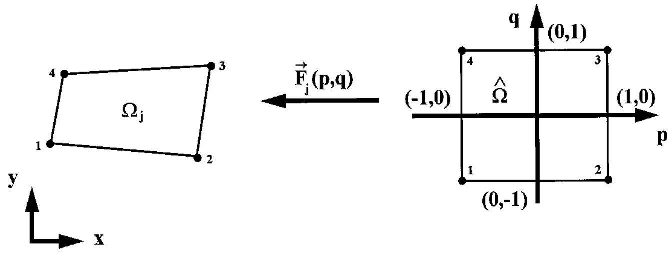


FIG. 3. Mapping for bilinear quadrilateral elements.

2.2. *Convex Four-Point Quadrilateral Elements*

We now consider that the domain Ω is divided into four-point convex quadrilaterals as the one shown in Fig. 3. The one-to-one transformation $\mathbf{F}_j : \hat{\Omega} \rightarrow \Omega_j$ is given by (see [10])

$$\begin{aligned} F_{1j}(p,q) &\equiv x = \sum_{i=1}^{4} x_i \Phi_i(p,q), \\ F_{2j}(p,q) &\equiv y = \sum_{i=1}^{4} y_i \Phi_i(p,q), \end{aligned} \tag{14}$$

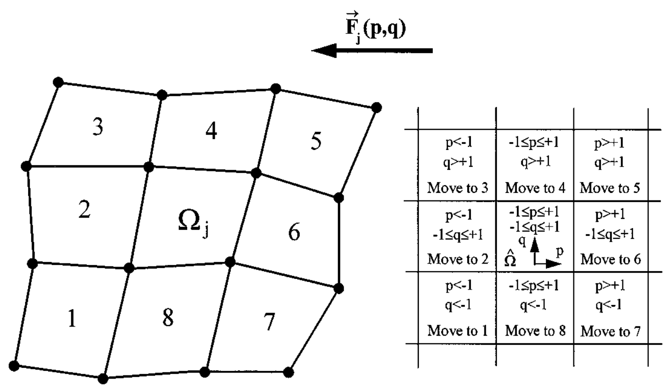
where the basis functions $\{\Phi_i\}_{i=1}^4$ are

$$\begin{aligned} \Phi_1(p,q) &= \tfrac{1}{4}(1-p)(1-q), \\ \Phi_2(p,q) &= \tfrac{1}{4}(1+p)(1-q), \\ \Phi_3(p,q) &= \tfrac{1}{4}(1+p)(1+q), \\ \Phi_4(p,q) &= \tfrac{1}{4}(1-p)(1+q), \end{aligned} \tag{15}$$

and $(x_i, y_i)_{i=1}^4$ denote the vertices of the convex quadrilateral Ω_j. Operating as we did with the linear triangle, we have the iterative process

- For $k = 0, 1, 2, \ldots$ set

$$\begin{Bmatrix} p^{k+1} \\ q^{k+1} \end{Bmatrix} = \begin{Bmatrix} p^{k} \\ q^{k} \end{Bmatrix} + \frac{1}{\Delta^k}\begin{bmatrix} b_2 + b_3 p^k & -a_2 - a_3 p^k \\ -b_1 - b_3 q^k & a_1 + a_3 q^k \end{bmatrix} \begin{Bmatrix} x_p - x_p^k \\ y_p - y_p^k \end{Bmatrix}, \tag{16}$$



$p<-1$ $q>+1$ Move to 3	$-1\le p\le+1$ $q>+1$ Move to 4	$p>+1$ $q>+1$ Move to 5
$p<-1$ $-1\le q\le+1$ Move to 2	$-1\le p\le+1$ $-1\le q\le+1$	$p>+1$ $-1\le q\le+1$ Move to 6
$p<-1$ $q<-1$ Move to 1	$-1\le p\le+1$ $q<-1$ Move to 8	$p>+1$ $q<-1$ Move to 7

FIG. 4. Search criterium for quadrilateral elements.

where

$$
\begin{aligned}
x_p^k &= \textstyle\sum_{i=1}^4 x_i\Phi_i(p^k,q^k), & y_p^k &= \textstyle\sum_{i=1}^4 y_i\Phi_i(p^k,q^k),\\
a_1 &= \tfrac{1}{4}(x_2-x_1+x_3-x_4), & b_1 &= \tfrac{1}{4}(y_2-y_1+y_3-y_4),\\
a_2 &= \tfrac{1}{4}(x_3-x_1+x_4-x_2), & b_2 &= \tfrac{1}{4}(y_3-y_1+y_4-y_2),\\
a_3 &= \tfrac{1}{4}(x_1-x_2+x_3-x_4), & b_3 &= \tfrac{1}{4}(y_1-y_2+y_3-y_4),\\
\Delta^k &= (a_1b_2-a_2b_1)+(a_1b_3-a_3b_1)\,p^k+(a_3b_2-a_2b_3)\,q^k.
\end{aligned}
$$

Note that for $k = 0$, it is computationally advantageous to take $(p^0, q^0) = (0, 0)$, the center of $\hat{\Omega}$. We notice, by inspection of (15), that x_p^k and y_p^k do not depend linearly upon p and q as in the linear triangle case. So if $\mathbf{x}_p \in \Omega_j$, it will generally be necessary more than one iteration to get $(\bar{p}, \bar{q}) \in \hat{\Omega}$, the image of $\mathbf{x}_p$. For convex four-point quadrilaterals Westermann [7] proposes an ad hoc iterative algorithm to compute $(\bar{p}, \bar{q})$. However, our iterative algorithm is more general because it can be applied to curved simplexes and quadrilaterals.

It remains to establish a selection criterion to know whether the iterates (p^k, q^k) obtained by (16) are in $\hat{\Omega}$ when $\mathbf{x}_p$ is in Ω_j. To do so, let us assume that the iterative scheme (16) is convergent. So, if $\mathbf{x}_p \in \Omega_j$ the iterates $(p^k, q^k) \in \hat{\Omega}$. Therefore, they satisfy the conditions

$$
\begin{aligned}
-1 \le p^k \le 1,\\
-1 \le q^k \le 1.
\end{aligned}
\tag{17}
$$

Then, if (17) does not hold, $\mathbf{x}_p \notin \Omega_j$ and the algorithm has to search a neighbour element to Ω_j. In order to decide on which neighbour to move to, we must take into account that if Ω_j is not a boundary element there are eight elements surrounding it (Fig. 4). We represent in this figure the neighbour selection criterion according to the values of (p, q). If one stores the eight neighbours of Ω_j in a two-dimensional integer array $IE(j, k = 1, ..., 8)$, then some of

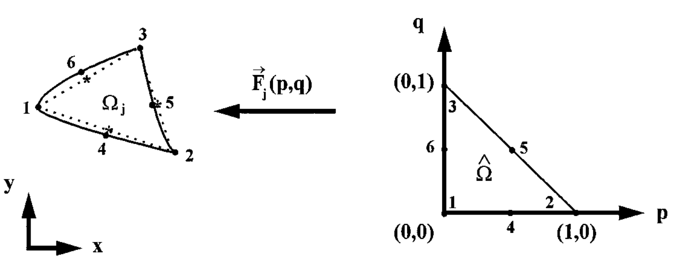


FIG. 5. Mapping for quadratic triangular elements.

the entries of such an array are zero when Ω_j is a boundary element. Hence, by checking $IE(j, k)$, one can easily know whether $\mathbf{x}_p$ is outside Ω. The selection criterion for convex four-point quadrilaterals is then

- Given the iterate (p^k, q^k), choose the neighbour element according to the directives given in Fig. 4.

Then, the search–locate algorithm for four-point convex quadrilaterals is

Search–Locate Algorithm for Linear Quadrilaterals (SALQ).

(S1) Given $\mathbf{x}_p$, pick any $\Omega_m \in \Omega$ and $\mathbf{p}^0 = (0, 0) \in \hat{\Omega}$ and build $\mathbf{F}_m(\mathbf{p}^0)$ using (14)–(15).

(S2) Use (16) to find the iterate (p^1, q^1).

(S3) Apply (17). If (17) holds, then GO TO **S2** and iterate until a given tolerance is attained. ELSE,

(S4) Apply selection criterium and GO TO **S1**.

2.3. *Curved Elements*

We next illustrate the application of our algorithm on grids composed of elements with curved sides (or faces in $\mathbf{R}^3$). The construction of a one-to-one mapping of $\hat{\Omega}$ onto a curved element Ω_j is less straightforward than in the two previous cases. We shall use the isoparametric element concept to define the one-to-one mapping $\mathbf{F}_j$ for curved elements. In this technique one defines the one-to-one mapping $\mathbf{F}_j$ on the reference element by a piecewise polynomial of order higher than one. Thus, with reference to Figs. 5 and 6, $\mathbf{F}_j$ is a quadratic polynomial for the triangle

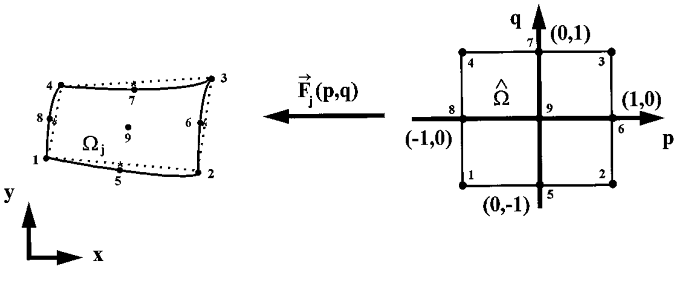


FIG. 6. Mapping for quadratic quadrilateral elements.

and a biquadratic polynomial for the quadrilateral. Following Ciarlet [10], we shall say that Ω_j is a regular isoparametric element if the following conditions are satisfied:

(i) Let h_j and ρ_j be defined as in the beginning of Section 2. Then there exists a positive constant $\sigma > 0$ such that $\rho_j/h_j \geq \sigma$.

(ii) h_j approximates to zero.

(iii) Let $h = \max(h_j)$. Then the distance between the midpoints $\bar{x}_i$ of the curved sides of Ω_i and the points $\bar{x}_i^*$ (see Figs. 5 and 6) is of order h^2. That is, $\|\bar{x}_i - \bar{x}_i^*\| = O(h^2)$.

The above conditions ensure the invertibility of the mapping $\mathbf{F}_j$. For further details on isoparametric elements of higher order and higher dimensions see Chapter IV of [10; 11]. For purposes of illustration and simplicity, we restrict the application of our algorithm to those elements shown in Figs. 5 and 6. These are commonly used in finite element codes for the solution of the Navier–Stokes equations.

2.3.1. *Quadratic Isoparametric Triangles.* The one-to-one mapping $\mathbf{F}_j : \hat{\Omega} \to \Omega_j$ (Fig. 5) is given as

$$\begin{aligned} F_{1j}(p,q) &\equiv x = \sum_{i=1}^{6} x_i \Phi_i(p,q), \\ F_{2j}(p,q) &\equiv y = \sum_{i=1}^{6} y_i \Phi_i(p,q), \end{aligned} \tag{18}$$

where $(x_i, y_i)_{i=1}^6$ are the coordinates of the nodes of Ω_j and the basis functions $\{\Phi_i\}_{i=1}^6$ are given by

$$\begin{aligned} \Phi_1(p,q) &= (1-2p-2q)(1-p-q), \\ \Phi_2(p,q) &= p(2p-1), \\ \Phi_3(p,q) &= q(2q-1), \\ \Phi_4(p,q) &= 4p(1-p-q), \\ \Phi_5(p,q) &= 4pq, \\ \Phi_6(p,q) &= 4q(1-p-q). \end{aligned} \tag{19}$$

Using (7) and (8), together with (18) and (19), yields the iterative procedure

- For $k = 0, 1, 2, \ldots$, set

$$\begin{Bmatrix} p^{k+1} \\ q^{k+1} \end{Bmatrix} = \begin{Bmatrix} p^{k} \\ q^{k} \end{Bmatrix} + \frac{1}{\Delta^k} \begin{bmatrix} d_1 + d_2 p^k + d_3 q^k & -b_1 - b_2 p^k - b_3 q^k \\ -c_1 - c_2 p^k - c_3 q^k & a_1 + a_2 p^k + a_3 q^k \end{bmatrix} \begin{Bmatrix} x_p - x_p^k \\ y_p - y_p^k \end{Bmatrix}, \tag{20}$$

where

$$\begin{aligned} a_1 &= 4x_4 - x_2 + 3x_1, & b_1 &= 4x_6 - x_3 - 3x_1, \\ a_2 &= 4(x_1 + x_2 - 2x_4), & b_2 &= 4(x_1 - x_4 + x_5 - x_6), \\ a_3 &= 4(x_1 - x_4 + x_5 - x_6), & b_3 &= 4(x_1 + x_3 - 2x_6), \end{aligned}$$

$$\Delta^k = (a_1 + a_2 p^k + a_3 q^k)(d_1 + d_2 p^k + d_3 q^k) - (b_1 + b_2 p^k + b_3 q^k)(c_1 + c_2 p^k + c_3 q^k).$$

Similar expressions for the c's and d's can be obtained by replacing x_i by y_i in each of the equations for the a's and b's. The neighbour criterion selection for this case is identical to that of linear triangles used in [8]. This is because $\hat{\Omega}$ for the present case is also defined by (3). We can now formulate a search–locate algorithm for quadratic curved triangles.

Search–Locate Algorithm for Quadratic Isoparametric Triangles (SAQT).

(S1) Given $\mathbf{x}_p$, pick any $\Omega_m \subset \Omega$ and any $\mathbf{p}^0 \in \hat{\Omega}$ to build $\mathbf{F}_m(\mathbf{p}^0)$ using (18)–(19).

(S2) Use (20) to find (p^1, q^1).

(S3) Apply (13). If (13) holds, then GO TO **S2** and iterate until a given tolerance is attained. ELSE,

(S4) Apply triangle selection criterium and GO TO **S1**.

Observe that SAQT differs from SALT in the expression for $\mathbf{F}_m$ and in **S3**. Also note that if isoparametric triangles of order higher than two are used, the expression for $\mathbf{F}_m$ in SAQT has to be changed.

2.3.2. *Quadratic Isoparametric Quadrilaterals.* The one-to-one mapping $\mathbf{F}_j : \hat{\Omega} \to \Omega_j$ (Fig. 6) is given as

$$\begin{aligned} F_{1j}(p,q) &\equiv x = \sum_{i=1}^{9} x_i \Phi_i(p,q), \\ F_{2j}(p,q) &\equiv y = \sum_{i=1}^{9} y_i \Phi_i(p,q), \end{aligned} \tag{21}$$

where $(x_i, y_i)_{i=1}^9$ are the coordinates of the nodes of Ω_j and the basis functions $\{\Phi_i\}_{i=1}^9$ are given by

$$\begin{aligned} \Phi_1(p,q) &= \tfrac{1}{4} pq(1-p)(1-q), \\ \Phi_2(p,q) &= \tfrac{1}{4} pq(1+p)(q-1), \\ \Phi_3(p,q) &= \tfrac{1}{4} pq(1+p)(1+q), \\ \Phi_4(p,q) &= \tfrac{1}{4} pq(p-1)(1+q), \\ \Phi_5(p,q) &= \tfrac{1}{2} q(1-p^2)(q-1), \end{aligned} \tag{22}$$

$$\Phi_6(p,q) = \tfrac{1}{2}p(1+p)(1-q^2),$$
$$\Phi_7(p,q) = \tfrac{1}{2}q(1-p^2)(q+1),$$
$$\Phi_8(p,q) = \tfrac{1}{2}p(p-1)(1-q^2),$$
$$\Phi_9(p,q) = (1-q^2)(1-p^2).$$

Substituting (22) and (21) in (7) and (8) yields the iterative procedure

- For $k = 0, 1, 2, ...$, set

$$\begin{Bmatrix} p^{k+1} \\ q^{k+1} \end{Bmatrix} = \begin{Bmatrix} p^{k} \\ q^{k} \end{Bmatrix} + \frac{1}{\Delta^k}\begin{bmatrix} a(p^k,q^k) & b(p^k,q^k) \\ c(p^k,q^k) & d(p^k,q^k) \end{bmatrix}\begin{Bmatrix} x_p - x_p^k \\ y_p - y_p^k \end{Bmatrix}, \tag{23}$$

where the entries of the 2×2 matrix are in this case given by

$$a(p,q) = a_0 + a_1 p + a_2 q + a_3 pq + a_4 p^2 + a_5 qp^2$$
$$b(p,q) = -(b_0 + b_1 p + b_2 q + b_3 pq + b_4 p^2 + b_5 qp^2)$$
$$c(p,q) = -(c_0 + c_1 p + c_2 q + c_3 pq + c_4 q^2 + c_5 pq^2)$$
$$d(p,q) = d_0 + d_1 p + d_2 q + d_3 pq + d_4 q^2 + d_5 pq^2$$
$$\Delta^k = a(p^k,q^k)d(p^k,q^k) - c(p^k,q^k)b(p^k,q^k).$$

The coefficients a_i, b_i, c_i, and d_i are expressed in terms of the coordinates of the nodes of Ω_j as

$$a_0 = \tfrac{1}{2}(y_7 - y_5),$$
$$a_1 = \tfrac{1}{2}(\overline{y}_{13} - \overline{y}_{24}),$$
$$a_2 = 2(\overline{y}_{57} - y_9),$$
$$a_3 = \overline{y}_{23} - \overline{y}_{14} + y_8 - y_6,$$
$$a_4 = \tfrac{1}{2}[(\overline{y}_{34} - y_7) - (\overline{y}_{12} - y_5)],$$
$$a_5 = \tfrac{1}{2}\textstyle\sum_{i=1}^{4} y_i - \sum_{i=5}^{8} y_i + 2y_9.$$

where $\overline{y}_{ij} = \frac{1}{2}(y_i + y_j)$. Equivalent expressions are obtained for the b_i's by replacing the y_i's by the x_i's. Similarly,

$$c_0 = \tfrac{1}{2}(y_6 - y_8),$$
$$c_1 = 2(\overline{y}_{68} - y_9),$$
$$c_2 = \tfrac{1}{2}(\overline{y}_{13} - \overline{y}_{24}),$$
$$c_3 = \overline{y}_{34} - \overline{y}_{12} + y_5 - y_7,$$
$$c_4 = \tfrac{1}{2}[(\overline{y}_{23} - y_6) - (\overline{y}_{14} - y_8)],$$
$$c_5 = \tfrac{1}{2}\textstyle\sum_{i=1}^{4} y_i - \sum_{i=5}^{8} y_i + 2y_9.$$

As before, expressions for the d_i's are obtained by replacing the y_i's by the x_i's.

The neighbour criterion selection for curved quadrilaterals is the same as for the straight side quadrilaterals because $\hat{\Omega}$ for this case is also defined by (3). A step by step formulation of our search–locate algorithm for quadratic isoparametric quadrilaterals is as follows.

Search–Locate Algorithm for Quadratic Isoparametric Quadrilaterals (SAQQ).

(S1) Given $\mathbf{x}_p$, pick any $\Omega_m \in \Omega$ and $\mathbf{p}^0 = (0, 0) \in \hat{\Omega}$ and build $\mathbf{F}_m(p^0)$ using expressions (21)–(22).

(S2) Use (23) to find the iterate (p^1, q^1).

(S3) Apply (17). If (17) holds, then GO TO **S2** and iterate until a given tolerance is attained. ELSE,

(S4) Apply selection criterium and GO TO **S1**.

Algorithm SAQQ can be used for higher order isoparametric quadrilaterals by changing the formulation of the mapping $\mathbf{F}_m$ in (21).

3. CONVERGENCE

We are now going to prove that the iterative procedure of our algorithm converges to a point $\overline{\mathbf{p}} \in \hat{\Omega}$ such that whenever $\mathbf{x}_p \in \Omega_j$, $\mathbf{F}_j(\overline{\mathbf{p}}) = \mathbf{x}_p$. Our strategy to prove convergence is based on results of Chapter X of [12].

Let us assume that after a finite number of searches, using the corresponding neighbour selection criterion, the algorithm has reached the right element Ω_j. Then we have to prove that starting with any $\mathbf{p}_0 \in \hat{\Omega}$ the iterates $\{\overline{\mathbf{p}}^k\}$, $k = 1, 2, ...$, converge to $\overline{\mathbf{p}}$. Thus,

$$\lim_{k\to\infty} \mathbf{p}^k = \overline{\mathbf{p}}.$$

For simplicity, let us write the iterations of the algorithm as

$$\mathbf{p}^{k+1} = \mathbf{T}_j \mathbf{p}^k, \quad k = 0, 1, 2, ..., \tag{24}$$

for $\hat{\Omega}$ included in a bounded domain D and any Ω_j, $\mathbf{T}_j : D \subset R^d$ ($d = 2$ or 3) is

$$\mathbf{T}_j \mathbf{p}^k = \mathbf{p}^k + J_j^{-1}(\mathbf{p}^k)(\mathbf{x}_p - F_j(\mathbf{p}^k)), \tag{25}$$

where J_j^{-1} is the inverse of the Jacobian matrix of $\mathbf{F}_j(\mathbf{p}^k)$. The point $\overline{\mathbf{p}} \in \hat{\Omega}$ is a point of attraction of (25) if there is an open neighbourhood U of $\overline{\mathbf{p}}$ such that U is included in D, and for any $\mathbf{p}^0 \in U$ the iterates $\{\mathbf{p}^k\}$ all lie in U and converge to $\overline{\mathbf{p}}$. We notice that by construction the following properties hold:

(i) $\mathbf{F}_j : \hat{\Omega} \to \Omega_j$ is continuous with first and second continuous derivatives on $\hat{\Omega}$.

(ii) The Jacobian matrix J_j is nonsingular on $\hat{\Omega}$.

(iii) $\mathbf{F}_j(\overline{\mathbf{p}}) - \mathbf{x}_p = 0$.

(iv) For any vector $\mathbf{h} \in \mathbf{R}^d$, $\mathbf{h} \neq 0$, $D^2\mathbf{F}_j(\overline{\mathbf{p}})\ \mathbf{hh} \neq 0$, where D^2 denotes second derivatives.

Then, the Newton attraction theorem (see Chapter X of [12]) guarantees that $\overline{\mathbf{p}}$ is a point of attraction of (24). Furthermore, there exists a constant K such that

$$|\mathbf{p}^{k+1} - \overline{\mathbf{p}}| \leq K\,|\mathbf{p}^k - \overline{\mathbf{p}}|^2 \quad \text{for any } k = 0, 1, 2, ...,$$

where $|\cdot|$ denotes maximum norm. This bound expresses the well-known quadratic convergence of the Newton method.

Remarks. (i) The first conclusion to be drawn from the convergence proof is that, for any initial guess element Ω_m, the algorithm will end up in the right element Ω_j. This is so, because if $\mathbf{x}_p$ is not in Ω_m, then there is no $\overline{\mathbf{p}}$ in $\hat{\Omega}$ such that the condition $\mathbf{F}_m(\overline{\mathbf{p}}) - \mathbf{x}_p = 0$ holds. Hence, $\overline{\mathbf{p}}$ is not a point of attraction of the iterative procedure and, therefore, the iterates $\mathbf{p}^k$ do not lie in $\hat{\Omega}$. So the algorithm will apply the proper neighbour selection criterion to move to another element.

(ii) The property that the Jacobian matrix J_j is not singular is a crucial point because it guarantees that our algorithm will never break down.

(iii) It is important to remark that our algorithm finds not only the element where $\mathbf{x}_p$ lies, but it also gives the point $\overline{\mathbf{p}}$ which is the image of $\mathbf{x}_p$ in $\hat{\Omega}$. This is an important feature, particularly for curved elements and bilinear quadrilaterals because, as finite element codes do, it is far more convenient to interpolate on the reference element $\hat{\Omega}$ than on the mesh in the physical domain.

(iv) Once the element hosting $\mathbf{x}_p$ has been identified, the number of iterations k required by the Newton method to obtain converged values (p, q) of the master element is certainly a function of the nonuniformity of the grid. However, if the grid varies progressively according to the hypotheses established in Section 2 and if using curved elements the conditions of isoparametric elements are satisfied, then the interval of variation of k over all the elements in the grid is small. Note that one should avoid highly distorted elements or elements with very small angles that produce very small values of the Jacobian of the transformation.

4. NUMERICAL RESULTS

In this section, numerical results are presented corresponding to simply connected domains using linear and quadratic triangular elements. For quadrilateral elements, searches are performed in a doubly connected domain containing a NACA 0012 airfoil at an angle of attack of 20°. In all simulations, the spatial position of the particle is entered as a known value. The search is commenced at an element that is specified a priori.

Figure 7 shows the search of a particle located on the bottom boundary of the triangular domain; the inset shows its position. The search was carried out using the criterium of Löhner and Ambrosiano [8]. Both our iterative localization scheme and the one in [8] were used to conduct this test. As expected, both methods give identical results for the particular case of a straight boundary. However, for a domain with curved elements, such as that shown in Fig. 8, this is not the case. It is evident from this figure that

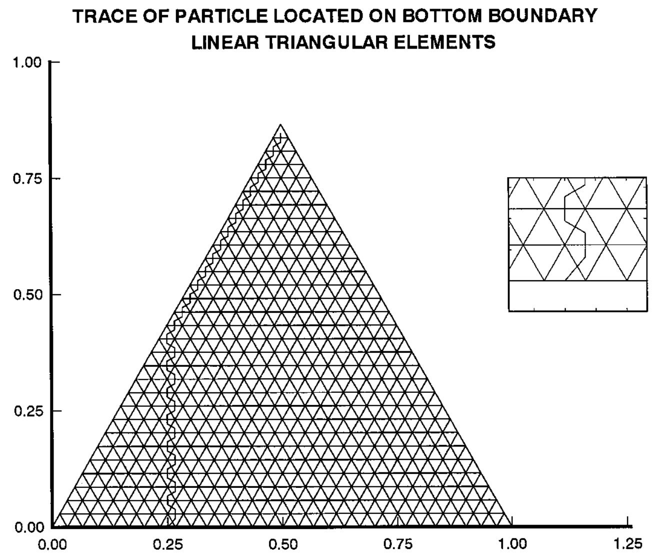

TRACE OF PARTICLE LOCATED ON BOTTOM BOUNDARY
LINEAR TRIANGULAR ELEMENTS
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25

FIG. 7. Particle trace using linear triangular elements.

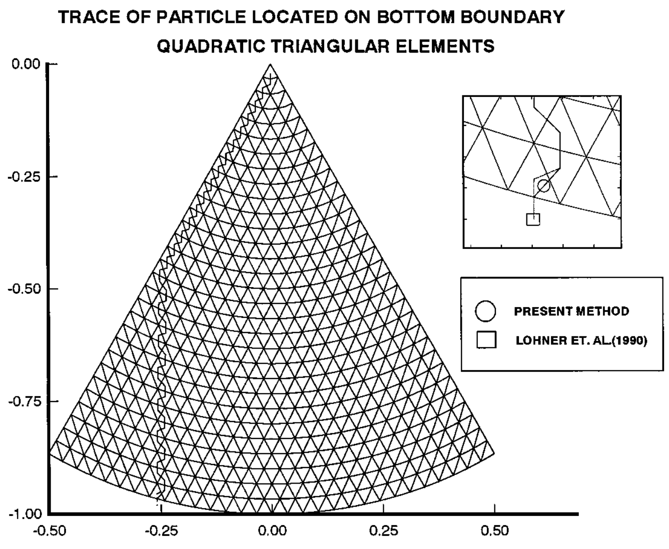


FIG. 8. Particle trace using quadratic triangular elements.

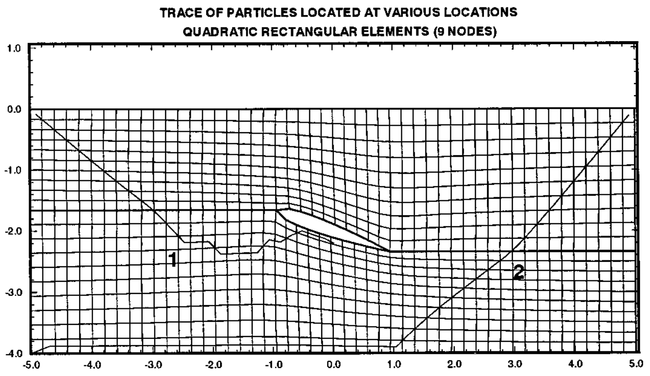

TRACE OF PARTICLES LOCATED AT VARIOUS LOCATIONS
QUADRATIC RECTANGULAR ELEMENTS (9 NODES)
1
2

FIG. 9. Particle trace using quadratic quadrilateral elements.

only our method is able to locate the particle positioned on the circular portion of the boundary.

Figure 9 shows two different searches on a structured grid generated by the finite element solution of an elliptic system [13]. The particle host elements are located at two different locations in the doubly connected domain. The first case, the search labelled as 1, the particle is located approximately at $(x, y) = (0, -2.2)$. The search is commenced from the element on the top left corner. Clearly, the particle is efficiently located from starting points located on opposite sides of the slit in the domain.

The search labelled as 2, is a more stringent test. The particle is located at $(x, y) = (-5.0, -4.0)$, the bottom left corner of the domain. The search is started from an element located on the corner diagonally opposite to the particle host element. The particle is found after checking only 50 elements in a domain of 1000 elements. Figure 10 shows

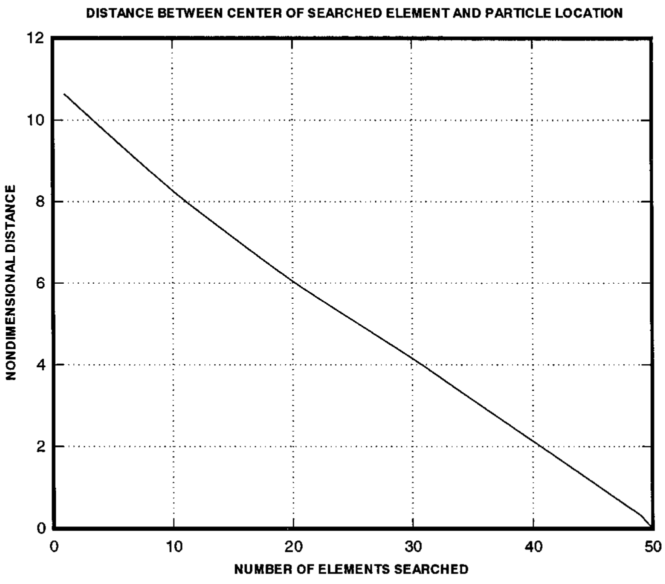


FIG. 10. Nondimensional distance between searched element and particle location for trace 2 in Fig. 9.

TABLE I

Performance of SALT, SAQT and SAQQ Algorithms

Case	Total CPU (seconds)	No. elements searched	CPU/element (seconds)
SALQ	27/100	59	0.457/100
SAQT	28/100	59	0.474/100
SAQQ	33/100	50	0.660/100

the distance measured as a straight line from the center of the element that is searched and the actual position of the fluid particle. It is apparent from this figure that the search is conducted in an element located progressively closer to the particle location.

At this point we would like to remark that it is possible that, when searching over long spatial distances in multiply connected domains, a particle search can experience a phenomenon we have termed "locking." By this we mean that the search process is carried out between the same two, or more, elements without progressing on its path to the target particle. This inconvenience can be easily avoided by establishing an appropriate logic that checks for this type of situations. Once the elements involved in the locking process have been identified, a simple solution is to "bump" the search to an element outside their sphere of action and proceed normally with the trace.

Finally, we look at the run-time performance of the searches conducted in Figs. 7, 8, and 9. These have been summarized in Table I. Linear triangles are marginally faster than quadratic ones, and triangles conduct the search more rapidly than quadrilaterals. Table II shows run-time comparisons using various initial conditions x_{in}, y_{in} for searching the target particle located at $(x, y) = (-5.0, -4.0)$ in Fig. 9. We notice that the CPU depends on the number of elements searched to find the host element rather than the initial distance from the target particle. Note that CPU times given in Tables I and II include various output instructions and should therefore be taken as reference values only. All computations were carried

TABLE II

Performance of SAQQ Algorithm

$x_{\text{in}}, y_{\text{in}}$	Initial distance from target	No. elements searched	CPU (seconds)
(4.70, −3.64)	0.47	2	<1/100
(−3.07, −1.77)	2.95	10	6/100
(−1.44, −.10)	5.28	22	11/100
(3.07, −2.01)	8.31	41	27/100
(4.89, −0.10)	10.63	50	33/100

out with a PC machine equipped with a Pentium 120 processor and Windows for Workgroups operating system.

It can be seen from the above examples, that our algorithm finds the element containing the particle even if this is several cells away, thus posing no restrictions of this nature. Furthermore, only a low number of elements (close to optimal) with respect to the total number of elements in the grid needs to be checked in the search process. As a final word, for all simulations conducted in this work, once the host element was found no more than 3 iterations were necessary to find converged values of (p, q) using a tolerance of 10^{-6}.

5. CONCLUSIONS

A generalized algorithm for search and location of points in arbitrary grids has been described. Such algorithm is a two stage procedure that combines the Newton method, to invert a family of one-to-one maps $\mathbf{F}_j$, with a neighbour selection criterion to search the elements where the points are located. Under some general conditions to be imposed on the shape of the mesh-elements, the algorithm can be applied to any type of mesh. Hence, it is useful for those methods that have moving grids and also for methods that need to transfer information among different meshes. Numerical examples show that the number of elements checked in the search path is very low, in comparison to the total number of elements in the grid. Based on this work's simulations, accurate localization within the host element can be achieved in no more than three iterations.

ACKNOWLEDGEMENTS

Dr. A. Allievi gratefully acknowledges the Industrial Research Fellowship provided by the Natural Sciences and Engineering Research Council of Canada. The facilities and assistance provided by Martec Limited while part of this work was conducted are very much appreciated. Dr. R. Bermejo was supported by grant AMB93-0199 from Comisión Interministerial de Ciencia y Tecnología de España. Finally, the authors thank the referees for their comments to clarify and improve the original manuscript.

REFERENCES

1. R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles* (Hilger, Bristol, 1988).
2. R. Bermejo, *SIAM J. Numer. Anal.* **32,** 425 (1995).
3. A. Staniforth and J. Coté, *Mon. Weather Rev.* **119,** 2206 (1991).
4. O. Pironneau, *Numer. Math.* **38,** 309 (1982).
5. J. Douglas and T. F. Russell, *SIAM J. Numer. Anal.* **19,** 871 (1982).
6. J. U. Brackbill and H. M. Ruppert, *J. Comput. Phys.* **65,** 314 (1986).
7. T. Westermann, *J. Comput. Phys.* **901,** 380 (1992).
8. R. Löhner and J. Ambrosiano, *J. Comput. Phys.* **91,** 22 (1990).
9. R. Löhner, *J. Comput. Phys.* **118,** 380 (1995).
10. P. Ciarlet, *The Finite Element Method for Elliptic Problems* (North-Holland, Amsterdam, 1978).
11. M. Lenoir, *SIAM J. Numer. Anal.* **23,** 563 (1986).
12. J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables* (Academic Press, New York/London, 1970).
13. A. Allievi and S. M. Calisal, *J. Comput. Phys.* **98,** 163 (1992).