Computers and Fluids 159 (2017) 338–355

Contents lists available at ScienceDirect

Computers and Fluids

journal homepage: www.elsevier.com/locate/compfluid

A particle tracking algorithm for parallel finite element applications

Giacomo Capodaglio*, Eugenio Aulisa

Department of Mathematics and Statistics, Texas Tech University, Broadway and Boston Lubbock, TX 79409-1042 United States

A R T I C L E I N F O

Article history:
Received 16 February 2017
Revised 14 July 2017
Accepted 11 October 2017
Available online 13 October 2017

Keywords:
Particle tracking
Point-Locating algorithm
Parallel computing
Unstructured mesh
Finite elements

A B S T R A C T

Numerical simulations of a particle tracking algorithm on parallel unstructured finite element grids are presented. The algorithm is designed to work for both 2D and 3D applications. To determine the position of the particle relative to the mesh, a new point-locating algorithm is proposed. The advection of the particle is performed on the physical domain in order to treat completely unstructured grids. As a consequence, the inversion of the isoparametric finite element mapping is requested. We comply with this demand implicitly using Newton–Raphson's iteration for linear, quadratic, bi-quadratic and tri-quadratic finite elements, and several element geometries, including quadrilaterals, triangles, tetrahedra, wedges and hexahedra. To investigate the performances of the proposed algorithm, results of standard numerical tests are shown, together with a fluid flow application that exemplifies an instance of a particle tracking problem.

1. Introduction

Particle tracking is employed in a variety of computational applications that range from fluid dynamics to biomedicine. When parallel simulations are performed in such applications, the computational domain is divided in subdomains and each subdomain is handled by a given process. In multi-phase flow simulations, the evolution of the interface between different phases can be modeled by tracking specific points that lie on the interface [1–3]. Particle-in-cell type algorithms [4–8] need to follow the movement of particles within the computational grid. Recent works coupled particle tracking with Navier–Stokes equations in laminar and turbulent flow regimes to describe particle deposition in human airways and lungs [9–13]. Another interesting biomedical application is the modeling of magnetic drug targeting for medical treatments [14–18]. Different approaches have been carried out to perform the tracking. A crucial problem that needs to be addressed is the determination of the position of the particle within the computational grid. In the case of structured grids, such an issue can be solved almost trivially. However, for most applications, unstructured grids need to be employed and the design of an efficient point-locating algorithm becomes a challenging task. Many methods have been proposed in the literature to address this matter [19–28]. In this work, we aim to design a particle tracking algorithm that is suitable for 2D and 3D parallel finite element applications with unstructured and possibly curved edges (in 2D) and non-planar faces (in 3D) that arise from a finite element setting. For this, we need a point-locating scheme that can perform well in this context. Algorithms like those developed by Zhou and Leschziner [26] and Chen and Pereira [27] determine if the particle lies inside an element of a 2D grid checking the so called particle-to-the-left condition. This condition works fine for convex polygons but fails to detect a point that lies inside a concave element. Therefore these two methods would fail if we consider elements in 2D with curved edges approximated by line segments. For the same reason the method by Chordá et al. [23] would fail in this situation. The approach of Haselbacher et al. [20] could work in 2D if curved edges were approximated by line segments. Kuang et al. however, pointed out in [24] that the way non-planar faces are dealt with in [20] is not the most straightforward. Kuang et al. addressed the case of non-planar faces on 3D unstructured grids in [24]. The authors provide a summary of the main methods developed in the literature observing how early approaches do not represent reliable methods in case of non-planar 3D faces, due to the presence of what they called a virtual gap. Kenjereš et al. [16] observed that when employing a tetrahedral decomposition of the elements of the grid, the virtual gap problem can be avoided performing multiple decompositions and choosing one that is appropriate. Macpherson et al. [29] developed an algorithm that replaces each non-planar face in 3D by a so called effective plane. They also addressed the problem of concave cells pointing out that their algorithm would fall into an infinite loop in such case and suggested that concave cells should be decomposed into smaller convex cells. To avoid dealing with the virtual gap problem, Kuang et al. proposed in [24] a triangular decomposition of the faces of

* Corresponding author.
E-mail address: giacomo.capodaglio@ttu.edu (G. Capodaglio).

https://doi.org/10.1016/j.compfluid.2017.10.015
0045-7930/© 2017 Elsevier Ltd. All rights reserved.

a given 3D element. However they also observed that, in this way, concave polyhedra are obtained and that there is no generalized point-locating scheme available for a concave grid. As a matter of fact, in their method, the triangle decomposition is performed only on non triangular faces. This means that, for instance, tetrahedral elements will not be involved in this process. Nevertheless when employing isoparametric finite elements, tetrahedral elements can be deformed and have curved faces. Therefore an even more general point-locating scheme than the one described in [24] needs to be appropriately designed to deal with finite element applications. The point-locating procedure we propose has a simple approach and it makes use of what we consider the best features of the algorithms present in the literature. It will be described in detail in Section 2. We now outline the steps that make up the particle tracking algorithm presented in this paper. We start from a given domain discretized with a structured or unstructured mesh and a velocity field known only at the nodes of the grid (in the finite element sense). Such a velocity field may be obtained numerically, for example it could be the solution of the Navier–Stokes equations. Assuming we know the position of a particle in the domain subject to the velocity field at a given initial time t_0, the goal is to determine its position at a time $t > t_0$. The structure of the method can be broken down in three major parts.

- The advection of the particle
  The position $\mathbf{x}_p$ of the particle is related to the velocity field $\mathbf{v}$ by the equation $\frac{d\mathbf{x}_p}{dt} = \mathbf{v}$. We solve this ODE via numerical integration, using Runge–Kutta (RK) schemes of different orders. Even if we implemented the forward Euler method, Heun's method (or RK2), RK3 and RK4, for the numerical simulations that follow we chose RK4 for its greater accuracy. The advected particle moves from its initial position to a new position that may or may not be on the same element from which it started. In a parallel framework, the next element that hosts the particle may not even belong to the process that started the numerical integration. This instance has to be taken in consideration when implementing the advection step by properly organizing the exchange of information between processes.
- The localization of the particle
  To proceed with the next advection step, the velocity field at a given point has to be computed via interpolation using the known values at the nodes of the element. This means that any time the particle moves to a different element, the values of the velocity field have to be updated. Therefore, at every time t, we need to know the element hosting the particle. This is where the point-locating algorithm comes into play. Extra care needs to be exerted when implementing such an algorithm in a parallel setting, as it will be shown in the next section.
- The inversion of the isoparametric mapping
  In a finite element setting, a reference element is employed and the reference system associated to it is called the local reference system. The finite element isoparametric mapping is the function that maps the reference element onto a given element in the global reference system (see Section 3). The local coordinates of the particle are needed to update the value of the velocity field via interpolation. Therefore, after every advection step, using the new global coordinates of the particle we need to determine the local coordinates that correspond to the new position of the particle in the global reference frame. To achieve this, the inversion of the isoparametric finite element mapping is necessary. This is done with an iterative method as it will be explained later in detail.

The paper is structured as follows. In Section 2 we present the parallel point-locating scheme that we employed within the particle tracking algorithm. We describe how the parallel implementation is carried out and what can be done to speed up the search. In Section 3 we describe the way that the Newton–Raphson scheme has been implemented to iteratively find the action of the inverse isoparametric mapping, knowing the nodes of a given element and the coordinates of the particle in the global reference frame. In Section 4 a description of the complete algorithm will be given, together with an explanation on how the different pieces that make it up are related to each other. In Section 5 we describe the results of standards test that we performed in order to validate our algorithm together with a tracking application of particles dispersed in a fluid flow inside a curved pipe. We conclude summarizing the results and discussing future applications.

2. A point-locating algorithm for parallel grids

In this section we describe a new point-locating algorithm for parallel finite element applications. The coordinates of a point in a 2-dimensional or 3-dimensional space are given together with a domain $\mathcal{D}$ discretized using a quasi-uniform finite element triangulation $\mathcal{T}$ [30,31]. With a little abuse of terminology, we will use the term triangulation even when the elements involved are not triangles. The purpose of the algorithm is to determine whether or not the given point lies within the domain and if it does, identify the element of the triangulation it belongs to. The algorithm is designed to work in a parallel environment where the load of elements to be tested for inclusion is divided among different processes. Once a given element has failed the inclusion test, information is produced in order to continue the search moving to a different element that is more likely to include the point. In the particle locating algorithms present in the literature, this is achieved making use of the new position of the particle obtained after the advection step. However, at the beginning of the algorithm, no advection has been performed and so a brute force search throughout all the elements of the grid is required in order to locate the particle. This is really demanding in a 3D case especially when the grid is composed by many elements or if there are many particles to track. Instead of using the trajectory of the particle to determine the next element, we propose an advection independent search path. Instead of using the position of the particle at two different instants of time, we make use of the line (in parametric form) passing through a point appropriately chosen in the interior of the element and the particle to be tracked. This idea was also present in [32] but it was only used as an additional tool of a modified version of their algorithm and not as the official way of determining the search path of the particle like we are doing. With this, after an initial guess element is chosen, we can move inside the mesh until the particle is located, without having to perform any advection and without the need to test all elements of the grid. Our approach is much more general than the one suggested by Vaidya et al. [19] that only works for quadrilateral elements in 2 dimensions. Moreover, our algorithm works for concave elements and does not incur in an infinite loop like in [32]. This represents one of the novelties of the point-locating algorithm of this paper. The choice of the initial guess element together with a way to speed up the search will be discussed in the rest of this section.

The point-locating algorithm is subdivided in two main parts, the actual inclusion test and, in case of a negative result, the parallel transmission of information necessary to test a new element.

2.1. The inclusion test

We now present the inclusion test algorithm. We first describe the situation where the polygon to be tested for inclusion is just a triangle. This is because in the 3D algorithm we decompose each element face in triangles, in order to properly handle situations