Available online at www.sciencedirect.com

ScienceDirect

Journal of Computational Physics 225 (2007) 2198–2213

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

An efficient and robust particle-localization algorithm for unstructured grids

A. Haselbacher [a,*], F.M. Najjar [b], J.P. Ferry [b]

[a] *Department of Mechanical and Aeronautical Engineering, University of Florida, Gainesville, FL 32611, United States*
[b] *Center for Simulation of Advanced Rockets, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States*

Received 1 August 2005; received in revised form 5 December 2006; accepted 13 March 2007
Available online 2 April 2007

Abstract

An efficient and robust particle-localization algorithm for unstructured grids is presented. Given a particle position and the cell containing that position, the algorithm determines the cell which contains a nearby position. The algorithm is based on tracking a particle along its trajectory by computing the intersections of the trajectory and the cell faces. Compared to previously published particle-localization algorithms, the new algorithm has several advantages. First, it can be applied to grids consisting of arbitrary polyhedral cells. Second, the algorithm is not limited to small particle displacements. Third, the interaction of particles with boundaries is dealt with correctly and naturally. Fourth, the algorithm is more efficient than other published algorithms. A modified version of the present algorithm is also presented which can detect inconsistencies and take appropriate corrective action. With these modifications, the computational cost becomes comparable to other published algorithms which cannot detect inconsistencies without resorting to fall-back algorithms such as exhaustive or Octree search.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Particle-localization; Particle tracking; Eulerian–Lagrangian simulations; Unstructured grids

1. Introduction

Multi-phase flows are encountered in many engineering applications. The evolution of the carrier phase is often determined using the Eulerian approach, in which partial differential equations for the mass, momentum, and energy per unit volume are solved. In the Lagrangian approach of simulating the disperse phase, ordinary differential equations for the position, mass, momentum, and energy of the droplets or particles are solved. The solution of these equations requires the evaluation of fluid properties at the droplet locations. If the fluid properties are evolved using an Eulerian approach, evaluating fluid properties at the droplet location requires locating the cell which contains the droplet. For consistency with previous work, we refer to this problem as the particle-localization problem because the distinction between droplets and particles is

* Corresponding author. Tel.: +1 352 392 9459; fax: +1 352 392 1071.
E-mail address: haselbac@ufl.edu (A. Haselbacher).

0021-9991/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2007.03.018

irrelevant in this context. A formal statement of the particle-localization problem is: Given a grid, a particle position, and the cell which contains that particle position, determine the cell which contains a nearby particle position. Algorithms which solve this problem are referred to as particle-localization algorithms.

Because particles typically move only small distances in each time step, particle-localization algorithms can take advantage of the position at the previous time step as an initial guess, thereby reducing the number of operations and making the particle-localization algorithm more efficient. A number of particle-localization algorithms have been presented in the literature.

Brackbill and Ruppel [3] developed a particle-localization algorithm for use with the particle-in-cell method in two dimensions. They developed an efficient algorithm based on the computational coordinates which limits searching to neighboring cells. Seldner and Westermann [11] used a Cartesian background grid to locate particles in boundary-fitted grids. This makes the localization very efficient, but leads to problems near curved boundaries. Subsequently, Westermann [13] presented several algorithms which eliminate problems with locating particles near curved boundaries on quadrilateral grids. Löhner and Ambrosiano [8] presented a particle-localization algorithm based on the properties of basis functions used in finite-element methods. This algorithm also limits searching to neighboring cells. They demonstrated the algorithm for unstructured triangular grids with linear basis functions. Löhner [7] subsequently termed this algorithm the known-vicinity algorithm. Li and Modest [6] tracked particles in triangular grids by comparing the ratio of the particle trajectory normal to a face to the normal distance between the old particle location and that face. If this ratio exceeds unity for a given face, the particle is passed to the cell adjacent to that face. The cell containing a particle location is found if the ratios associated with all faces are less than unity. Subramaniam and Haworth [12] used trilinear basis functions to track particles on polyhedral grids in a hybrid Lagrangian–Eulerian probability-density function approach for the simulation of internal combustion engines. In their approach, a particle is tracked explicitly along its trajectory by computing the time to the intersection of the closest face which contains the particle. Together with an appropriate data structure, the particle is passed to the cell adjacent to the intersected face. The tracking of a particle is continued until the accumulated intersection times equal or exceed the time step. Apte et al. [1] employed a version of the known-vicinity algorithm of Löhner [7] to track the evolution of particles in a coaxial-jet combustor on unstructured hexahedral grids. Given the old and new locations and the cell containing the old location, the algorithm places the particle in successive face-neighboring cells such that the distance between the centroid of a cell and the new particle location is minimized. Because minimizing the distance between the particle location and a cell centroid does not guarantee that the particle is located in that cell, it may be necessary to fall back on an alternative localization algorithm. In the work of Apte et al., an exhaustive search is used if a particle is not located within 10–15 attempts.

It can be seen that previous work resulted in a number of efficient particle-localization algorithms. With the exception of Löhner [7] and Apte et al. [1], the robustness of particle-localization algorithms is generally not addressed. A robust particle-localization algorithm is not limited to small particle displacements. In addition, few authors consider the interaction with boundaries when developing particle-localization algorithms. The goal of the present work is to develop a particle-localization algorithm which is applicable to polyhedral unstructured grids, is robust, treats boundary interactions correctly and easily, and is efficient.

The rest of this article is structured as follows: The new particle-localization algorithm is described in Section 2. A detailed discussion of the new algorithm and a comparison to other published algorithms are given in Section 3. The serial and parallel implementation of the new algorithm is presented in Section 4. The serial and parallel performance of the new algorithm is assessed in Section 5. Results obtained by applying the new algorithm to the flow in a solid-propellant rocket are shown in Section 6. Conclusions are drawn in Section 7.

2. Particle-localization algorithm

The basic idea of the present particle-localization algorithm is the following: Assume that the particle is known to be located in cell c_1 and to move along a given trajectory. Assume further that we can determine which face of cell c_1 is intersected by the particle trajectory. If the cell adjacent to the intersected face is c_2, the particle must pass from cell c_1 into cell c_2. By applying this idea repeatedly, we can determine the cell c_n which contains the predicted new particle position. A cell is said to contain a particle location $\mathbf{r}_P$ if this position satisfies the so-called "in-cell test," i.e., if for each face of the cell,

$$(\mathbf{r}_C - \mathbf{r}_P) \cdot \mathbf{n} \geqslant 0, \tag{1}$$

where $\mathbf{r}_C$ is the centroid of the face and $\mathbf{n}$ is the outward unit normal of the face.

The present algorithm is based on the same idea as that of Subramaniam and Haworth [12] but was developed independently. The main difference is that the present algorithm is based on computing intersection distances rather than intersection times. This choice appears more intuitive given that the particle-localization problem is a spatial searching problem. The present algorithm has the added advantage, as discussed below, that the localization of a moving particle can be made consistent with the localization of a stationary particle and thus allows a unified treatment. Because the present algorithm focuses on intersections with faces and boundaries are represented by a collection of faces, the algorithm treats boundary interactions in a natural way. For simplicity, the algorithm is first described for the case where boundary interactions do not occur. For ease of exposition, the algorithm will be described mostly in a two-dimensional setting. It is important to understand, however, that the new algorithm is not restricted to two dimensions and has been implemented in three dimensions.

2.1. Description of algorithm without boundary interactions

Fig. 1 illustrates the particle-localization problem: We are given a particle position $\mathbf{r}_P$ and the cell which contains that position and we are to find the cell which contains a nearby particle position $\mathbf{r}_Q$. From the given positions of the particle the distance traveled, $d = \|\mathbf{r}_Q - \mathbf{r}_P\|$, and the trajectory, $\mathbf{t} = (\mathbf{r}_Q - \mathbf{r}_P)/d$, can be computed.

The algorithm will be described in stages. For the moment, consider only the cell which contains the particle position $\mathbf{r}_P$. The cell is defined by the four vertices V_1, V_2, V_3, and V_4, see Fig. 2. The vertices are connected to give faces with outward unit normal vectors $\mathbf{n}_1$, $\mathbf{n}_2$, $\mathbf{n}_3$, and $\mathbf{n}_4$. The faces may be defined by the parametric representation of a straight line,

$$\mathbf{r}(\xi) = (1-\xi)\mathbf{r}_1 + \xi\mathbf{r}_2, \quad 0 \leqslant \xi \leqslant 1. \tag{2}$$

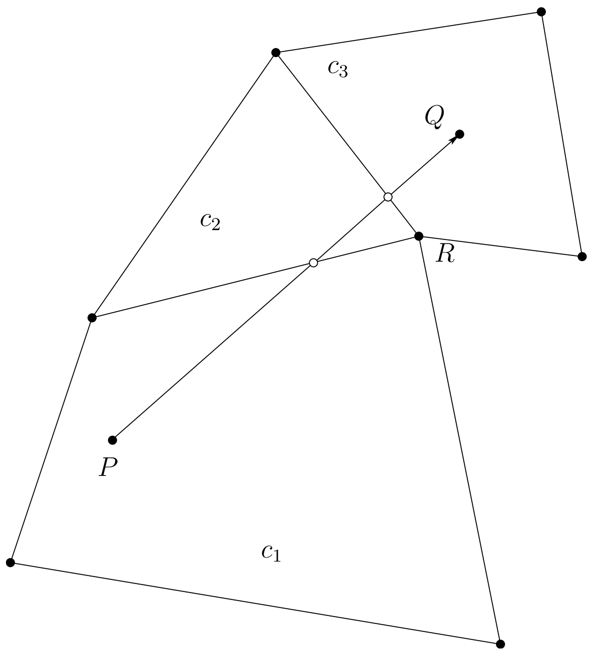

Fig. 1. Illustration of particle-localization problem: find the cell which contains the particle position Q given that the particle was known to be located at P in cell 1.

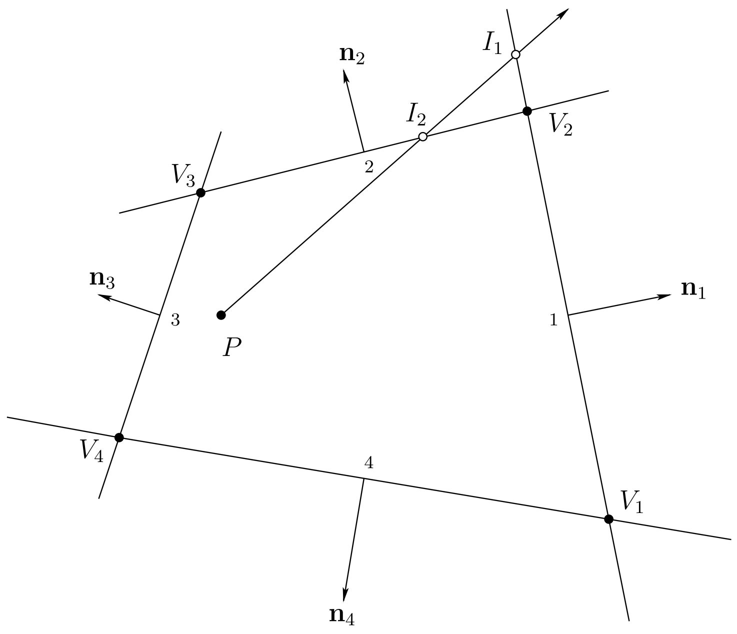

Fig. 2. Computation of intersection points given a particle position P and a particle trajectory **t**. Small numbers indicate the face numbering local to the cell.

The algorithm computes the intersection points I_i of the trajectory with the faces. At this stage of explaining the algorithm, the precise method of computing the intersection points is immaterial. The method used in the present work is described in Section 2.3. For each intersection point, the associated intersection distance $\alpha_{I_i} = \|\mathbf{r}_{I_i} - \mathbf{r}_{\mathrm{P}}\|$ is computed. It is obvious that the only faces for which intersection points must be computed are those for which $\mathbf{t} \cdot \mathbf{n} > 0$. The intersection points of the trajectory with the lines defined by the faces are denoted by I_1 and I_2, respectively, in Fig. 2. Note that the intersection point I_1 lies outside of face 1. That is, I_1 does not satisfy the parametric representation of face 1 given by Eq. (2) because $\xi_{I_1} > 1$. It would appear, therefore, that in addition to computing the intersection points of the trajectory with the faces, it is also necessary to check whether these intersection points actually lie within the faces. However, this is not necessary because we are ultimately only interested in the face with the smallest intersection distance. This is because in traveling along the trajectory, the plane with the smallest intersection distance will be intersected first. It is easy to see that the smallest intersection distance must always lie within its associated face so that it is not necessary to check whether a given intersection point is located within its associated face. This simplification is very important, particularly in three dimensions, because determining whether the intersection point lies within a face in three dimensions is tantamount to dealing with a two-dimensional intersection problem.

Once the algorithm determines which face is intersected by the trajectory, the particle can be assigned to the cell adjacent to that face and the distance which remains to be traveled is updated according to

$$d \leftarrow d - \min_i \alpha_{I_i}. \tag{3}$$

After the particle is assigned to the new cell, the algorithm is simply applied again in the same manner as just described until the minimum intersection distance exceeds the distance which remains to be traveled.

It is worth pointing out that the algorithm deals naturally with trajectories that pass directly through vertices. In this case, the intersection distances of two or more faces meeting at the vertex are equal to within machine precision. The algorithm will pick the face associated with the smallest intersection distance. Suppose that the trajectory passed through the vertex R in Fig. 1 and that the algorithm will pass the particle to cell 2. The smallest intersection distance for cell 2 will be zero because the intersection point coincides with R. Once the particle is passed to cell 3, the algorithm continues as described above.

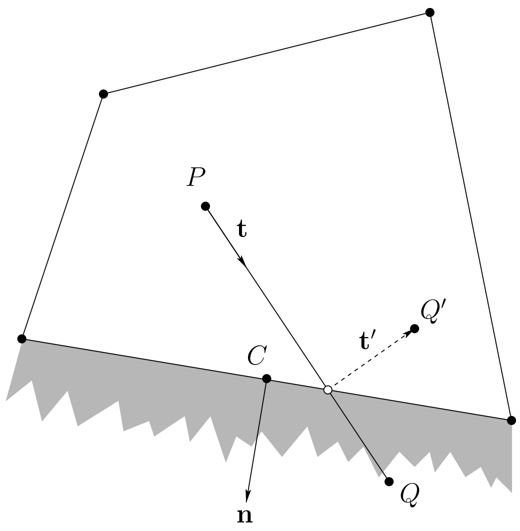

Fig. 3. Illustration of reflection of the predicted position Q and trajectory **t** on solid walls.

2.2. Description of algorithm with boundary interactions

Consider the case where the predicted new position of the particle lies outside the computational domain, as depicted in Fig. 3. The shading indicates the presence of a solid wall, which means that the predicted particle location must be reflected about the wall normal. This is treated naturally with the present algorithm because the algorithm detects that the intersection point is associated with a boundary face. This allows boundary conditions to be treated in a unified manner by modifying the trajectory and predicted new position through the appropriate transformations,

$$\mathbf{t}' = \mathbf{A}\mathbf{t}, \tag{4}$$

where $\mathbf{t}$ is the predicted trajectory, $\mathbf{t}'$ is the modified trajectory, and

$$\mathbf{r}_{Q'} = \mathbf{A}\mathbf{r}_Q + \mathbf{b}, \tag{5}$$

where $\mathbf{r}_Q$ is the predicted particle position and $\mathbf{r}_{Q'}$ is the modified predicted particle position. The precise forms of $\mathbf{A}$ and $\mathbf{b}$ depend on the boundary condition. For example, for a solid or symmetry boundary, the appropriate operation is a reflection, so $\mathbf{A} = \mathbf{I} - 2\mathbf{n}\mathbf{n}$ and $\mathbf{b} = 2(\mathbf{r}_C \cdot \mathbf{n})\mathbf{n}$, where $\mathbf{I}$ is the identity tensor and $\mathbf{n}$ and $\mathbf{r}_C$ are the outward unit normal vector and the position vector of the centroid of the boundary face, respectively. Other boundaries can be treated similarly.

2.3. Computation of trajectory-face intersections

The fundamental operation of the present particle-localization algorithm is the computation of the intersection of the particle trajectory with the faces of the cell which contains the particle. The following subsections describe the computation of the intersection of the trajectory with planar and non-planar faces.

2.3.1. Planar faces

The problem of finding the intersection of the particle trajectory with a planar face can be abstracted as determining the intersection of a ray $\mathbf{t}$ anchored at the point $\mathbf{r}_P$,

$$\mathbf{r}(\alpha) = \mathbf{r}_P + \alpha\mathbf{t}, \tag{6}$$

with a plane specified by the normal vector $\mathbf{n}$ and anchored at the point $\mathbf{r}_C$,

$$(\mathbf{r} - \mathbf{r}_C) \cdot \mathbf{n} = 0. \tag{7}$$

Substituting Eq. (7) into Eq. (6) gives the distance between the intersection point $\mathbf{r}_\mathrm{I}$ and $\mathbf{r}_\mathrm{P}$ as

$$\alpha_\mathrm{I} = \frac{(\mathbf{r}_\mathrm{C} - \mathbf{r}_\mathrm{P}) \cdot \mathbf{n}}{\mathbf{t} \cdot \mathbf{n}}. \tag{8}$$

Note that it is neither necessary to solve a linear system for the intersection point nor to compute the intersection distance from the square root of the summed squares of coordinate differences.

It is instructive to consider the meaning of the numerator and denominator of the right-hand side of Eq. (8) in the context of particle tracking. The numerator is the signed normal distance between the plane and the particle position; it is positive if the particle is located in the cell. The denominator indicates the orientation of the trajectory relative to the face normal; if the denominator is positive (negative), the particle is moving toward (away from) the face, and if it is zero, no intersection is possible. The interpretation of Eq. (8) is summarized in Fig. 7, which will be discussed further in Section 3.1.

2.3.2. Non-planar faces

Non-planar faces can be treated if Eq. (7) is replaced by the appropriate equation. For example, the parametric equation describing a bilinear patch can be written as

$$\mathbf{r}(u,v) = uv\mathbf{a} + u\mathbf{b} + v\mathbf{c} + \mathbf{d}, \tag{9}$$

where $0 \leqslant u \leqslant 1$ and $0 \leqslant v \leqslant 1$ are parameters, $\mathbf{a} = \mathbf{r}_{00} - \mathbf{r}_{10} + \mathbf{r}_{11} - \mathbf{r}_{01}$, $\mathbf{b} = \mathbf{r}_{10} - \mathbf{r}_{00}$, $\mathbf{c} = \mathbf{r}_{01} - \mathbf{r}_{00}$, $\mathbf{d} = \mathbf{r}_{00}$, and $\mathbf{r}_{ij}$ are the position vectors of four non-coplanar points, i.e., $\mathbf{a} \neq 0$. The intersection point(s) of a ray and a bilinear patch can then again be computed from Eqs. (9) and (6). Note that a ray may intersect a bilinear patch in one location, two locations, or not at all. Ramsey et al. [10] recently presented an efficient and robust algorithm to compute the intersection point(s) of a ray and a bilinear patch. Here we present a modification of this algorithm which is more efficient.

The intersection point with the ray given by Eq. (6) is given by

$$uv\mathbf{a} + u\mathbf{b} + v\mathbf{c} + \tilde{\mathbf{d}} = \alpha_\mathrm{I}\mathbf{t}, \tag{10}$$

where $\tilde{\mathbf{d}} = \mathbf{d} - \mathbf{r}_\mathrm{P}$. The intersection distance is obtained immediately from

$$\alpha_\mathrm{I} = (uv\mathbf{a} + u\mathbf{b} + v\mathbf{c} + \tilde{\mathbf{d}}) \cdot \mathbf{t}. \tag{11}$$

Substituting Eq. (11) into Eq. (10) gives

$$uv\mathbf{a}^\perp + u\mathbf{b}^\perp + v\mathbf{c}^\perp + \tilde{\mathbf{d}}^\perp = 0, \tag{12}$$

where the superscript $\perp$ denotes the component normal to the ray $\mathbf{t}$, e.g. $\mathbf{a}^\perp = \mathbf{a} - (\mathbf{a} \cdot \mathbf{t})\mathbf{t}$. Eq. (12) can be expressed as

$$uva^\perp_{xz} + ub^\perp_{xz} + vc^\perp_{xz} + \tilde{d}^\perp_{xz} = 0, \tag{13}$$

$$uva^\perp_{yz} + ub^\perp_{yz} + vc^\perp_{yz} + \tilde{d}^\perp_{yz} = 0, \tag{14}$$

where the subscripts denote differences of components, e.g. $a^\perp_{xz} = a^\perp_z - a^\perp_x$. Solving Eq. (13) for u and substituting into Eq. (14) leads to

$$(a^\perp_{xz}c^\perp_{yz} - a^\perp_{yz}c^\perp_{xz})v^2 + (a^\perp_{xz}\tilde{d}^\perp_{yz} - a^\perp_{yz}\tilde{d}^\perp_{xz} + b^\perp_{xz}c^\perp_{yz} - b^\perp_{yz}c^\perp_{xz})v + b^\perp_{xz}\tilde{d}^\perp_{yz} - b^\perp_{yz}\tilde{d}^\perp_{xz} = 0. \tag{15}$$

For any root $0 \leqslant v \leqslant 1$, the corresponding u can be computed from Eqs. (13) or (14) and if $0 \leqslant u \leqslant 1$, the associated intersection distance is determined from Eq. (11). Otherwise, the ray does not intersect the bilinear patch.

If the ray intersects the bilinear patch in more than one location, then the intersection point with the smaller intersection distance should be taken in principle, as discussed above. However, if the distance which remains to be traveled exceeds the larger intersection distance, then the intersections with that face can be ignored to improve efficiency. This simplification is possible because the ray, after exiting the cell through the intersection point with the smaller intersection distance, would reenter the same cell through the intersection point with the larger intersection distance.

3. Discussion

3.1. Variant of present algorithm

The description of the algorithm in Section 2 indicated that the intersection distance only needed to be computed for faces for which $\mathbf{t}\cdot\mathbf{n} > 0$. This improves the efficiency of the algorithm because fewer arithmetic operations are performed: If $\mathbf{t}\cdot\mathbf{n} \leqslant 0$, it is neither necessary to compute $(\mathbf{r}_\mathrm{C} - \mathbf{r}_\mathrm{P})\cdot\mathbf{n}$ nor to divide the latter by the former to obtain α_I. The increased efficiency is obtained by sacrificing the capability of checking for self-consistency. In the present context, self-consistency means that a particle satisfies the in-cell test for the cell with which it is associated. Recall that the term $(\mathbf{r}_\mathrm{C} - \mathbf{r}_\mathrm{P})\cdot\mathbf{n}$ appears in Eqs. (8) and (1) and that a negative value indicates that the particle is outside the given cell.

Now assume that for some reason the particle fails the in-cell test and that the face which lead to the failure is one for which $\mathbf{t}\cdot\mathbf{n} \leqslant 0$. The algorithm in its presented form cannot detect this problem and is therefore unable to take corrective action. However, the algorithm is easily modified to detect this problem and take the appropriate corrective action at the expense of additional arithmetic operations. Hence we refer to the modified algorithm as the "robust version" and to the original algorithm as the "fast version."

The modified algorithm computes the numerator of Eq. (8) for all faces of a given cell irrespective of the sign of $\mathbf{t}\cdot\mathbf{n}$. If the numerator for any face is negative, the algorithm assigns the particle to the adjacent cell while leaving the physical position of the particle unaltered. With this simple modification, the present algorithm can detect and deal with internal errors gracefully and the use of fall-back algorithms such as exhaustive or Octree search becomes unnecessary.

One example of where internal errors can occur is with the creation of particles at inflow or injection boundaries. We take the injection boundary as an example because of its relevance to solid-rocket motor simulations. When particles are created at an injection boundary, a random position $\mathbf{r}_\mathrm{B}$ is generated on the boundary face at which the particle is assumed to be injected. The initial position $\mathbf{r}_\mathrm{P}$ of the particle is then given as $\mathbf{r}_\mathrm{P} = \mathbf{r}_\mathrm{B} - d\mathbf{n}$, where $\mathbf{n}$ is the outward unit normal vector to the boundary face and d is a distance normal to the boundary face. This distance is taken as a fraction of the ratio of the volume of the cell adjacent to the boundary face divided by the area of the boundary face. Depending on the shape of the cell adjacent to the boundary face, the injection position $\mathbf{r}_\mathrm{B}$, and the injection distance d, $\mathbf{r}_\mathrm{P}$ can lie outside of the cell adjacent to the boundary face as depicted in Fig. 8. If the injection algorithm assumes that the particle is always assigned the cell adjacent to the boundary face, then the in-cell test would fail. There are two possible solutions to this problem with current particle-localization algorithms. First, it is possible to generate a sequence of injection positions until the particle position is located inside the cell adjacent to the boundary face. This is undesirable because it would introduce a bias in the injected particle positions. Second, the correct cell could be found by fall-back algorithms such as exhaustive or Octree search. With the robust version of the present algorithm, this problem is solved without introducing a bias or resorting to fall-back algorithms.

3.2. Comparison with other algorithms

The correct and natural treatment of boundary interactions by the present algorithm could be labeled a mere convenience. However, it is important to recognize that some other algorithms, at least in their published forms, do not appear to treat boundary interactions correctly. For example, in the algorithm of Zhou and Leschziner [14], the particle is assigned to the cell adjacent to the first face which fails their version of in-cell test. In doing so, Zhou–Leschziner algorithm can fail to detect boundary interactions if boundary faces of a given cell are not tested first. The algorithm of Löhner and Ambrosiano [8] assigns the particle to the cell adjacent to the face which fails the in-cell test "the most" and hence can also fail to detect boundary interactions unless the failure of a boundary face takes precedence. With the present algorithm, no modifications are required near boundaries.

In the context of general search algorithms, the use of the trajectory is useful in addressing a deficiency of known-vicinity algorithms near boundaries, see Fig. 5 in Löhner [7]. Instead of stopping near the boundary and reverting to a fall-back algorithm, the search can continue by considering the intersections of the trajectory with other boundary faces. In this case, only boundary faces for which $\mathbf{t} \cdot \mathbf{n} < 0$ need to be considered.

4. Implementation

4.1. Programming the present algorithm

Programming the present algorithm is relatively simple. The only data structures required are face-to-cell and cell-to-face connectivity tables. The former is required by finite-volume methods without particle localization. The latter is easily constructed given the former. For reference, the present particle-localization algorithm is summarized below in a format which is independent of the precise format of data structures. For simplicity, only intersections with planar faces and interactions with solid boundaries are included. The extension to non-planar faces and other boundaries is straightforward following the descriptions in Sections 2.3 and 2.2. The algorithm descriptions in Figs. 4–6 use the same nomenclature as the descriptions above. In addition, the following symbols are used: $\mathscr{F}_\Omega$ denotes the set of interior faces and $\mathscr{F}_{\partial\Omega}$ denotes the set of boundary faces.

```
Require: $\mathbf{r}_P^n, \mathbf{r}_P^{n+1}, c_P^n$
Ensure: $c_P^{n+1}$
 1: procedure ParticleTracking($\mathbf{r}_P^n, \mathbf{r}_P^{n+1}, c_P^n$)
 2:     $d = \|\mathbf{r}_P^{n+1} - \mathbf{r}_P^n\|$
 3:     $\mathbf{t} = (\mathbf{r}_P^{n+1} - \mathbf{r}_P^n)/d$
 4:     $\mathbf{r}_P = \mathbf{r}_P^n$
 5:     $c_P = c_P^n$                                   Initialize particle location
 6:     loop
 7:         Intersect($\mathbf{t}, \mathbf{r}_P, c_P, \alpha_{\min}, f_{\min}$)
 8:         $d \leftarrow d - \alpha_{\min}$
 9:         if $d \geq 0$ then
10:             if $f_{\min} \in \mathcal{F}_\Omega$ then
11:                 $c_P = \mathrm{c2f}(c_P, f_{\min})$            Get adjacent cell
12:             else
13:                 if $f_{\min} \in \mathcal{F}_{\partial\Omega}$ then
14:                     Reflect($\mathbf{t}, \mathbf{r}_P, f_{\min}$)      Reflect for solid wall
15:                 end if
16:             end if
17:         else
18:             $c_P^{n+1} = c_P$                  Found cell containing new position
19:             exit
20:         end if
21:     end loop
22: end procedure
```

Fig. 4. Summary of particle tracking algorithm.

4.2. Impact of finite-precision arithmetic

Algorithms which make decisions based on geometric tests can fail if these tests return false answers because of round-off errors introduced by finite-precision arithmetic. A common problem is testing two floating-point numbers for equality, or, equivalently, determining the sign of their difference. If the two numbers are similar in magnitude, the test is likely to be dominated by round-off errors and therefore may not return the correct answer.

The present algorithm, being based on geometric tests, is no exception. In the present algorithm, the numerator of Eq. (8) can lead to problems because it is based on determining the sign of coordinate differences and these differences will be vanishingly small at the intersection points. To eliminate this problem, a tolerance ε is introduced, i.e., in deciding whether a particle is located inside a given cell, the numerator of Eq. (8) is replaced by

$$(\mathbf{r}_\mathrm{C} - \mathbf{r}_\mathrm{P}) \cdot \mathbf{n} \geqslant -\varepsilon, \tag{16}$$

where ε is a positive number, typically $O(10^2\varepsilon_\mathrm{mach})$ with $\varepsilon_\mathrm{mach}$ being the machine precision. This approach is based on the concept of epsilon-geometry of Guibas et al. [5] and is equivalent to introducing a "fuzziness" into the topological position of a particle (the cell to which the particle is assigned). The introduction of the tolerance ε to the in-cell test leads to the modified algorithm depicted in Fig. 9.

It is important to note that the use of a tolerance as in Eq. (16) is only required for the robust version of the present particle-localization algorithm. This is because the robust version computes the numerator of Eq. (8) for all faces of a given cell irrespective of the sign of $\mathbf{t} \cdot \mathbf{n}$. Hence it is possible that cells adjacent to a given face consider a particle positioned on that face to be located in the opposite cell because for each cell the numerator of Eq. (8) is dominated by round-off error, i.e., $(\mathbf{r}_\mathrm{C} - \mathbf{r}_\mathrm{P}) \cdot \mathbf{n} \approx -\varepsilon_\mathrm{mach}$. The use of Eq. (16) resolves this problem. The fast version of the present algorithm does not require the use of a tolerance because only the faces for which $\mathbf{t} \cdot \mathbf{n} > 0$ are considered.

```
Require: $\mathbf{t}, \mathbf{r}_P, c_P$
Ensure: $\alpha_{\min}, f_{\min}$
 1: procedure INTERSECT($\mathbf{t}, \mathbf{r}_P, c_P, \alpha_{\min}, f_{\min}$)
 2:     $\alpha_{\min} = \infty$
 3:     for all $f \in \mathcal{F}(c_P)$ do                      Loop over faces of cell $c_P$
 4:         $\mathbf{n} = \mathbf{n}(f)$                                Get face normal
 5:         if $\mathbf{t} \cdot \mathbf{n} > 0$ then
 6:             $\mathbf{r}_C = \mathbf{r}_C(f)$                        Get face centroid
 7:             $\alpha_I = [(\mathbf{r}_C - \mathbf{r}_P) \cdot \mathbf{n}] / (\mathbf{t} \cdot \mathbf{n})$        Compute distance
 8:             if $\alpha_I < \alpha_{\min}$ then
 9:                 $\alpha_{\min} = \alpha_I$
10:                 $f_{\min} = f$
11:             end if
12:         end if
13:     end for
14:     $\mathbf{r}_P \leftarrow \mathbf{r}_P + \alpha_{\min}\mathbf{t}$                  Update particle position
15: end procedure
```

Fig. 5. Summary of fast intersection algorithm for planar faces.

```
Require: $\mathbf{t}, \mathbf{r}_P, c_P$
Ensure: $\alpha_{\min}, f_{\min}$
 1: procedure INTERSECT($\mathbf{t}, \mathbf{r}_P, c_P, \alpha_{\min}, f_{\min}$)
 2:     $\alpha_{\min} = \infty$
 3:     for all $f \in \mathcal{F}(c_P)$ do                    Loop over faces of cell $c_P$
 4:         $\mathbf{n} = \mathbf{n}(f)$                                  Get face normal
 5:         $\mathbf{r}_C = \mathbf{r}_C(f)$                                Get face centroid
 6:         if $(\mathbf{r}_C - \mathbf{r}_P) \cdot \mathbf{n} \geq -\varepsilon$ then
 7:             if $\mathbf{t} \cdot \mathbf{n} > 0$ then
 8:                 $\alpha_I = \max\{[(\mathbf{r}_C - \mathbf{r}_P) \cdot \mathbf{n}] / (\mathbf{t} \cdot \mathbf{n}), 0\}$    Compute distance
 9:                 if $\alpha_I < \alpha_{\min}$ then
10:                     $\alpha_{\min} = \alpha_I$
11:                     $f_{\min} = f$
12:                 end if
13:             end if
14:         else                                             Failed in-cell test
15:             $\alpha_{\min} = 0$                      Do not change physical position
16:             $f_{\min} = f$
17:             exit
18:         end if
19:     end for
20:     $\mathbf{r}_P \leftarrow \mathbf{r}_P + \alpha_{\min}\mathbf{t}$                   Update particle position
21: end procedure
```

Fig. 6. Summary of robust intersection algorithm for planar faces.

Another instance in which tolerances can be useful is in computing intersections of the trajectory with non-planar faces using the intersection algorithm for planar faces. The purpose of the tolerance is then to reconcile the use of an algorithm formally restricted to planar faces as it is applied to non-planar faces. Suitable values of the tolerance can be estimated from $\varepsilon = \mathbf{a} \cdot \mathbf{n}$ or $\varepsilon = \max_{ij}|(\mathbf{r}_C - \mathbf{r}_{ij}) \cdot \mathbf{n}|$ where $\mathbf{a}$ and $\mathbf{r}_{ij}$ are as defined in Section 2.3.2 and $\mathbf{n}$ is a representative normal vector. Typical values of ε estimated using these methods are $O(10^{-6})$ for relatively mild deviations from planarity.

4.3. Parallelization

The main challenge in parallelizing particle-localization algorithms is that the number of particles to be communicated can vary both in space and time. For the present particle-localization algorithm, a further challenge is caused by the requirement that it not be limited to small particle displacements. Hence the parallelization must be able to treat correctly particles which cross one or more partition boundaries between the old and new positions. An algorithm which meets these challenges can be summarized as follows:

		$(\mathbf{r}_C - \mathbf{r}_P) \cdot \mathbf{n}$	
		≥ 0	< 0
$\mathbf{t} \cdot \mathbf{n}$	> 0	$\alpha_I = \dfrac{(\mathbf{r}_C - \mathbf{r}_P) \cdot \mathbf{n}}{\mathbf{t} \cdot \mathbf{n}}$	Error
	≤ 0	No intersection	

Fig. 7. Summary of fast algorithm in terms of numerator and denominator of Eq. (8).

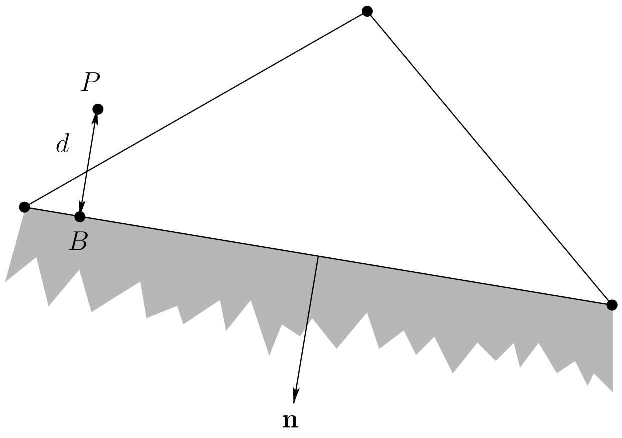

Fig. 8. Illustration of problem with initial location of particle near injection boundary.

(1) If a particle is detected to intersect a face straddled by an interior and a dummy cell, the corresponding partition-boundary index is determined, the counter of particles to be sent is incremented and the particle index is saved for that partition boundary, and the particle is flagged for communication.
(2) Once all particles have been either located or flagged for communication, the total number of particles which need to be communicated between all partitions is determined by a reduction operation.
(3) Assuming the total number of particles to be communicated is non-zero, each partition sends messages containing the number of particles to be sent for each partition boundary to the corresponding neighboring partition. While these messages are in transit, send buffers for the solution data associated with the particles (e.g. position, mass, momentum, energy, composition) are allocated and filled. On receiving the number of particles to be sent, the corresponding receive buffers are allocated.
(4) The send buffers are sent. While these messages are in transit, the particle data structure is updated by deleting all particles flagged for communication.
(5) The send buffers are received and the particles in these buffers are added to the data structure. The send and receive buffers are deallocated. All particles which were received are passed to the particle-localization algorithm and are continued to be tracked.

		$(\mathbf{r}_C - \mathbf{r}_P) \cdot \mathbf{n}$		
		≥ 0	< 0	
			$\geq -\varepsilon$	$< -\varepsilon$
$\mathbf{t} \cdot \mathbf{n}$	> 0	$\alpha_I = \dfrac{(\mathbf{r}_C - \mathbf{r}_P) \cdot \mathbf{n}}{\mathbf{t} \cdot \mathbf{n}}$	$\alpha_I = 0$	$\alpha_I = 0$
	≤ 0	No intersection		

Fig. 9. Summary of robust algorithm in terms of numerator and denominator of Eq. (8). Compare to Fig. 7.

5. Performance

5.1. Serial performance

5.1.1. Planar faces

The serial performance of the present algorithm is assessed by measuring the time required to locate 100000 particles on tetrahedral and hexahedral grids for a rectangular region of extent $-32.5 \leqslant x \leqslant 32.5$ and $-0.5 \leqslant y, z \leqslant 0.5$. The tetrahedral and hexahedral grids contain 54,648 and $260 \times 4 \times 4$ cells, respectively. To determine the time to locate particles as they are displaced over a distance d, they are initially placed at $x = -d/2$ and at randomly chosen positions in the range of the y- and z-coordinates.

The algorithms considered in this study are the present algorithm (in its fast and robust versions) and a generalization of the algorithm of Löhner and Ambrosiano [8] (hereafter abbreviated as LA). It should be noted that while each algorithm was programmed carefully, the resulting routines were not optimized in any detail. The modified LA algorithm was implemented based on the numerator of Eq. (8) rather than determining the element shape functions, i.e., a given particle is passed to the cell adjacent to the face with the largest negative right-hand side of Eq. (8). The motivation for using the modified algorithm is that the original LA algorithm requires inverting a 3×3 matrix for each cell and a matrix-vector product for each particle in a cell and is applicable only to tetrahedral cells. The modified LA algorithm is applicable to any cell type and gives an operation count which is substantially lower. Note that the modified algorithm does not return as soon as a negative right-hand side of Eq. (8) is encountered. This is consistent with the original LA algorithm in that all shape functions must be known before it can be determined which of the neighboring cells is to be tested next. The operation counts per particle of the original and modified LA algorithms and the present algorithms on tetrahedral grids are summarized in Table 1, where N_c^{LA} and N_c^{T} denote the number of cells traversed in locating one particle by the LA and present algorithms, respectively. It should be noted that in general $N_c^{\mathrm{LA}} \geqslant N_c^{\mathrm{T}}$. Based on Table 1, the modified LA algorithm may be expected to be about twice as fast as the original algorithm.

Because the number of faces considered by the fast and robust versions of the present algorithm is dependent not only on the cell type and geometry but also on the trajectory, it is difficult to derive precise estimates. Instead, Table 1 presents the minimum and maximum number of operations, which correspond to one or three face intersections, respectively. For the original and modified LA algorithms, the minimum and maximum number of operations are identical because all faces are tested.

Table 1
Operation count per particle for various particle-localization algorithms on tetrahedral grids

Algorithm	Minimum	Maximum
LA, original	$N_c^{LA}(3D + 27M + 51A)$	
LA, modified	$N_c^{LA}(12M + 20A)$	
Present, fast	$N_c^{T}(1D + 6M + 7A)$	$N_c^{T}(3D + 18M + 21A)$
Present, robust	$N_c^{T}(1D + 15M + 22A)$	$N_c^{T}(3D + 21M + 26A)$

N_c^{LA} and N_c^{T} denote the number of cells traversed by algorithms; typically $N_c^{LA} \geqslant N_c^{T}$. D, M, and A denote the operations of division, multiplication, and addition (or subtraction), respectively.

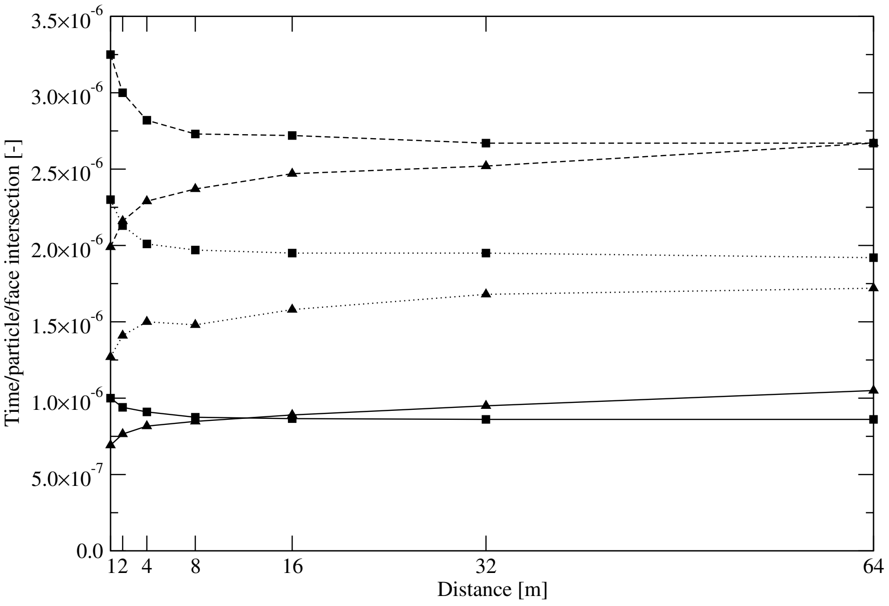

Fig. 10. Results of performance study for tetrahedral and hexahedral grids. Lines indicate algorithm and symbols indicate grid type. (—) Fast version of present algorithm; (- - -) robust version of present algorithm, (· · ·) modified algorithm of Löhner and Ambrosiano [8]; (▲) tetrahedral grid; (■) hexahedral grid.

The results of the performance study are summarized in Fig. 10. The timings are normalized by the number of particles being tracked and the number of face intersections, which means that scalable algorithms lead to horizontal lines. The results demonstrate that the fast version of the present algorithm is approximately twice as fast as the modified LA algorithm on both tetrahedral and hexahedral grids irrespective of the distance over which particles are tracked. The robust version of the algorithm, in turn, is about 2.5 times slower than the fast version. These trends are consistent with the operation counts presented in Table 1, where it should be noted that for these tests, $N_c^{LA} \approx N_c^{T}$.

Additional studies determined that locating particles for a given displacement including a single wall reflection per particle resulted in an increase of about 10% compared to the time required to locate particles for the same displacement without wall reflections.

5.1.2. Non-planar faces

The performance of the present algorithm for non-planar faces is assessed by comparing it to the fast algorithm for planar faces on a hexahedral grid with 54,720 cells and 100,000 particles. For the non-planar version of the present algorithm, the interior vertices are displaced by 1% of the grid spacing. (The amount of non-planarity does not affect the performance.) The algorithm for non-planar faces was found to be about 3.3 times slower than the fast algorithm for planar faces irrespective of the displacement.

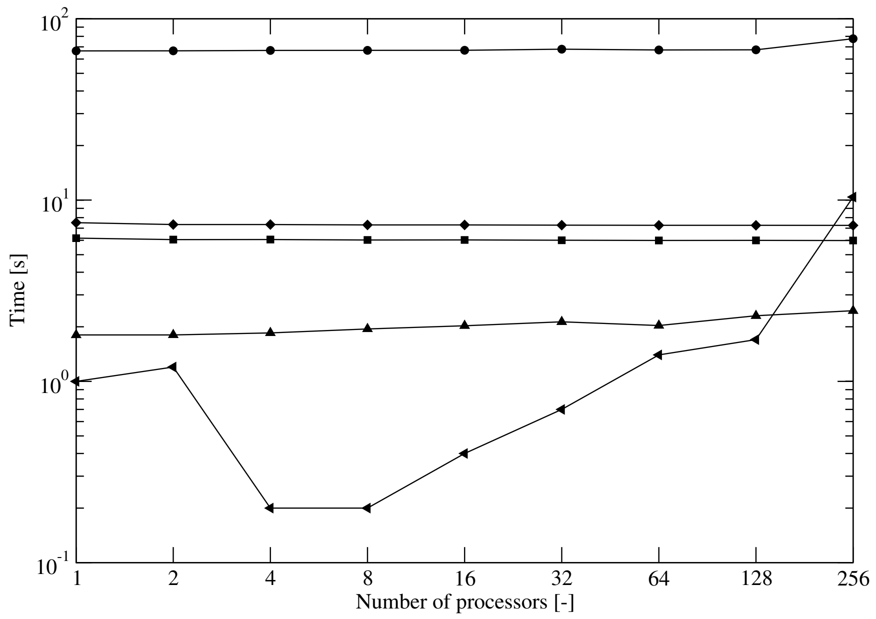

Fig. 11. Results of parallel scalability study on an IBM SP5 machine at the Lawrence Livermore National Laboratory. (●) Run time of solver (excluding I/O); (■) update of particle position, mass, momentum, and energy; (♦) particle-localization algorithm; (▲) communication driver (excluding MPI calls); (◀) reduction operation to determine total number of particles to be communicated.

5.2. Parallel performance

The parallel performance of the robust version of the present algorithm is assessed by studying a scaled problem with 32^3 hexahedral cells per partition. The partitions are stacked in the x-direction to generate a square-pencil-like geometry as the number of partitions increases. A uniform flow field is specified with zero velocity components in the y- and z-directions. Periodic boundary conditions are employed in the x-direction to ensure that the ratio of communication to computation of the underlying Eulerian solver is constant across the partitions. The Eulerian solver was previously shown to scale perfectly up to 1920 processors of the IBM Blue Gene machine at the San Diego Supercomputing Center. Each partition is assigned 16,384 particles at random locations within its bounding box. The random locations of the particles within a partition means that the number of particles being communicated between partitions at a given instant is only approximately constant, thus introducing a slight load imbalance.

The scaling of key routines up to 256 processors on an IBM SP5 machine at the Lawrence Livermore National Laboratory is shown in Fig. 11. It can be seen that the scaling is nearly perfect except for a slight increase in the run time from 128 to 256 processors. This increase is caused by the non-scalable behavior of the collective communication needed to determine the total number of particles to be communicated (step 2 in the description of the parallel algorithm in Section 4.3).[1] Other key routines, such as the localization algorithm, the update of the position, mass, momentum, and energy of the particles, and the communication driver are scaling very well. It must be emphasized that the collective communication is necessary to treat correctly particles which cross more than one partition between old and new positions and would be required by localization algorithms other than the present one.

[1] Also, it should be noted that with the exception of the run on 256 processors, only seven of the eight processors on a node were used. Previous studies indicated that using all processors on a node can lead to non-scalable behavior because of interference from the operating system.

6. Sample application

Results from the application of our algorithm to the Space Shuttle solid-propellant rocket motor are shown in Fig. 12. This motor is characterized by the star grain at the head end, three inter-segment slots with inhibited propellant surfaces, and a submerged nozzle. The propellant is enriched with aluminum particles to increase the specific impulse and to damp combustion instabilities. As the propellant burns, aluminum droplets are entrained into the flow. The grid consists of about 1.8 million cells and is split into 256 partitions. The static temperature and particle field are depicted in Figs. 12a and b at a representative time after quasi-steady conditions are established. About 4 million particles are located in the rocket motor. The effect of the geometry on the particle distribution is illustrated in Figs. 12c and d. The influence of the star grain on the particle distribution is still noticeable in the nozzle.

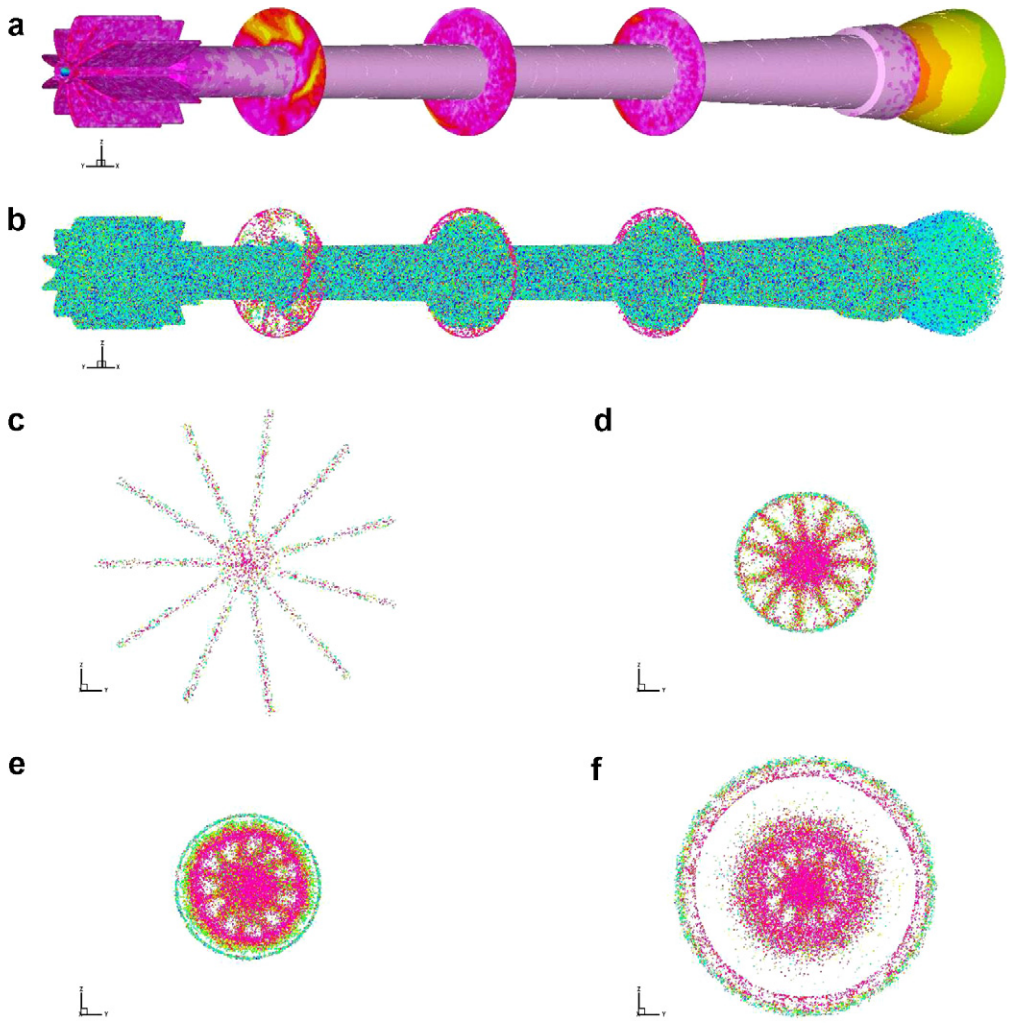

Fig. 12. Example solution for multiphase flow in Space Shuttle solid-propellant rocket motor at quasi-steady conditions. (a) Static temperature. (b) Particle field. (c) Particle distribution in star grain. (d) Particle distribution in first propellant segment. (e) Particle distribution in third propellant segment. (f) Particle distribution in nozzle.

7. Conclusions

An efficient and robust particle-localization algorithm for unstructured grids was presented. Given a particle position and the cell containing that position, the algorithm determines the cell which contains a nearby position. The algorithm is based on tracking a particle along its trajectory by computing the intersections of the trajectory and cell faces.

Compared to previously published particle-localization algorithms, the new algorithm has several advantages. First, it can be applied to grids consisting of arbitrary polyhedral cells. The cells may consist of planar and/or non-planar faces. Second, the algorithm is not limited to small particle displacements. In principle, it can track a particle for an arbitrarily long distance without requiring a fall-back algorithm. For this reason, the algorithm may be viewed as a general spatial search algorithm. Third, the interaction of particles with boundaries is dealt with correctly and naturally because the algorithm detects that an intersection with a boundary face occurs. The algorithm can take appropriate action depending on the boundary condition applied to that face. Fourth, the algorithm is more efficient than other published algorithms.

A modified version of the present algorithm was presented which can detect inconsistencies and take appropriate corrective action at the expense of additional operations. With these modifications, the computational cost becomes comparable to other published algorithms which cannot detect inconsistencies without resorting to fall-back algorithms such as exhaustive or Octree search.

Acknowledgments

This work was funded by the US Department of Energy through the University of California under Subcontract No. B523819. The authors acknowledge Dr. S.V. Apte for helpful discussions about the algorithm used by Apte et al. [1].

References

[1] S.V. Apte, K. Mahesh, P. Moin, J.C. Oefelein, Large-eddy simulation of swirling particle-laden flows in a coaxial-jet combustor, Int. J. Multiphase Flow 29 (2003) 1311–1331.

[3] J.U. Brackbill, H.M. Ruppel, FLIP: a method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions, J. Comput. Phys. 65 (1986) 314–343.

[5] L. Guibas, D. Salesin, J. Stolfi, Epsilon geometry: building robust algorithms from imprecise computations, in: Proceedings of the Fifth Annual ACM Symposium on Computational Geometry, 1989, pp. 208–217.

[6] G. Li, M.F. Modest, An effective particle tracing scheme for structured/unstructured grids in hybrid finite volume/PDF Monte Carlo methods, J. Comput. Phys. 173 (2001) 187–207.

[7] R. Löhner, Robust, vectorized search algorithms for interpolation on unstructured grids, J. Comput. Phys. 118 (1995) 380–387.

[8] R. Löhner, J. Ambrosiano, A vectorized particle tracer for unstructured grids, J. Comput. Phys. 91 (1990) 22–31.

[10] S. Ramsey, K. Potter, C. Hansen, Ray bilinear patch intersections, J. Graph. Tools 9 (3) (2004) 41–47.

[11] D. Seldner, T. Westermann, Algorithms for interpolation and localization in irregular 2D meshes, J. Comput. Phys. 79 (1998) 1–11.

[12] S. Subramaniam, D.C. Haworth, A probability density function method for turbulent mixing and combustion on three-dimensional unstructured deforming meshes, Int. J. Engine Res. 1 (2) (2000) 171–190.

[13] T. Westermann, Localization schemes in 2D boundary-fitted grids, J. Comput. Phys. 101 (1992) 307–313.

[14] Q. Zhou, M.A. Leschziner, An improved particle-locating algorithm for Eulerian–Lagrangian computations of two-phase flows in general coordinates, Int. J. Multiphase Flow 25 (1999) 813–825.