

Technical Note

A method for particle location and field interpolation on complex, three-dimensional computational meshes

P. J. Oliveira^{a,*}, A. D. Gosman^b & R. I. Issa^b

^a*Departamento de Eng. Electromecânica, Universidade da Beira Interior, R. Marquês d'Ávila e Bolama, 6200 Covilhã, Portugal*

^b*Department of Mechanical Engineering, Imperial College of Science, Technology and Medicine, London, UK*

(Received 24 July 1996; accepted 24 July 1997)

A method for locating particles within arbitrary three-dimensional computational meshes is described. It is based on an iterative procedure which uses transformed coordinates defined by iso-parametric functions. The method also enables one to interpolate field values from the mesh nodes to the particle position. Example applications demonstrate how effective the method is. For very distorted computational cells special practices have to be introduced in order to keep the number of iterations to a minimum. © 1997 Elsevier Science Limited.

Key words: particle location, location, interpolation, computational mesh.

1 INTRODUCTION

In computational fluid dynamics it is often necessary to locate and track particles as they move in space. The following applications are examples of situations where such procedure is required:

- Lagrangian two-phase flow calculations, where a dispersed cloud of liquid drops or solid particles interacts with a continuous surrounding phase: e.g. modelling of sprays, particle laden jets, etc.
- Generation of streamlines, i.e. fluid-particle trajectories in three-dimensional flows to enable visualisation.
- Generation of profiles of variables along an arbitrarily defined line in a complex three-dimensional mesh by interpolation from nodal values.

The starting and common point to all these computations is the determination of the cell in the computational mesh in which the particle in question is located, and the objective of this paper is to address this problem by proposing a consistent mean of interpolating field values within arbitrary

computational meshes. To this end, a novel method is developed which, given the particle position $\mathbf{P} \equiv (x_p, y_p, z_p)$, and a cell in a computational mesh, can be used to verify whether or not the particle lies inside the cell and to interpolate any given variable from the cell vertices to the particle point. The computational cells in question are hexahedral in shape, but may otherwise be quite general: three-dimensional, of the finite-volume or finite-element type, non-orthogonal, fixed or moving, or part of an unstructured mesh. The only requirement here to define a cell is knowledge of the coordinates of its eight vertices, $(x, y, z)_n$, $n = 1-8$. Cells where some vertices coincide with each other, e.g. tetrahedra, can also be dealt with.

Procedures for locating particles within two- or three-dimensional Cartesian meshes are straightforward, since the cells are delineated by planes normal to coordinate directions, thus enabling a separate search along each coordinate direction. Some curvilinear orthogonal coordinates are also easy to deal with, such as cylindrical and spherical coordinates.

However, for arbitrary meshes this simplicity does not obtain, even for two-dimensional cases as Fig. 1 illustrates. It is now necessary to determine the location of the particle relative to all four edges to find out whether it falls inside the cell. This is usually performed using geometrical reasoning

*Author to whom all correspondence should be addressed.

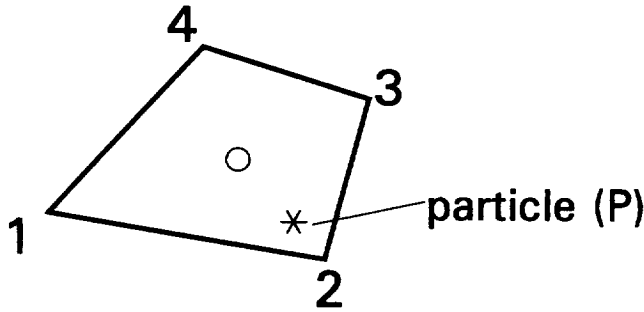


Fig. 1. Illustration of a two-dimensional non-orthogonal cell.

(for example, for the case of Fig. 1 it is necessary to check that all cross-products $\overline{12} \times \overline{1P}$, $\overline{23} \times \overline{2P}$, etc. are positive). This becomes very cumbersome in three dimensions, in which case a cell face may not necessarily be a plane, thus complicating the geometrical approach. Nonetheless, most previous work is based on different forms of this geometrical approach¹⁻³ and other geometrical algorithms are reviewed in Ref. 3. A different approach is taken by Seldner and Westermann⁴ who located the particles on a finely meshed rectangular grid superimposed over the general non-orthogonal one. This method requires a preparatory stage where the nodes of the general mesh are first located on the rectangular mesh using any of the mentioned geometry-based algorithms. Interest was focussed on the development of a fast algorithm for particle-location and field interpolation with the capability of vectorization; it is however applicable only to two-dimensional meshes and is impractical in the case of coupled lagrangian/eulerian calculations, when particles influence the flow as they move every time-step. The only work reporting three-dimensional particle location is by Amsden *et al.*,⁵ who follow the geometrical approach.

Finding the particle location solves only half of the problem; it still leaves the task of field variable interpolation from mesh values to the particle location. This is usually based on interpolation methods for non-grided data,⁶⁻⁸ where nodal values are fitted by an assumed variation. For example in two dimensions the surface:

$$\phi = A + Bx + Cy + Dxy$$

may be used to express the variations of property ϕ in space. The coefficients A – D are determined from the known ϕ s at the four nodes ($n = 1, 4$):

$$\phi_n = A + Bx_n + Cy_n + Dx_ny_n \quad (n = 1-4)$$

which entails the inversion of a 4×4 matrix:

$$\{\phi\} = [A_{xy}]\{\text{Coef}\} \Rightarrow \{\text{Coef}\} = [A_{xy}^{-1}]\{\phi\} \quad (1)$$

In three dimensions these matrices become 8×8 (or 7×7 if the origin is placed at one of the cell vertices), and have to be inverted for each cell in the mesh. Since the system shown in eqn (1) is usually solved by L–U decomposition instead of actually inverting matrix A , it means that every field variable to be interpolated will require its own 8×8 matrix inversion. This can be an

expensive task as present-day practical computational fluid dynamics (CFD) applications involve of the order of 200 000–500 000 cells.

Seldner and Westermann, following the localization procedure referred to earlier, use an area-weighting interpolation which is somewhat similar to the approach used here when applied to the two-dimensional case.

The method developed here solves both problems: those of locating the particle and of field interpolation, simultaneously. The original concept⁹ is based on isoparametric interpolation functions, which transform each cell in the physical space into a well-behaved cubic unit cell in transformed space. If the transformed coordinates, $(\xi_l)_P \equiv (\xi, \eta, s)_P$ for $l = 1-3$, of a particle can be determined, then it is easy to check whether the particle is inside the cell (if $-1 \leq \xi_l \leq +1$, $l = 1-3$), and to obtain interpolated values. The way the transformed point $(\xi_l)_P$ is obtained together with other details of the method are given below; this is followed by an assessment of its performance in sample applications; and finally a proposal for improvement when the cells are very distorted is presented.

2 THE PRESENT METHOD

The tri-linear isoparametric functions¹⁰ applied to a six-faced cell with the local node-indexing given in Fig. 2 are

$$\mathcal{N}_n(\xi, \eta, s) = \frac{1}{8}(1 + \xi_n \xi)(1 + \eta_n \eta)(1 + s_n s) \quad (2)$$

where ξ_n , η_n and s_n take the values ± 1 corresponding to the node in question ($n = 1-8$). Cells are not restricted to be six-faced, and collapsing shapes are allowed. Any variable $\phi(x, y, z)$, with known values at the eight vertices (ϕ_n , $n = 1-8$), can be expressed at any other point as

$$\phi(x, y, z) = \bar{\phi}(\xi, \eta, s) = \sum_{n=1}^8 \mathcal{N}_n(\xi, \eta, s) \cdot \phi_n \quad (3)$$

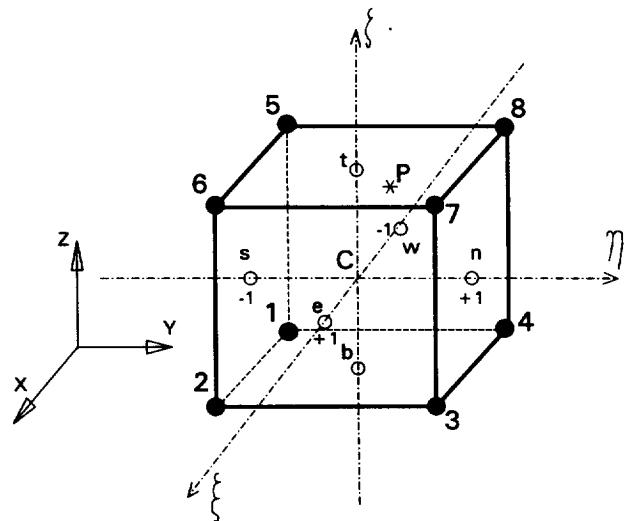


Fig. 2. Cell in the transformed space.

where $\bar{\phi}$ represents the same variable in the transformed space. Similar expressions hold for the Cartesian coordinates of any point P :

$$\begin{aligned} x_P &= \sum_{n=1}^8 \mathcal{N}_n(\xi_P, \eta_P, s_P) \cdot x_n \\ y_P &= \sum_{n=1}^8 \mathcal{N}_n(\xi_P, \eta_P, s_P) \cdot y_n \\ z_P &= \sum_{n=1}^8 \mathcal{N}_n(\xi_P, \eta_P, s_P) \cdot z_n \end{aligned} \quad (4)$$

Hence, given the physical coordinates of a particle, $\mathbf{P} = (x_P, y_P, z_P)$, eqn (4) may be inverted to obtain the transformed coordinates, $\xi_P = (\xi_P, \eta_P, s_P)$; eqn (3) may then be applied for interpolation purposes. Therefore, knowledge of ξ_P is the key point to fulfil the following objectives:

- location—the particle is inside the cell if, and only if, $-1 \leq (\xi, \eta, s)_P \leq +1$;
- interpolation—eqn (3) can be applied readily to find the required value at the point in question;
- search for new position—if the particle is outside the cell, its relative position is roughly indicated by $(\xi, \eta, s)_P$; for example if $\xi = 1.5$ and both η and s are less than ± 1 , then the particle is probably in the adjacent cell in the $+\xi$ direction. The task therefore is to devise a method to solve eqn (4), in order to obtain ξ_P . The proposed iterative method is described below.

2.1 Iterative solution

The iterative method is based on the following algorithm:

1. Guess the transformed coordinates of the point, $\xi^* = (\xi_P^*, \eta_P^*, s_P^*)$
2. use eqn (4) to obtain the corresponding physical point, (x_P^*, y_P^*, z_P^*) ;
3. calculate the distance between this guess and the particle position, $d = ((x_P^* - x_P)^2 + (y_P^* - y_P)^2 + (z_P^* - z_P)^2)^{1/2}$;
4. if this distance is small, convergence is achieved;
5. otherwise, use this distance to obtain a new guess and return to step 1.

The key step is 5, i.e. how to make a good guess for the new iteration. Analysis of this question revealed that the best option is to approximate the position of the point in the transformed space by the normalized contravariant coordinates of the physical point, when referenced to the local covariant unit basis. This basis is formed by vectors joining centres of opposite cell faces, see Fig. 2. The normalization is made by dividing the coordinates by the half-length of these spanning vectors. From Fig. 2 the spanning vectors are defined as:

$$\text{Spanning vectors} \equiv \overline{\mathbf{w}\mathbf{e}}, \overline{\mathbf{s}\mathbf{n}}, \overline{\mathbf{b}\mathbf{t}} \quad (5)$$

where points at the centre of each cell face are calculated as (west face, for example)

$$\mathbf{w} = \frac{1}{4}(\mathbf{X}_1 + \mathbf{X}_4 + \mathbf{X}_5 + \mathbf{X}_8) \quad (6)$$

where \mathbf{X}_n is the vector defining vertex n . The centroid (\mathbf{C}) of the cell is defined by

$$\mathbf{C} = \frac{1}{8} \sum_{n=1}^8 \mathbf{X}_n \quad (7)$$

and the local covariant unit basis, with origin at \mathbf{C} , is defined by

$$\mathbf{e}_1 = \frac{\overline{\mathbf{w}\mathbf{e}}}{\|\overline{\mathbf{w}\mathbf{e}}\|}, \mathbf{e}_2 = \frac{\overline{\mathbf{s}\mathbf{n}}}{\|\overline{\mathbf{s}\mathbf{n}}\|} \text{ and } \mathbf{e}_3 = \frac{\overline{\mathbf{b}\mathbf{t}}}{\|\overline{\mathbf{b}\mathbf{t}}\|} \quad (8)$$

where $\|\cdot\|$ denotes the Euclidian norm.

It is easy to demonstrate that if the centroids of faces are defined by eqn (6), then the centroid of the cell (point \mathbf{C}) is in the middle of any of the three spanning vectors [i.e. $\mathbf{C} = 1/2(\mathbf{e} + \mathbf{w}) = 1/2(\mathbf{s} + \mathbf{n}) = 1/2(\mathbf{b} + \mathbf{t})$]. The normalizing distances for each direction (ξ, η, s) can therefore be defined by

$$l_\xi \equiv \|\mathbf{e} - \mathbf{C}\|, l_\eta \equiv \|\mathbf{n} - \mathbf{C}\| \text{ and } l_s \equiv \|\mathbf{t} - \mathbf{C}\| \quad (9)$$

To obtain the contravariant coordinates of the particle point it is necessary to compute and invert the change-of-basis matrix. The usual Cartesian basis, denoted by $\mathbf{i}_j \equiv (\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}})$, is related to the covariant basis by

$$\mathbf{e}_i = a_{ij} \mathbf{i}_j$$

or

$$\{\mathbf{e}\} = [\mathbf{A}]\{\mathbf{i}\}$$

where the elements of matrix $[\mathbf{A}]$ are obtained from eqn (8). The coordinates of point \mathbf{P} in the new covariant basis are denoted by \bar{x}^i , and can be determined from

$$\mathbf{P} = x^k \mathbf{i}_k = \bar{x}^l \mathbf{e}_l = \bar{x}^l a_{lj} \mathbf{i}_j \Rightarrow x^k = \bar{x}^l a_{lk} \Rightarrow \bar{x}^j = x^k a_{kj}^{-1}$$

This can be expressed in matrix form as

$$\{\bar{\mathbf{x}}\} = [\mathbf{A}^{-1}]^T \{\mathbf{x}\} \quad (10)$$

and the approximation to (ξ, η, s) is given by the components of vector $\bar{\mathbf{x}}$ after normalizing by the spanning distances [eqn (9)], to yield

$$\xi^* = \frac{\bar{x}}{l_\xi}, \eta^* = \frac{\bar{y}}{l_\eta} \text{ and } s^* = \frac{\bar{z}}{l_s} \quad (11)$$

The algorithm used in the actual computations differs from the description above in one minor point: the normalization in eqns (8) and (11) is not required if the spanning vectors used in eqn (8) to define matrix $[\mathbf{A}]$ are computed by starting from point \mathbf{C} . This point is better clarified by the new definition of matrix $[\mathbf{A}]$ given in step 2 of the algorithm description that follows, and the consequence is that less computational work is required per iteration.

2.2 Overall algorithm

For a given particle position (**P**), and a given cell defined by its eight vertices, the following steps are performed:

- (1) Compute central points of east, north and top faces (**e**, **n**, and **t** in Fig. 2) from eqn (6), and cell centroid **C** from eqn (7).
- (2) Determine matrix of change-of-basis [**A**], whose rows are formed from the Cartesian components of the following three vectors:

$$\mathbf{A}_1 = \mathbf{e} - \mathbf{C}, \quad \mathbf{A}_2 = \mathbf{n} - \mathbf{C} \text{ and } \mathbf{A}_3 = \mathbf{t} - \mathbf{C}, \text{ that is}$$

$$A_{lj} = (X_j)_{\text{face } l} - C_j$$

- (3) Calculate distance from centroid to particle position:

$$\mathbf{d} \equiv \mathbf{P} - \mathbf{C}, \text{ and } d \equiv \|\mathbf{d}\| = \text{dist}(\mathbf{C}, \mathbf{P})$$

$$= ((x_C - x_P)^2 + (y_C - y_P)^2 + (z_C - z_P)^2)^{1/2}$$

If this distance is less than the given tolerance, there is no need to proceed with the iteration process below, and calculations are terminated at this stage.

- (4) Invert and transpose the 3×3 -matrix [**A**] $\Rightarrow [\mathbf{A}^{-1}]^T$.

- (5) Find first guess, defined by components of vector **d** in the new basis {**e**}.

$$\{\xi^*\} = [\mathbf{A}^{-1}]^T \{\mathbf{d}\}$$

- (6) Determine isoparametric functions for point ξ^* [eqn (2)].

- (7) Find new approximation to particle position, from eqn (4) $\Rightarrow \mathbf{P}^*$.

- (8) Compute error, defined by distance between **P** and **P**^{*}.

$$\mathbf{d}' \equiv \mathbf{P} - \mathbf{P}^*$$

$$d' \equiv \text{dist}(\mathbf{P}, \mathbf{P}^*) = \|\mathbf{d}'\|$$

- (9) Estimate new guess if d' is not small, from:

$$\{\xi^*\} = \{\xi^*\} + \{\delta\xi^*\}, \text{ with } \{\delta\xi^*\} = [\mathbf{A}^{-1}]^T \{\mathbf{d}'\}$$

and return to step (6); otherwise stop (convergence achieved).

Note that steps (1)–(5) in the algorithm above are necessary only for the initial guess. After iteration 1, the algorithm proceeds through steps (6)–(9) only.

3 APPLICATION AND IMPROVEMENT

The algorithm described has been implemented and tested for a number of cell shapes and particle positions. Depending on the skewness of the cell, the number of iterations required to achieve a convergence tolerance of 1% varied from 1 to 5.

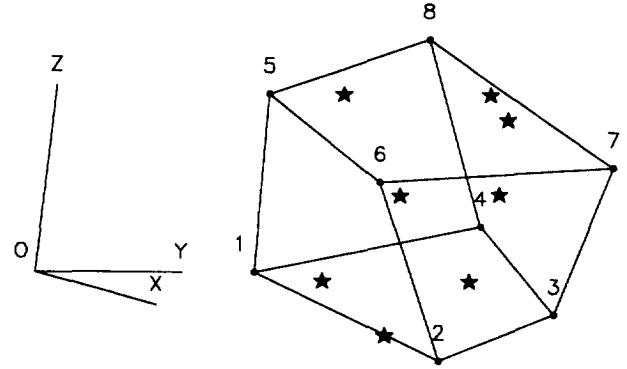


Fig. 3. Example of a three-dimensional cell with several particle points.

For Cartesian meshes the initial guess gives the solution immediately. For slightly skewed cells, two to three iterations are typical. Some strongly skewed cells, such as triangular prisms, will require a special treatment in order to achieve fast convergence; this will be explained in the next section.

Convergence was measured by d' , the normalized distance between the guessed and actual particle position; it was always monotonic. Divergence only occurred when the particle fell outside of, and far away from, the cell being checked. Divergence was also found to be monotonic and rapid; therefore after a few iterations (say five) one can safely decide to stop the iterative process if d' continually increases, and look for the particle in a neighbouring cell.

A sample of results for the cell shown in Fig. 3 is listed in Table 1. Convergence is assumed when the distance between the guessed and actual position, normalized by a characteristic cell dimension [defined as $(\text{cell volume})^{1/3}$], falls below a tolerance of 10^{-2} .

In order to demonstrate that the present method when used for interpolation alone is competitive with that based on inversion of an 8×8 matrix, a comparison has been made of CPU time required to compute 10^5 interpolations.

The coordinates of the eight cell vertices and the corresponding values of the variable ϕ are given in Table 2. Interpolation is at the point $x_P = 8.0$, $y_P = 1.3$ and $z_P = 0.7$.

CPU times on a PC 486/66 MHz machine were:

- direct inversion (L–U decomposition), 54.7 s; $\phi_{\text{interpolated}} = 1.63$;
- present method (tolerance 10^{-2}), 9.3 s; $\phi_{\text{interpolated}} = 1.62$.

These results show that the new method is about five times faster for this particular case, with the important advantage that it locates the particle as well, whereas the matrix inversion method does not do this. The interpolated values above differ slightly because the two interpolation formulae used, namely the three-dimensional extension of eqn (1) for the L–U decomposition method, and eqn (3) for the present method, are somewhat different.

The present location/interpolation method can be compared with the location method of Ref. 5 for three-dimensional cells, which is based on a geometrical

Table 1. Example of location results for the particles in Fig. 3

Particle position			Number of iterations	Result: transformed coordinates		
X	Y	Z		ξ	η	s
1.0	0.75	0.6	2	+0.65	+0.29	+0.49
1.0	0.75	1.0	3	+0.69	+0.30	+1.43
0.25	0.25	0.0	3	-0.61	-0.70	-0.98
0.25	0.25	1.0	2	-0.83	-0.24	+0.87
0.5	0.5	0.5	1	-0.28	-0.092	-0.0073
0.5	0.50	-0.25	2	-0.15	-0.45	-1.43
0.25	1.25	0.0	3	-0.61	+0.69	-1.34
0.25	1.25	1.0	3	-0.49	+1.12	+0.71

approach: the particle point is inside the cell if it lies on the inner side of each of the six cell-faces. This is so if the dot-product of each face-area vector with the particle-point vector (referred to the face in question) is positive. Reference⁵ gives the actual computer program used, and the location procedure (subroutine PFIND) requires 342 arithmetic operations. The present method requires 119 operations in the preparatory stage [steps (1)–(5) of the algorithm given above], and 88 operations for each iteration [steps (6)–(9)]. Therefore, the two methods will require approximately the same CPU time for locating a particle, assuming that the present method takes on average three iterations to converge. This has been confirmed by writing the subroutine given in Ref. 5 and measuring CPU times in actual computations. Thus for the task of particle location alone the present method has a certain edge over the geometrical approach since two iterations are often enough and, furthermore, the preparatory operations involved are unnecessary in practice when the scheme is used in conjunction with a flow code since most of the values required at that stage are computed in the flow code itself (for example the matrix $[A^{-1}]^T$ is related to the face area-components). Furthermore, the present method provides the necessary data for interpolations simultaneously, whereas the geometrical method will require, in addition to the time spent locating the particle, five-fold that time for performing the interpolation.

4 CONVERGENCE ENHANCEMENT

Application of the base algorithm given above to some highly deformed cells may result in slow convergence. A typical case is given by triangular cross-section cells, used for example in a polar mesh, as represented in Fig. 4 where two-dimensionality is used for simplicity. For some of these

Table 2. Definition of cell and function ϕ used for interpolation test

	n	1	2	3	4	5	6	7	8
x	2	11	16	5	2	11	16	5	
y	1	1	4	4	1	1	4	4	
z	0	1	1	0	3	5	5	3	
ϕ	1	2	2	1	1	2	2	1	

case, it has been observed that the slow convergence is due to the fact that corrections to ξ^* in step (9) have opposite signs in successive iterations, yielding an oscillatory convergence to the final value of ξ . This is demonstrated in Fig. 5, where variation of the distance and the correction $\delta\xi$ and $\delta\eta$, with the iterations, are shown for a particle situated at $x_p = 0.8$ and $y_p = 0.2$ (see Fig. 4). In this case, 15 iterations are required to bring the distance d' to a value below a tolerance of 10^{-3} .

An obvious way to remedy this behaviour is to use only half of the correction to ξ^* whenever oscillatory convergence occurs. This is implemented by rewriting step (9) as

(9) At iteration k , compute the correction $\{\delta\xi^*\} = [A^{-1}]^T \{d'\}$; IF $((\delta\xi \cdot \delta\xi^{(k-1)}) \leq 0)$, then $\delta\xi = 0.5 \delta\xi$ (the same for η and s); update ξ^* as before, $\{\xi^*\} = \{\xi^*\} + \{\delta\xi^*\}$.

With this new step (9), the previous example took just six iterations to converge (Fig. 5). A number of other examples that have been tackled confirmed that this modification works well whenever oscillatory convergence occurs.

5 IMPROVEMENT FOR TRIANGULAR CELLS

While the previous modification works well when oscillatory convergence occurs, it has been observed that for some particles within triangular cross-section cells convergence may be slow, although monotonic (in that successive corrections $\delta\xi_i$ have the same sign). This can also occur for other types of cells which are very skewed,

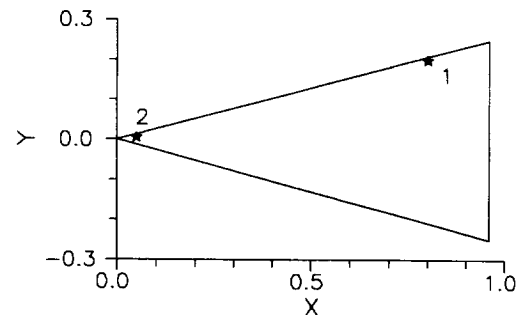


Fig. 4. Example of triangular cross-section cell with two particle points.

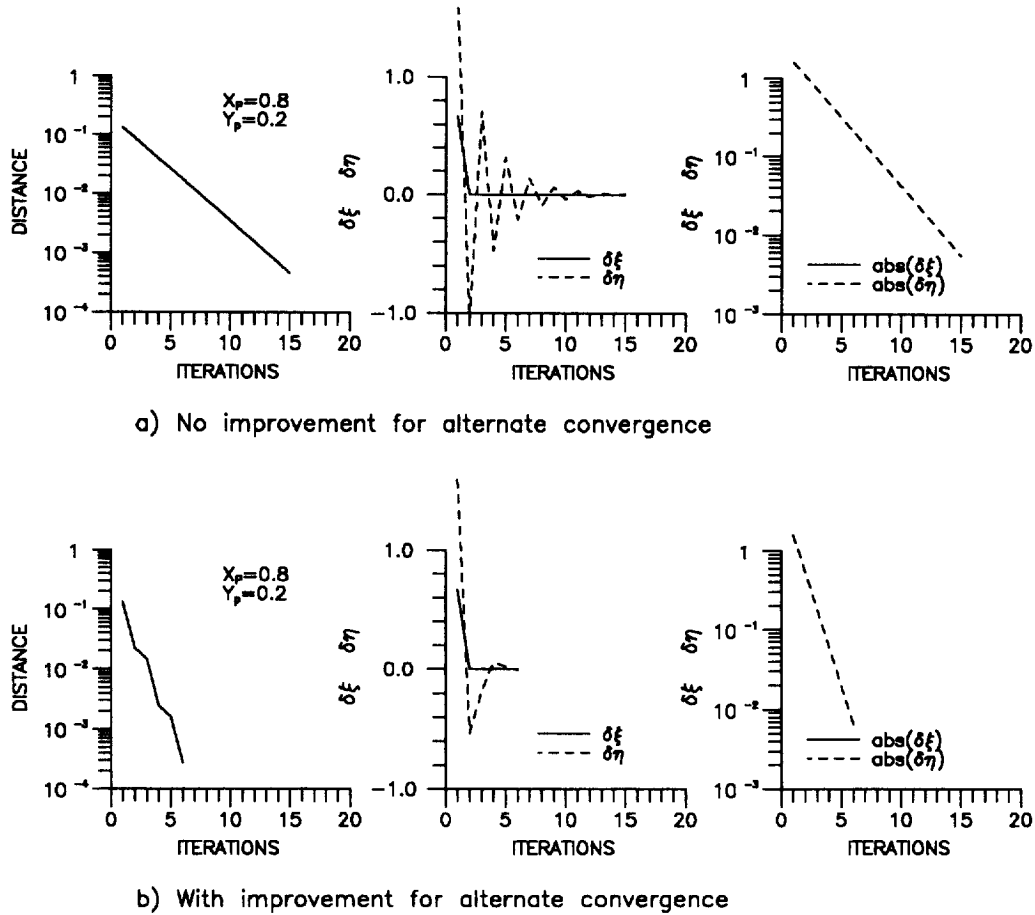


Fig. 5. Convergence history for particle 1 of Fig. 4: (a) without; and (b) with improvement for alternate convergence.

for example when two opposite edges form an acute angle, instead of being parallel. Figure 6 illustrates the convergence history for point $x_p = 0.05$, $y_p = 0.005$, very close to the vertex of the triangle in Fig. 4. Unlike the case given above (see Fig. 5), there is no oscillatory convergence here, but 19 iterations are still required to bring the error d' to below 10^{-3} .

Inspection of the convergence history for these particular particle points in Figs 5 and 6, provides a hint as to how to

reduce the number of iterations. It can be seen that the absolute values of $\delta\eta$ (the slow-converging component for these cases) follow a perfectly linear variation with the number of iterations in a semi-log graph. From this observation, one can assume that

$$\log_{10}(\delta\eta^{(k)}) = Ak + B \Rightarrow \delta\eta^{(k)} = Ce^{-ak} \quad (12)$$

where A , B , C and a are constants, and k is an iteration counter. The iterative particle-locating procedure amounts

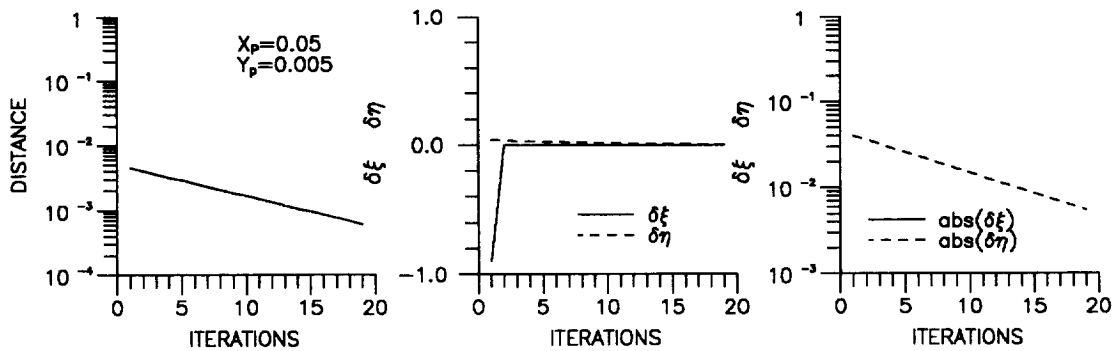


Fig. 6. Convergence history for locating particle 2 of Fig. 4.

to a successive correction of ξ_j , as given by step (9) of the algorithm, that is:

$$\begin{aligned}\eta^{(1)} &= \delta\eta^{(1)}, \\ \eta^{(2)} &= \eta^{(1)} + \delta\eta^{(2)}, \\ &\dots \\ \eta^{(k)} &= \eta^{(k-1)} + \delta\eta^{(k)} = \sum_{n=1}^k \delta\eta^{(n)}\end{aligned}$$

Using eqn (12), the sequence can be represented by

$$\eta^{(k)} = \sum_{n=1}^k C e^{-an} = C \sum_{n=1}^k (e^{-a})^n$$

which is the sum of a geometrical series with a ratio equal to (e^{-a}) ; since $a > 0$, then $(e^{-a}) < 1$, and the sum converges to

$$\lim_{k \rightarrow \infty} (\eta^{(k)}) \equiv \eta^\infty = C \left(\frac{e^{-a}}{1 - e^{-a}} \right) = C \left(\frac{1}{e^a - 1} \right) \quad (13)$$

The constants ' C ' and ' a ' can be determined from application of eqn (13) at two iterations, k_1 and k_2 , to yield

$$a = \frac{\ln(\eta^{(k_1)}) - \ln(\eta^{(k_2)})}{k_2 - k_1}, \quad \ln(C) = \frac{k_2 \ln(\eta^{(k_1)}) - k_1 \ln(\eta^{(k_2)})}{k_2 - k_1} \quad (14)$$

The reasoning above can also be applied to the oscillatory

case, for which one has

$$\begin{aligned}\eta^\infty &= C \sum_{n=1}^{\infty} (-1)^n (e^{-a})^n \\ &= C \{ (e^{-1a} + e^{-2a} + e^{-3a} + \dots) \\ &\quad - 2(e^{-2a} + e^{-4a} + e^{-6a} + \dots) \}\end{aligned}$$

and, in the limit:

$$\eta^\infty = C \left(\frac{1}{e^a - 1} - \frac{2}{e^{2a} - 1} \right) \quad (15)$$

The constants are determined from expressions similar to eqn (14) but now based on the absolute values of $\delta\eta$. It was found satisfactory to find these constants from the first and second iteration ($k_1 = 1$, $k_2 = 2$), for which case the following expressions are obtained:

$$a = \ln|\eta^{(1)}|, \eta^{(2)}|, \text{ and } C = e^a \eta^{(1)} \quad (16)$$

Implementation of this 'extrapolation to the limit' technique is explained below. At the end of iteration 2, before updating ξ^* in step (9):

- compute ' a ' and ' C ' from eqn (16);
- compute a limiting value for ξ_j based on the appropriate equation,
- IF $\{(\delta\xi_i^{(2)} \cdot \delta\xi_i^{(1)}) \leq 0\}$ use eqn (15) to obtain ξ_i^∞ , otherwise use eqn (13);
- update ξ from $\{\xi^*\} = \{\xi^\infty\}$ and go to step (6).

After introduction of this modification, the problematic case depicted in Fig. 6 was found to converge in just two

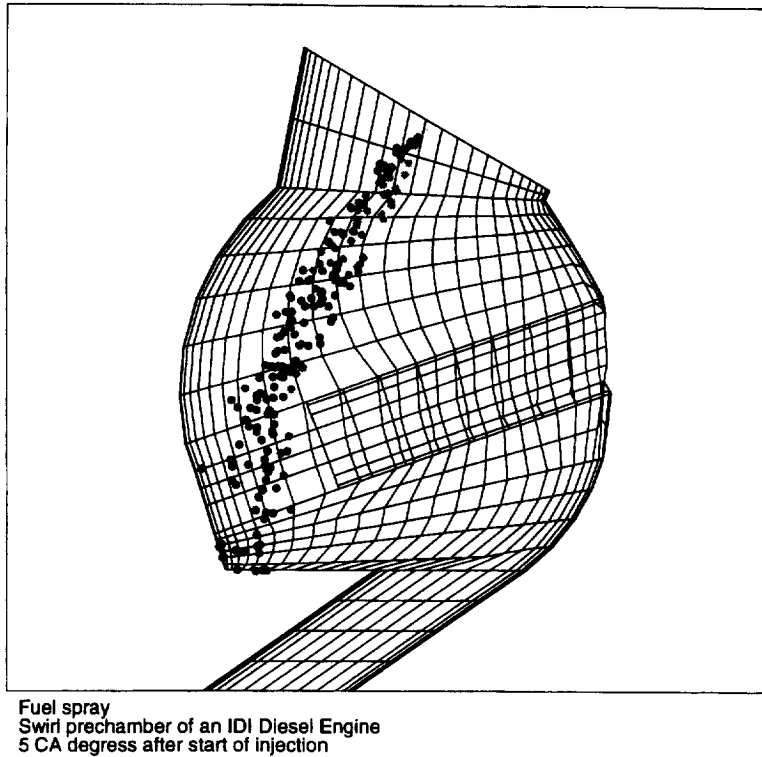


Fig. 7. Application of the locating method to track fuel drops in a Diesel engine flow.

iterations. For this case, since ξ and ς were already converged, the extrapolation to the limit of η arrives at the solution immediately. Note that the modification just described can be used in general, and not only for triangular cells. Indeed, a check was made to verify that this modification does not degrade the convergence rate for cases when it is not required and it was found that in all tests, at most one additional iteration was necessary compared to the standard algorithm.

6 EXAMPLE OF APPLICATION

As an actual example of application of the locating procedure to a CFD problem, Fig. 7 shows the position of droplet-parcels at a given time during a simulation of the flow in a cylinder of a Diesel engine. For this case the unstructured mesh moves as the piston-head advances in the cylinder creating an unsteady flow; fuel drops are injected as a spray in the pre-chamber shown in the figure. The flow code used in this simulation calls the algorithm just described whenever drop location and interpolation is required. In this, and in a variety of other calculations, the present algorithm proved to be robust and efficient.

7 CONCLUSIONS

A method has been developed to locate particles within arbitrary three-dimensional computational meshes (either of the finite-volume or finite-element type), and to interpolate field values from the mesh points to the particle position. In order to check whether a particle is located in a given computational cell, the cell is transformed using isoparametric functions onto a well-behaved cubic cell. The particle is located inside the cell if the transformed coordinates of the particle position are in the range $[-1, +1]$. These coordinates are obtained by means of an iterative method in which the position of the particle in transformed space is estimated and successively shifted

until the correct position in transformed space corresponding to the true position in physical space is found. It has been shown that this method has fast convergence in most cases. For irregular computational cells (e.g. collapsed triangular cells) two modifications are introduced to ensure fast and smooth convergence.

The method can easily be incorporated in a general CFD code to track particles within a computational mesh whether fixed or moving. The only variables required as input are the Cartesian coordinates of the particle and of the eight cell vertices.

REFERENCES

1. Gosman, A. D. & Peric, M., A computer program for tracking of particles in fluid flow and electrostatic fields. Mechanical Engineering Department Report, Imperial College, London, 1985.
2. Angelov, T. A. & Manoach, E. S., A point-in-domain identification program. *Adv. Engng Software*, 1989, **11**, 99–106.
3. Milgram, M. S., Does a point lie inside a polygon?. *J. Comput. Phys.*, 1989, **84**, 134–144.
4. Seldner, D. & Westermann, T., Algorithms for interpolation and localization in irregular 2D meshes. *J. Comput. Phys.*, 1988, **79**, 1–11.
5. Amsden, A. A., Ramshaw, J. D., O'Rourke, P. J. & Dukowicz, J. K., KIVA: a computer program for two- and three-dimensional fluid flows with chemical reactions and fuel sprays. Report LA-10245-MS, 1985.
6. Jensen, P. S., Finite difference techniques for variable grids. *Comp. Struct.*, 1972, **2**, 17–29.
7. Perrone, N. & Kao, R., A general finite difference method for arbitrary meshes. *Comp. Struct.*, 1975, **5**, 45–58.
8. Liszka, T. & Orkisz, J., The finite difference method at arbitrary irregular grids and its application in applied mechanics. *Comp. Struct.*, 1980, **11**, 83–95.
9. Oliveira, P. J., A method to locate particles and interpolate field values in an arbitrary grid. Internal Report, Mineral Research Engineering Department, Imperial College, London, February 1988.
10. Zienkiewicz, O. C. & Irons, B. M., Isoparametric elements. In *Finite Elements Techniques*, ed. H. Tottenham & C. Brebbia. University of Southampton, 1970.