COMMUNICATIONS IN NUMERICAL METHODS IN ENGINEERING
Commun. Numer. Meth. Engng 2009; **25**:263–273
Published online 3 June 2008 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cnm.1128

Particle tracking in unstructured, arbitrary polyhedral meshes for use in CFD and molecular dynamics

Graham B. Macpherson[1,*,†], Niklas Nordin[2] and Henry G. Weller[3]

[1]*Department of Mechanical Engineering, University of Strathclyde, Glasgow G1 1XJ, U.K.*
[2]*Scania CV AB, SE-151 87 Södertälje, Sweden*
[3]*OpenCFD Ltd., 9 Albert Road, Reading RG4 7AN, U.K.*

SUMMARY

Many classes of engineering fluid dynamics simulation require the tracking of discrete elements, for example, dispersed particles in a solvent, a spray of diesel injected into an internal combustion engine, or the dynamics of granular materials. Realistic simulations often require complex, 3D geometries, generated from CAD models and meshed with unstructured polyhedra, which may be deforming, in motion or spatially decomposed for parallel computation. We present an algorithm to track the motion of particles in such geometries which is designed to be computationally efficient and robust in imperfect 3D meshes where small holes or overlaps are present. It has been applied to a wide range of engineering problems ranging from injected fuel sprays in internal combustion engines to molecular dynamics modelling of nanoscale flows. Copyright 2008 John Wiley & Sons, Ltd.

Received 20 July 2007; Revised 28 January 2008; Accepted 22 February 2008

KEY WORDS: particle tracking; molecular dynamics; droplet spray; unstructured mesh; OpenFOAM

INTRODUCTION

Tracking the motion of simulated particles is straightforward in a geometry represented by a mesh where an algebraic expression can be used to determine which cell a particle occupies, such as a simple Cartesian grid. A particle may be moved along the trajectory for its current time step without regard to the underlying mesh (in some cases no mesh is required); it discards the information about which cell it previously occupied because it can easily and quickly determine which cell it occupies in its new position.

*Correspondence to: Graham B. Macpherson, Department of Mechanical Engineering, University of Strathclyde, Glasgow G1 1XJ, U.K.
†E-mail: graham.macpherson@strath.ac.uk

Contract/grant sponsor: Miller Foundation
Contract/grant sponsor: James Weir Foundation

When the geometry is complex, comprising a mesh of unstructured, arbitrary polyhedral cells, as encountered in realistic engineering applications, discarding and redetermining which cell a particle is in is no longer computationally efficient because it requires a time-consuming search of the mesh. It is more efficient to carefully track where and when particles cross mesh faces and change cell. Maintaining information about which cell a particle occupies is necessary for tracking the particle in subsequent time steps and is also used where the spatial relationship between particles in different cells is important—calculating intermolecular forces in molecular dynamics (MD), for example [1].

We present a tracking algorithm for 3D simulations, suitable for parallel computation, that is robust despite the imperfect face geometries encountered in complex, unstructured meshes. It has been implemented in OpenFOAM [2], an open-source C++ computational fluid dynamics (CFD) toolbox. In OpenFOAM version 1.4.1 (the current version available at the time of writing) the implementation of the algorithm can be found in the particle class in the `$FOAM_SRC/lagrangian/basic/particle` directory.

The algorithm is generic: applicable to any type of ‘particle’, having been used for (i) CFD, (ii) granular flow simulations, (iii) ray tracing, and (iv) molecular modelling:

(i) particles dispersed in a solvent; fluidized bed simulations; tracking fuel droplet dynamics in diesel spray combustion simulations, see Figure 1;
(ii) hopper emptying [3]; chromatography column packing [4], see Figure 2;
(iii) radiative heat transfer simulation using the discrete transfer radiation model;
(iv) implementing the motion of molecules in MD simulations [1] and computational molecules in the direct simulation Monte Carlo [5] technique.

Specialized particle types are derived from the basic particle class and inherit the tracking functionality automatically.

Existing tracking algorithms

A current and detailed review of existing, published particle tracking algorithms can be found in Reference [6] and the references therein. The algorithm presented here bears some resemblance

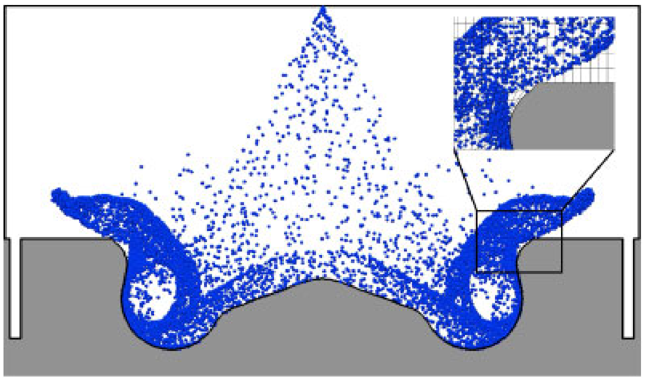
Figure 1. A slice through the centre of a simulation of a diesel spray in an internal combustion engine. Fuel droplets have their dynamics coupled to a continuum CFD solver and are modelled as particles that can evaporate, break-up, and coalesce.

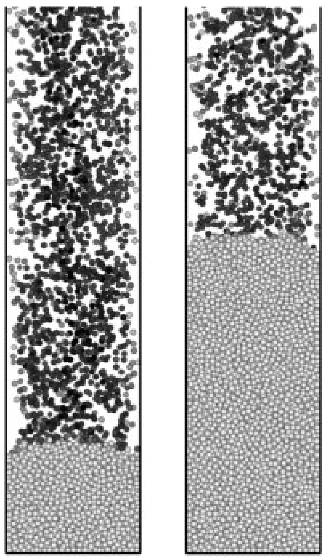
Figure 2. Two snapshots of a simulated chromatography column filling with beads, from [4].

to that presented in Reference [6] because it involves tracking the motion of particles from cell to cell by calculating and identifying face crossings. There is, however, a flaw in the algorithms described and reviewed in this reference: a particle may be assigned to (think that it occupies) a cell that is inconsistent with its position. As will be shown, this situation occurs when considering the imperfect mesh geometry encountered in realistic meshes of polyhedral cells.

A similar situation where this flaw occurs is mentioned in Reference [6]: the '... highly improbable situation... of having a particle trajectory crossing through a cell vertex'. The algorithm makes an assumption about which face the particle actually crosses (there is a choice of at least three for a vertex of a 3D cell) and therefore which cell the particle enters. In large simulations, involving millions of particles and hundreds of thousands of cells, for example [1], such 'highly improbable situations' are observed. They cause the simulation to crash or freeze if the assumed face crossing is wrong, unless a computationally expensive check is performed. The algorithm we present efficiently mitigates this flaw by construction.

Reference [7], which has been published since Reference [6], proposes a new tracking algorithm. The authors specify the desirable features of a tracking algorithm:

- efficient determination and tracking of the location and cell occupancy of a particle;
- the ability to calculate fluid–particle (or Eulerian–Lagrangian) coupling terms, based on the time a particle spends in each cell during its trajectory;
- no restriction on the particle time step should be imposed;
- particle tracking in moving and deforming meshes.

The algorithm we present possesses these features. The algorithm described in Reference [7] involves locating which cell a particle occupies by mesh searching at the start of each tracking time step. Our tracking algorithm automatically maintains information stating which cell a particle occupies; hence, there is no need to locate the particle at the beginning of every time step. We consider that our algorithm will, in general, provide a faster solution. The algorithm as described in Reference [7] only applies to 2D simulations, although this is not explicitly stated. If implemented in 3D, it would suffer from the same flaw as described above, although it is not as dangerous because which cell a particle occupies is redetermined at the start of each tracking time step, as long as the particle stays within the domain. If it strays outside of the domain then this algorithm will be unable to locate it at the beginning of next tracking time step.

BASIC PARTICLE TRACKING ALGORITHM

Consider the situation in Figure 3, where a particle is located at position **a** and is required to move to **b**, determined by solving the equation of motion for the particle. The motion can be performed as a series of individual tracking events, each ending when the particle either crosses a face of a cell or arrives at the final destination. The particle must move to position **p**, where the line **ab** intersects face 2, then change to the neighbouring cell. It will carry on to $\mathbf{p}'$, change cell again, before finally arriving at **b**. For the first part of the motion, **a** to **p**, the position **p** is found using

$$\mathbf{p}=\mathbf{a}+\lambda_{\mathrm{a}}(\mathbf{b}-\mathbf{a}) \tag{1}$$

where λ_{a} is the fraction along the line **ab** where the intersection occurs with the plane defined by a face centre, $\mathbf{C}_{\mathrm{f}}$ and face normal vector, **S**. Therefore, because **p** lies on this plane

$$(\mathbf{p}-\mathbf{C}_{\mathrm{f}})\cdot\mathbf{S}=0 \tag{2}$$

Substituting Equation (1) into Equation (2) gives

$$\lambda_{\mathrm{a}}=\frac{(\mathbf{C}_{\mathrm{f}}-\mathbf{a})\cdot\mathbf{S}}{(\mathbf{b}-\mathbf{a})\cdot\mathbf{S}} \tag{3}$$

Equation (3) is applied to calculate a value of λ_{a} for *each* face of the cell that the particle currently occupies, using each face's own $\mathbf{C}_{\mathrm{f}}$ and **S** vectors. The face that the particle actually crosses is that which has the lowest value of λ_{a} in the interval $0\leqslant\lambda_{\mathrm{a}}\leqslant 1$. For the example shown in Figure 3, faces 1 and 2 have λ_{a} values in this interval, with face 2 having the lower value, giving the correct face to be crossed.

The particle is moved to **p** and the particle's cell occupancy information is changed to the neighbouring cell using the mesh connectivity: cell A, where the particle started, crosses face 2, cell A shares face 2 with cell C, therefore, the particle changes occupancy to cell C. Using this connectivity information means that computationally expensive mesh searching is avoided when maintaining cell occupancy information. A search of the mesh is only required once in the simulation to identify which cell a particle occupies: either at the start of the simulation or when a new particle is introduced. The tracking algorithm automatically keeps track of which cell a particle occupies or 'belongs to'.

The next tracking event is then executed in the same way: λ_{a} is calculated for each of the faces of the new cell, and the particle is tracked to the next face it has to cross, or to the final destination, if that is in the new cell.

If no face satisfies $0\leqslant\lambda_{\mathrm{a}}\leqslant 1$, then **b** must be inside the same cell as the particle started in, and it can be moved directly to the end point of the motion. For a particular face, $\lambda_{\mathrm{a}}<0$ corresponds to the particle motion from **a** to **b** being away from the face; $\lambda_{\mathrm{a}}>1$ corresponds to the motion from **a** to **b** being towards the face, but not reaching it. These two cases are shown in Figure 4.

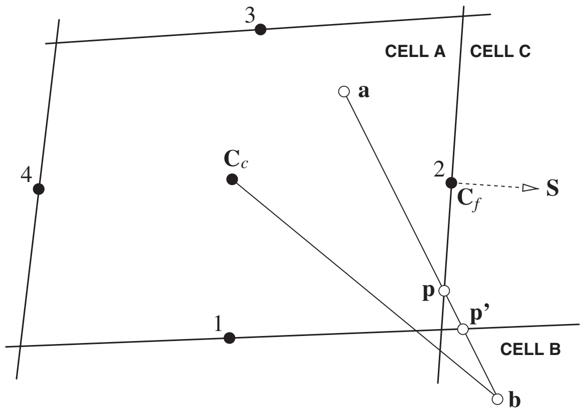

Figure 3. A particle moving from position **a** to **b**, crossing two faces at **p** and $\mathbf{p}'$ and changing cell twice. The cell comprises four numbered faces, each of which stores face centre, $\mathbf{C}_\mathrm{f}$, and face normal, **S**, vectors. The cell centre, $\mathbf{C}_\mathrm{c}$, is shown.

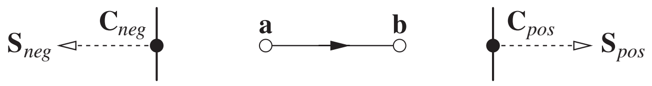

Figure 4. Face 'neg' yields a negative value of λ_a and face 'pos' yields a value of λ_a greater than 1.

Deficiencies of the basic algorithm related to non-planar cell faces

Each face in the mesh stores a vector describing its face centre position and face normal vector; these define the plane of the face used when calculating if a particle crosses it. Where a face comprises more than three vertices, all of the vertices will not necessarily lie on a single plane. Therefore, the mesh stores face centroid and integrated area normal vectors which represent the *effective* plane of the face. Non-flat faces lead to a representation of the mesh that is no longer a set of space-filling cells. An example of this can be seen in Figure 5, where face 2 has an exaggerated twist; plane 2 is the effective plane based on the centroid and integrated area normal vector. Plane 2 does not meet exactly with face 1, which is perfectly planar in this example.

There is a possibility of losing track of particles when they cross a face close to a vertex. In Figure 5, a particle moving from **a** to **b** or $\mathbf{b}'$ will cross face 1, and, by mesh connectivity, change cell. The particle will, however, as it crosses face 1, be located physically on the wrong side of plane 2 to be consistent with occupying this new cell. The implications of this are clearer when considering the situation shown in Figure 6, which is a 2D representation showing a situation *analogous* to that encountered during an event such as that shown in Figure 5. Figures 5 and 6 are not directly related because it is only possible for the non-planar face issue to occur, and be represented, in 3D.

In Figure 6 imagine that the particle has crossed face 1, leaving cell D, and, by mesh connectivity, its cell occupancy information is set to cell A. However, the position, **p**, that the particle has tracked to lies outside of cell A. How the basic tracking algorithm responds depends on the final destination of the particle. The final destination can be either

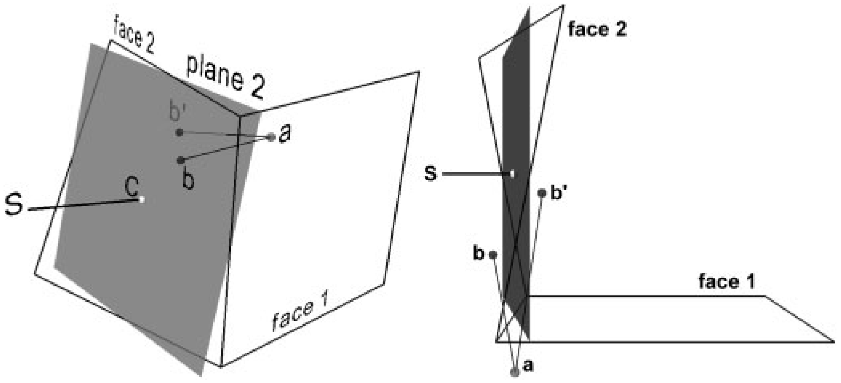

Figure 5. A mesh with a twisted face (face 2) which differs from its effective plane (plane 2). The figure on the right is a top-down view of the figure on the left.

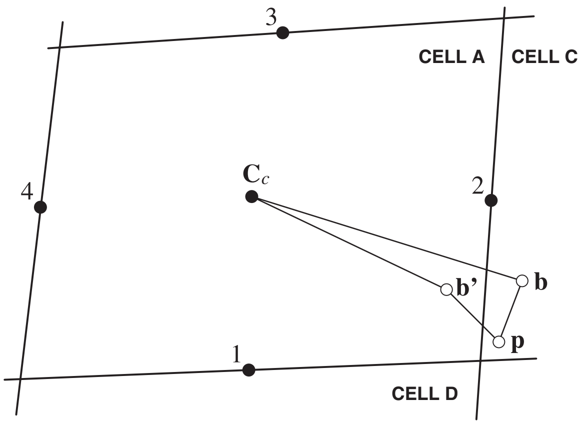

Figure 6. An analogous situation to a particle tracking across a face close to a vertex, in the vicinity of a non-planar face. The particle's post tracking event position, **p**, is located physically outside of the cell determined by mesh connectivity information in the previous tracking event: cell A in this case.

Case **b**: inside the cell that is physically consistent with its current position (cell C). Values of λ_a calculated using **p** and each face of cell A (the cell that the particle's cell occupancy information states it should be in) are $\lambda_a<0$ or $\lambda_a>1$, which is equivalent to the particle not needing to change the cell. The particle will move to **b**, but it will not change its occupancy information from cell A to cell C.

Case $\mathbf{b}'$: inside the cell that the particle's cell occupancy information has been set to by mesh connectivity during the previous tracking event (cell A). The values of λ_a calculated using **p** and each face of cell A will determine that face 2 is intersected and the particle will change its occupancy information from cell A to cell C, but will be tracked into the physical region of cell A.

In either case, the particle is not physically located in the cell that its cell occupancy information states it should be, and is effectively lost to the simulation. Correcting this problem would require a computationally expensive mesh search *every time a particle changes cell* in order to check that it moved into the cell it should have, and finding it if not.

MODIFIED TRACKING ALGORITHM

With reference to Figure 3, if **b** is outside the starting cell, then a particle may be tracked from any position inside the cell and will cross the planes of the same faces that it would have if it had started at **a**. Therefore, rather than using **a** to find which face the particle will hit when moving from **a** to **b**, the cell centre, $\mathbf{C}_c$, can be used to calculate λ_c by replacing **a** with $\mathbf{C}_c$ in Equation (3)

$$\lambda_c = \frac{(\mathbf{C}_f - \mathbf{C}_c)\cdot\mathbf{S}}{(\mathbf{b}-\mathbf{C}_c)\cdot\mathbf{S}} \tag{4}$$

Using λ_c to find which face the particle will hit will result in $0 \leqslant \lambda_c \leqslant 1$ for faces 1 and 2 in Figure 3, as before. If $\lambda_c < 0$ or $\lambda_c > 1$ for all faces, then again as before, **b** must be inside the cell. It is necessary to calculate λ_a for all faces whose planes are crossed (where $0 \leqslant \lambda_c \leqslant 1$, faces 1 and 2 in this case) and the lowest value of λ_a determines which face was actually hit. This value of λ_a is stored for use in the remainder of the calculation. The complete algorithm is summarized in Algorithm 1.

The final components of the modified algorithm, and the reasons for the modification, become clear when returning to the two cases shown in Figure 6:

Case **b**: face 2 produces $0 \leqslant \lambda_c \leqslant 1$, but $\lambda_a < 0$, meaning that the **pb** trajectory points away from face 2. Here the particle is not moved, but the cell occupancy change determined by λ_c is performed, i.e. face 2 has been crossed. The particle changes occupancy from cell A to cell C and continues tracking to **b**. If the **pb** trajectory had pointed towards face 2 (but **b** was still in cell C), then $\lambda_a > 1$, and the particle is moved to **b** and the cell occupancy is changed to cell C as above. In both situations this is implemented by moving the particle to **p**, given by

$$\mathbf{p} = \mathbf{a} + \lambda_m(\mathbf{b}-\mathbf{a}), \quad \lambda_m = \min(1, \max(0, \lambda_a)) \tag{5}$$

Case $\mathbf{b}'$: all values of $\lambda_c < 0$ or $\lambda_c > 1$; hence, the particle is moved to $\mathbf{b}'$ and the cell occupancy stays as cell A.

Algorithm 1 Complete tracking algorithm.

```
while the particle has not yet reached its end position at b do
    find the set of faces, $F_i$ for which $0 \leqslant \lambda_c \leqslant 1$
    if size of $F_i = 0$ then
        move the particle to the end position
    else
        find face $\mathscr{F} \in F_i$ for which $\lambda_a$ is smallest
        move the particle according to Equation (5) using this value of $\lambda_a$
        set particle cell occupancy to neighbouring cell of face $\mathscr{F}$
    end if
end while
```

It is possible to reduce the problems created by non-planar faces by decomposing each face into triangles and applying the basic algorithm to each of these sub-faces [8]. This is, however, computationally more expensive than the modified algorithm above, and in practice, particles are still lost from the simulation due to numerical rounding errors, especially in moving meshes (see below). Rounding errors can give rise to the same issue as encountered with non-planar faces, i.e. the particle's position is inconsistent with its cell occupancy information. The modified algorithm accommodates this, and as such rounding errors do not cause tracking failures or problems.

Concave cells

There is a possibility that the modified algorithm can enter an infinite loop if the cell it occupies is concave. Consider the example in Figure 7; as the particle moves from **a** to **b** it does not need to leave (concave) cell A. It will, however, calculate an intersection with the plane defined by face 1 at **p**, and, by mesh connectivity, change occupancy to cell B. It is, however, moving away from face 2 in cell B, and will change occupancy to cell C, where it is moving away from face 3, and will change back to cell A and the process will start again, generating an infinite loop. Meshes using this algorithm must have concave cells decomposed into smaller convex cells.

Moving meshes

The algorithm as described is able to track particles in meshes that are moving by simply altering how λ_a is calculated, provided that the mesh does not move too far in a single step relative to the particle. Assuming that the mesh is moved before the particle tracking occurs, the mesh must not move so far as to place the particle more than one cell away from the cell it started in. It is assumed that particle tracking time steps are short in comparison with the rate of mesh motion, and as such the motion of the mesh during the time step can be assumed to be at a constant velocity. Figure 8 shows a single tracking event (not necessarily a full time step) where a particle is attempting to track from **a** to **b** across a face in the mesh that is moving from a cell centre, face normal pair, $\mathbf{C}_s, \mathbf{S}_s$, at the start of the tracking event, to an end state, $\mathbf{C}_e, \mathbf{S}_e$. The particle will

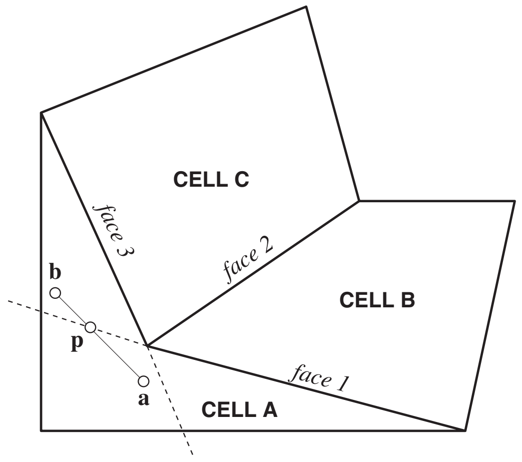

Figure 7. A particle moving in concave cell A from **a** to **b** will become stuck in an infinite loop.

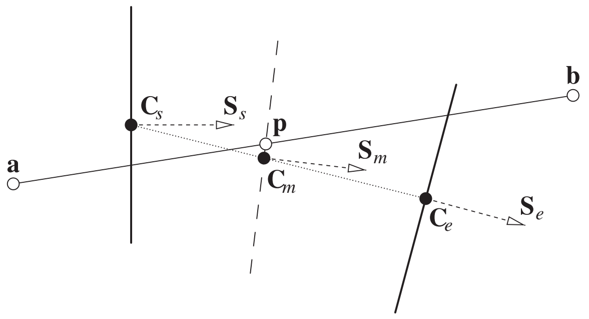

Figure 8. A particle tracking across a face that is translating and rotating.

intersect the moving face at $\mathbf{p}$, when the face is part-way through its motion at $\mathbf{C}_\mathrm{m}, \mathbf{S}_\mathrm{m}$. Given that the velocity of the particle and the linear and rotational velocities of the face are all considered constant throughout the tracking event, and that the mesh and particle are moving over the same time interval, then the same value of λ_a applies to determining the face centre and normal vectors at intersection as for the particle motion in Equation (1), i.e.

$$\mathbf{C}_\mathrm{m} = \mathbf{C}_\mathrm{s} + \lambda_\mathrm{a}(\mathbf{C}_\mathrm{e} - \mathbf{C}_\mathrm{s}) \tag{6}$$

$$\mathbf{S}_\mathrm{m} = \mathbf{S}_\mathrm{s} + \lambda_\mathrm{a}(\mathbf{S}_\mathrm{e} - \mathbf{S}_\mathrm{s}) \tag{7}$$

Substituting Equations (6) and (7) for $\mathbf{C}_\mathrm{f}$ and $\mathbf{S}$ into Equation (3) results in a quadratic in terms of λ_a

$$A_2\lambda_\mathrm{a}^2 + A_1\lambda_\mathrm{a} + A_0 = 0 \tag{8}$$

where

$$A_2 = ((\mathbf{b}-\mathbf{a}) - (\mathbf{C}_\mathrm{e} - \mathbf{C}_\mathrm{s}))\cdot(\mathbf{S}_\mathrm{e} - \mathbf{S}_\mathrm{s})$$

$$A_1 = ((\mathbf{b}-\mathbf{a}) - (\mathbf{C}_\mathrm{e} - \mathbf{C}_\mathrm{s}))\cdot\mathbf{S}_\mathrm{s} + (\mathbf{a} - \mathbf{C}_\mathrm{s})\cdot(\mathbf{S}_\mathrm{e} - \mathbf{S}_\mathrm{s})$$

$$A_0 = (\mathbf{a} - \mathbf{C}_\mathrm{s})\cdot\mathbf{S}_\mathrm{s}$$

Equation (8) can either have two positive, real roots, in which case the root with the smallest magnitude is chosen, or imaginary roots, meaning the face was not intersected; hence, an (arbitrary) value of $\lambda_\mathrm{a} > 1$ is returned. If the mesh undergoes linear motion only then $\mathbf{S}_\mathrm{e} = \mathbf{S}_\mathrm{s}$ and Equation (8) reduces to

$$\lambda_\mathrm{a} = \frac{-(\mathbf{a} - \mathbf{C}_\mathrm{s})\cdot\mathbf{S}_\mathrm{s}}{((\mathbf{b}-\mathbf{a}) - (\mathbf{C}_\mathrm{e} - \mathbf{C}_\mathrm{s}))\cdot\mathbf{S}_\mathrm{s}} \tag{9}$$

This process is equally valid for finding λ_c by substituting Equations (6) and (7) into Equation (4).

BOUNDARY INTERACTIONS AND PARALLELIZATION

At every face crossing, a check is performed to determine whether the face forms part of a boundary or is internal to the mesh. Specific actions can be taken depending on the type of boundary encountered, for example

- cyclic boundary: the particle is physically moved to ‘wrap around’ to the appropriate face on the other side of the cyclic boundary, then continues the tracking step;
- interprocessor boundary: the particle is removed from the current processor and recreated at the appropriate face on the destination processor, where it completes the remainder of its motion;
- solid wall: the velocity of the particle is altered according to the wall model employed, e.g. specularly or diffusely reflected—the particle now travels towards a different destination;
- outlet: the particle is deleted and removed from the simulation.

After a particle encounters a periodic boundary or a solid wall then, due to a position or velocity change, the particle tracks towards a different destination in the next tracking event than the one to which it was moving to at the beginning of the time step. This is compatible with the algorithm as described.

CONCLUSIONS

We have described an algorithm that provides robust and computationally efficient tracking of the motion of particles in unstructured, 3D arbitrary polyhedral mesh geometries. The mesh may be deforming, moving or spatially decomposed for parallel computation. The only restriction on mesh geometry is that cells must be convex. The algorithm is generic and may be applied to a wide range of scientific and engineering problems. The issue of losing track of particles in holes or overlaps in the mesh caused by non-planar faces has been resolved without introducing significant additional computational cost.

ACKNOWLEDGEMENTS

The authors would like to thank Jason M. Reese and Chris J. Greenshields of Strathclyde University (U.K.), and Mattijs Janssens of OpenCFD Ltd. (U.K.) for useful discussions. G. B. M. gratefully acknowledges the financial support provided by the Miller Foundation and the James Weir Foundation.

REFERENCES

1. Macpherson GB, Reese JM. Molecular dynamics in arbitrary geometries: parallel evaluation of pair forces. *Molecular Simulation* 2008; accepted.
2. OpenFOAM: The Open Source CFD Toolbox. Available from: http://www.openfoam.org.
3. Hemph R, van Wachem BGM, Almstedt AE. DEM modeling of hopper flows: comparison and validation of models and parameters. *Fifth World Congress of Particle Technology*, Orlando, FL, 2006.
4. Hemph R, Svensson J, van Wachem BGM, Almstedt AE. Discrete element simulations and experimental validation of particle packing in a 5 mm chromatography column. *Sixth International Conference on Multiphase Flow*, Leipzig, Germany, 2007.
5. Allen J, Hauser T. foamDSMC: an object oriented parallel DSMC solver for rarefied flow applications. *Forty-fifth AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 2007.

6. Chorda R, Blasco JA, Fueyo N. An efficient particle-locating algorithm for application in arbitrary 2D and 3D grids. *International Journal of Multiphase Flow* 2002; **28**(9):1565–1580.
7. Vaidya AM, Subbarao PMV, Gaur RR. A novel and efficient method for particle locating and advancing over deforming, nonorthogonal mesh. *Numerical Heat Transfer Part B*: *Fundamentals* 2006; **49**(22):67–88.
8. Nordin N. Complex chemistry modeling of diesel spray combustion. *Ph.D. Thesis*, Chalmers University of Technology, Gothenburg, Sweden, 2000.