JOURNAL OF COMPUTATIONAL PHYSICS 91, 22–31 (1990)

A Vectorized Particle Tracer for Unstructured Grids

RAINALD LÖHNER

CMEE, School of Engineering and Applied Science, The George Washington University, Washington, DC 20052

AND

JOHN AMBROSIANO

University of California, Lawrence Livermore National Laboratory, Livermore, California 94550

Received May 16, 1989; revised October 4, 1989

A vectorized particle tracer for unstructured grids is described. The basic approach is to use elementary properties of the linear basis functions to search for particles on the grid using the element last occupied as an initial guess. To permit vectorization, a simple binary sort of the particles is performed every timestep such that all particles that have as yet not found their host element remain at the top of the list. In this way, vector-loops can be easily formed. Timings taken from a numerical example indicate that speed-ups of the order of 1:14 can be obtained on vector-machines when using this algorithm.

1. INTRODUCTION

In many applications of computational physics, particles are used in conjunction with finite difference of finite element grids. This strategy is commonly adopted in applications where the physical fields are most conveniently sampled at Eulerian mesh points but certain aspects of transport are best represented by the motion of Lagrangian particles. In fluid dynamics, for example [1, 2], flow fields may be updated on an Eulerian finite difference grid using information from Lagrangian particles which carry mass and, in some cases, momentum and energy. A more widespread example is the use of particle–mesh techniques to model plasmas or particle beams [3, 4]. Here the particles carry electric charge and mass and interact through the Lorentz force equation with the electric and magnetic fields sampled from the grid. These fields in turn are updated in an Eulerian fashion using sources accumulated from the particles.

In most applications, the number of particles required to faithfully reproduce the physical processes at hand is considerable. In fact, some applications have used on

0021-9991/90 $3.00

the order of 10^6 particles. During each timestep of the simulation, the particles change position. But before they can contribute information to the grid or sample field information from it, their new host element or cell must be identified. Therefore they must be traced through the grid. Regular, uniform cartesian grids pose no special problem. The new host cell (i_c, j_c) is easily computed as

$$i_c = \text{Int}\left\{\frac{(x_p - x_{\min})}{(x_{\max} - x_{\min})} N_x\right\} + 1, \qquad j_c = \text{Int}\left\{\frac{(y_p - y_{\min})}{(y_{\max} - y_{\min})} N_y\right\} + 1. \tag{1}$$

Here $\mathbf{x}_p$ is the current location of the particle, $\mathbf{x}_{\min}$, $\mathbf{x}_{\max}$ denotes the range of coordinate values for the current mesh, and N_x, N_y the number of cells in the x and y directions. Int{ } represents truncation to the integer part.

Locating particles on a mesh becomes more complicated as soon as any irregularity is introduced in the mesh structure. Even for rectangular grids with nonuniform spacing, (1) is no longer valid. In the case we wish to consider, namely unstructured grids, the level of irregularity of the mesh is assumed to be much higher. Unstructured grids contain elements of arbitrary size and different mesh points are connected to different numbers of neighbors. On the other hand, unstructured grids can be used to great advantage in modelling geometrically complicated domains because of the ability to place points wherever needed. For the same reason, they are easier to generate automatically and to dynamically refine.

Two basic approaches have been used to trace particles through nonuniform, curvilinear, or irregular meshes. They are mapping and searching. In the first case, a curvinlinear or simply nonuniform mesh is mapped into a logically rectangular grid via coordinate transformations based on interpolation. Particles are then advanced in the logical grid space rather than physical space [5, 6]. This approach has the advantage of avoiding a search procedure which has often been considered to have poor vectorization potential. The disadvantage is that the mapping is only approximate and particle velocities are discontinuous. These properties introduce errors in particle trajectories although the errors can be kept acceptably small by special means [7]. However, mapping is not an option for unstructured grids, since these cannot be embedded in a logically rectangular mesh.

Searching to find a particle's true host element or cell is the most desirable approach from a calculational viewpoint if it can also be made efficient. There are many possible strategies. Among these, the following three appear to be the most promising:

(a) *Use of a Cartesian background grid.* The idea is to superimpose the irregular foreground grid on a regular background grid. The elements of the foreground mesh that cover each cell of the cartesian mesh are stored in a linked list. Given the new particle position $\mathbf{x}_p$, the cell of the Cartesian background grid is obtained from (1). Then, all elements covering this cell are checked to find the new host element for the particle. For the special case in which the foreground grid is logically rectangular, the approximate weights of the particles with respect to the foreground grid can be inferred from the precomputed weights of the foreground

nodes with respect to the background nodes [8]. The main shortcoming of this approach is the inefficiency and inaccuracy that arises when meshes with large variations in element size are employed. If the number of cells employed for the background mesh is too small, the number of elements that need to be checked may become excessive. If, on the other hand, the number of cells employed for the background mesh is too large, the storage overhead may become extravagant. Nevertheless, the method is easy to implement and easy to vectorize.

(b) *Use of tree structures.* Here one attempts to circumvent the problems encountered by the previous method for meshes with large variations in element size by using a hierarchy of cartesian meshes. This is most easily accomplished with quad trees (2D) or octal trees (3D) [9]. The main shortcoming of this procedure is the additional complexity in coding and the essentially scalar nature of the procedure. A tree-based scheme introduces additional indirect addressing that makes it all but impossible to vectorize.

(c) *Use of successive neighbor searches.* The idea is to exploit as a judicious guess the host element before the particle was moved. Then, should the particle no longer be found in that element, the immediate neighbor most likely to contain the particle is searched next [2]. This procedure is repeated until the new host element is found. The algorithm is easy to implement and is very fast. It makes sense to assume that the new particle position is not many elements away from its previous position. This assumption stems from restrictions arising naturally out of considerations of accuracy and stability. Therefore, the number of neighbor-element searches required is typically small. Moreover, the performance does not degrade for meshes with greatly varying mesh sizes. On the other hand, the number of searches required does vary from particle to particle and thus would seem to inhibit vectorization. This paper describes a simple way to overcome the problem.

The remainder of the paper is organized as follows: Section 2 describes the basic scalar algorithm in more depth. Section 3 shows how to vectorize the procedure, while Section 4 describes its implementation on parallel machines. Finally, Section 5 contains some examples and timings.

2. The Scalar Algorithm

The basic strategy for searching neighboring elements to find the new particle host element is based on simple properties of linear basis functions [2, 10, 11]. Therefore, the ideas presented here would be equally applicable for bilinear basis or shape functions on quad elements as well as for linear basis functions on triangles. We will only discuss the case of linear shape functions on triangles. For a triangle such as the one pictured in Fig. 1 with nodes $\mathbf{x}_A$, $\mathbf{x}_B$, $\mathbf{x}_C$, we may introduce the local coordinates ξ, η as follows:

$$\mathbf{x} = \mathbf{x}_A + (\mathbf{x}_B - \mathbf{x}_A)\xi + (\mathbf{x}_C - \mathbf{x}_A)\eta. \tag{2}$$

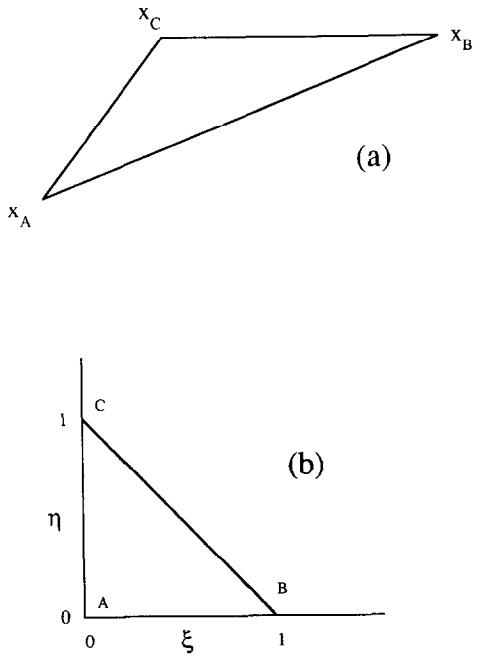

FIG. 1. (a) A triangular element with the coordinates of each node X_A, X_B, and X_C labeled in counterclockwise order. (b) The same element mapped into "natural" coordinates ξ and η.

Thus, if we desire to compute the ξ, η-coordinates of an arbitrary given point $\mathbf{x}_P$, we have

$$\mathbf{x}_P - \mathbf{x}_A = (\mathbf{x}_B - \mathbf{x}_A)\xi + (\mathbf{x}_C - \mathbf{x}_A)\eta. \tag{3}$$

Introducing the notation $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$, we obtain

$$\begin{pmatrix} \xi \\ \eta \end{pmatrix} = \frac{1}{x_{BA}\, y_{CA} - y_{BA}\, x_{CA}} \cdot \begin{bmatrix} y_{CA} & -x_{CA} \\ -y_{BA} & x_{BA} \end{bmatrix} \cdot \begin{pmatrix} x_{PA} \\ y_{PA} \end{pmatrix}. \tag{4}$$

The ξ, η-coordinates are related to the element basis or shape functions N_1, N_2, N_3 in the following way [12]:

$$N_1 = 1 - \xi - \eta, \qquad N_2 = \xi, \qquad N_3 = \eta. \tag{5}$$

Simply put, the basis function at a given node has a value of unity at that node and decreases linearly in the direction of the opposite face at which it is equal to zero by construction. Therefore, a particle at the point $\mathbf{x}_P$ lies in the element if and only if

$$\max(N_1, N_2, N_3) \leqslant 1 \qquad \text{and} \qquad \min(N_1, N_2, N_3) \geqslant 0. \tag{6}$$

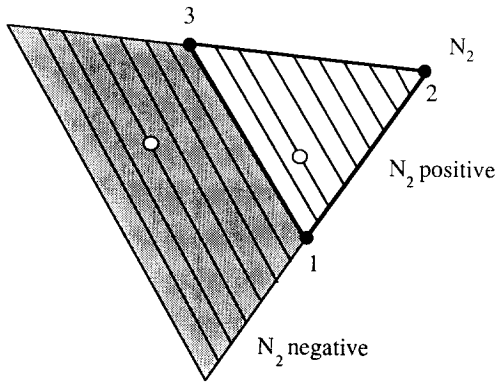

FIG. 2. A triangular element with a portion of the linear basis function associated with the second node shown in contours. The shaded contours indicate that the function becomes negative past the side opposite node 2. Also pictured are two hypothetical particles, one inside and one outside the element.

The strategy is then to evaluate the shape functions of the element where the particle was last known to be, but at its new position. Any negative value means the particle is now absent from that element, and the element adjacent to the face opposite the node with the minimum weight should be searched. Figure 2 shows an arbitrary triangular element with one of its associated shape functions contoured. The shaded portion indicates the region where the shape function becomes negative. Examples of hypothetical particle search paths are given in Fig. 3. Two paths are shown indicating what could occur when a particle is relatively near or relatively

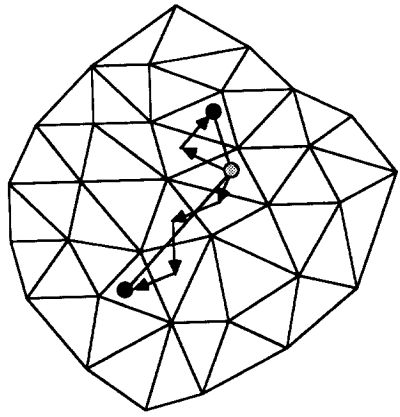

FIG. 3. Hypothetical search paths through an unstructured grid. The gray point is the position where the particle was last known to be. Two different positions are shown, one near and one far from the original position. The arrows show the sequence of elements that would be searched in each case.

far from the original location. The algorithmic steps required to trace one particle are as follows:

S1. Given the current element E, gather the coordinates for this element.

S2. Given the current particle position $\mathbf{x}_p$, evaluate from (4), (5) the shape functions N_1, N_2, N_3 at $\mathbf{x}_p$.

S3. If N_1, N_2, N_3 satisfy (6), the particle is in the current element E and N_1, N_2, N_3 are stored for later retrieval to interpolate charge, fields, etc. Therefore stop.

S4. As N_1, N_2, N_3 do not satisfy (6), we should continue the search with the neighboring element:

Set $E' = E_s(E, I_{\min})$, where $I_{\min}$ denotes the node with the smallest value of the shape-functions N_1, N_2, N_3, and E_s is a two-dimensional integer array which details elements surrounding a given element and which is keyed to the opposite node. A value of zero is substituted when no element exists opposite a given node, e.g., at boundaries. Hence:

If $E' = 0$, the particle lies outside the computational domain. Therefore stop.
If $E' > 0$, set $E \leftarrow E'$ and proceed to step S1.

3. VECTORIZATION

The algorithm described above for one particle may be vectorized by operating on many particles at the same time. Before describing the vectorized version, we make the observation that even the scalar algorithm for one particle requires gather-type operations. For those unfamiliar with the term, a gather operation is one in which values from unordered locations in an array are brought together using another integer array to index the desired locations. The opposite, i.e., distributing values to unordered locations, is called a scatter. Gather operations are typical in any algorithm based on unstructured grids, but this is also the case for particle codes on structured grids. This is because particles and fields are advanced as separate objects. Thus linking them together at any step inevitably requires indirect addressing. Given that we need a certain number of gather operations in any case, the additional gather operations required of the vectorized particle tracer on unstructured grids represent a small increase in computational overhead.

To obtain a vectorized algorithm we must perform steps S1 to S4 as described above in vector mode, executing the same operations on as many particles as possible. The obstacle to this approach is that not every particle will satisfy (6) in one pass. The remedy is to reorder the particles after each pass that all particles that have as yet not found their host element are at the top of the list. Such an algorithm would proceed in the following fashion:

V0. Set the remaining number of particles $N_R = N_T$, where N_T is the total number of particles

V1. Perform steps S1 to S4 in vector mode for all remaining particles N_R.

V2. Write the indices of the M particles that do not satisfy (6) into a list of indices LPCUR (locations 1 to M). If M = 0: stop.

V3. Write the $N_R - N_T$ particles that do satisfy (6) into LPCUR (locations $M+1$ to N_R).

V4. Reorder all particle arrays using the list LPCUR. In this way, all particles that have not yet found their host element are at the top of their respective lists (locations 1 to M).

V5. Set $N_R \leftarrow M$ and go to 1.

Special vector operations involving indirect addressing are required to optimize this algorithm. Optimized gather operations are available on most vector computers to accomplish steps V1 and V4. Vectorized comparison-merge operations are used in place of if statements to optimize steps V2 and V3.

One can reduce the additional memory requirements associated with indirect addressing by breaking up all loops over the N_R remaining particles into subgroups. This is accomplished automatically by using scalar temporaries on register to register machines. For memory to memory machines, a user-specified maximum group vector length must be specified.

4. Multiprocessor Environments

The above algorithm can be ported to multiprocessor environments in the two following ways:

(a) By operating on several groups of particles at the same time; thus each processor would perform V0 to V5 on its group of particles.

(b) By breaking up all loops over the N_R remaining particles into subgroups. Each processor would perform steps V1 to V4 on its group of particles.

It remains to be seen which of the two approaches works better. Our impression is that the individual computer architecture more than anything else will shape the answer to this question.

5. Numerical Example and Timings

The vectorized particle finder described above was coded and applied to trace many particles simultaneously on a typical unstructured grid. The task was stated as follows: discretize the square region $[0, 1] \times [0, 1]$ using triangles. Then place a large number of particles in the mesh. Move all particles the same fixed distance, but in random directions. If a particle lies outside the unit square, move it back inside. For the discretization, we used the mesh shown in Fig. 4 which contains 870

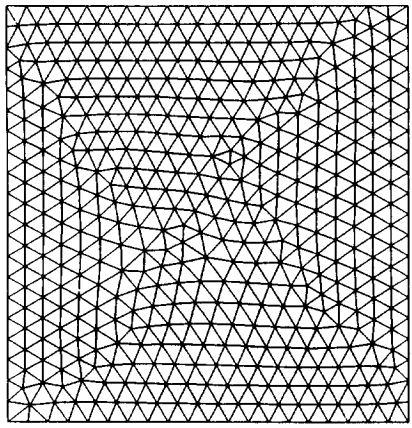

FIG. 4. The unit square discretized into 870 triangular elements comprising an unstructured grid. This is the grid used in the timing example.

triangular elements. At the beginning, the particles were placed uniformly across the mesh forming a square lattice of particles. Figure 5 shows a typical sequence of 10 steps for $N_T = 5 \times 5$ particles. To obtain accurate timings we performed a similar exercise but much larger in scope using $N_T = 10{,}000$ particles to minimize the overhead associated with subroutine-calls and executing 500 timesteps to reduce the start-up overhead. The particles were again moved a distance every timestep of 0.05—about the same as the average element size. On the mesh used (see Fig. 4), at most six elements had to be tested for each particle to locate the new host

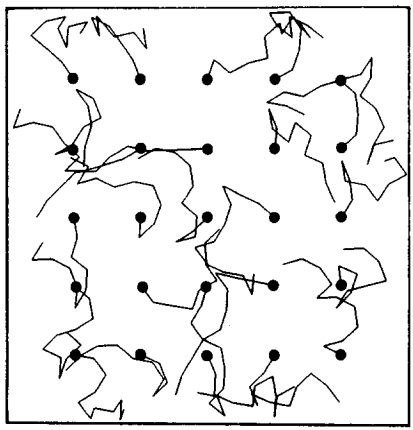

FIG. 5. Examples of random particle trajectories to be traced through the grid of Fig. 4. Each particle moves a distance of 0.05 in an arbitrary direction each step. The 5×5 lattice of points represents the starting positions.

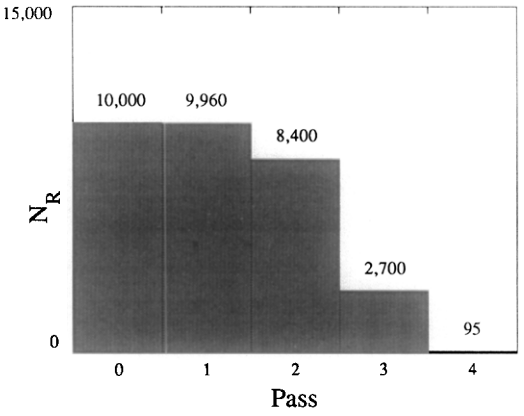

FIG. 6. A histogram of the number of particles remaining out of an original 10,000 after each pass of the vectorized particle tracer. Since each particle is moved about a cell per step in a random direction, random walk statistics dictate that most particles will be found in three passes.

element. However, in most cases only five elements had to be examined. The number of particles remaining after each step, i.e., those that had not yet found their host elements, is pictured in the histogram shown in Fig. 6. Because in this test the particles are moved a fixed distance each time, the histogram remains fairly flat and then drops suddenly reflecting a corresponding drop in the probability that a particle has moved farther than three elements in any given direction. A different distribution of particle displacements could produce a somewhat different result, but in a statistically transparent way.

To obtain timings, two runs were performed on a CRAY-XMP under the operating system COS. In the first one, the CFT compiler options ON = F, OFF = V were invoked. This activates flop-tracing, but deactivates automatic vectorization. In the second run, only the ON = F option was used. The CRAY-XMP used had hardware gather and scatter operations. The timings obtained are listed in Table I.

TABLE I

Timings for Test Run

	Unvectorized	Vectorized
Particle tracing time (s)	257.0	17.6
MFLOP rate recorded	1.11	20.82
Total program time (s)	290.0	21.45
Timing (μs/particle/timestep)	51.4	3.52

The results demonstrate a speed-up factor of 1:14 when using the automatic vectorization. The search time per particle for the vectorized version is also encouraging and suggests that the overhead in using unstructured meshes can be made quite acceptable.

6. Conclusions

A vectorized particle tracer for unstructured grids has been described. The basic approach is to use the elementary properties of linear basis functions to search for particles on the grid using the element last occupied as an initial guess. To permit vectorization, a simple binary sort of the particles is performed every timestep such that all particles that have as yet not found their host element remain at the top of the list. In this way, vector-loops can be easily formed. Timings taken from a numerical example indicate that speed-ups of the order of 1:14 can be obtained on vector-machines when using this algorithm.

Although we have used two-dimensional planar triangles in our example, this algorithm generalizes easily to three dimensions. The elements in that case are tetrahedra, pyramids, or parallepipeds. Linear or bilinear shape functions defined at the element nodes possess the same properties that we have exploited in the examples presented and thus the same procedure is valid.

References

1. F. H. Harlow, "The Particle-In-Cell Computing Method for Fluid Dynamics," in *Methods in Computational Physics*, Vol. 3, edited by B. Alder, S. Fernbach, and M. Rotenberg (Academic Press, New York, 1964), p. 319.
2. J. U. Brackbill and H. M. Ruppel, *J. Comput. Phys.* **65**, 314 (1986).
3. C. K. Birdsall and A. B. Langdon, *Plasma Physics via Computer Simulation* (McGraw-Hill, New York, 1985).
4. R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles* (McGraw-Hill, New York, 1981).
5. J. C. Peterson, Thesis, University of California at Berkeley, 1984 (unpublished).
6. M. E. Jones, "Electromagnetic PIC Codes with Body-Fitted Coordinates," *12th Conference on the Numerical Simulation of Plasmas*, San Francisco, California, Sept. 20–23, 1987) (unpublished).
7. M. E. Jones, R. K. Keinings, W. Peter, and S. C. Wilkes, Los Alamos National Laboratory Report LA-UR-89-1230, 1989 (unpublished).
8. D. Seldner and T. Westermann, *J. Comput. Phys.* **79**, 1 (1988).
9. H. Samet, *Comput. Surveys* **16**, No. 2, 187 (1984).
10. M. J. Fritts and A. T. Drobot, "Plasma Simulations on an Unstructured Grid," in *IEEE International Conference on Plasma Science*, Seattle, Washington, June 6–8, 1988 (unpublished).
11. R. Löhner, *Commun. Appl. Numer. Methods* **4**, 123 (1988).
12. O. C. Zienkiewicz and K. Morgan, *Finite Elements and Approximation* (Wiley, New York, 1983).