Journal of Computational Physics **172,** 766–807 (2001)
doi:10.1006/jcph.2001.6858, available online at http://www.idealibrary.com on IDEAL®

Algorithms for Particle-Field Simulations with Collisions

Hersir Sigurgeirsson,* Andrew Stuart,† and Wing-Lok Wan‡

*SCCM Program, Stanford University, Stanford, CA 94305-4040; and †Mathematics Institute, University of Warwick, Coventry CV4 7AL, United Kingdom; and ‡Department of Computer Science, University of Waterloo, 200 University Avenue West, Waterloo, Ontario N2L 3G1, Canada
E-mail: hersir@hersir.com

Received October 24, 2000; revised June 26, 2001

We develop an efficient algorithm for detecting collisions among a large number of particles moving in a velocity field, when the field itself is possibly coupled to the particle motions. We build on ideas from molecular dynamics simulations and, as a byproduct, give a literature survey of methods for hard sphere molecular dynamics. We analyze the complexity of the algorithm in detail and present several experimental results on performance which corroborate the analysis. An optimal algorithm for collision detection has cost scaling at least like the total number of collisions detected. We argue, both theoretically and experimentally, that with the appropriate parameter choice and when the number of collisions grows with the number of particles at least as fast as for billiards, the algorithm we recommend is optimal.

Key Words: collision detection algorithm; hard sphere molecular dynamics; complexity; particle laden flow; fluid suspension; back-coupling.

INTRODUCTION

Consider a system of n Newtonian particles colliding with each other, but otherwise moving along independent trajectories. This can be cast as a solution to the system

$$m_i \ddot{x}_i(t) = F(t, x_i(t), \dot{x}_i(t)), \quad x_i(0) = q_i, \quad \dot{x}_i(0) = p_i, \quad i = 1, \ldots, n, \qquad (1)$$
$$+ \text{ Collisions},$$

where $x_i \in \mathbb{R}^d (d > 1)$ and m_i are the position and mass of particle i and $F(t, x, v)$ defines the external force exerted on a particle located at x with velocity v at time t. Collisions refer

0021-9991/01 $35.00

to discontinuous changes in the states of two particles[1] with labels i and j, at a time t_c such that $\|x_i(t_c) - x_j(t_c)\| = r_i + r_j$, where r_i and r_j are the radii of the particles. The system (1) is supplemented with boundary conditions.

A simple example of a collision is an *elastic collision*, in which the particles involved change the magnitude of their momenta along the line of contact in such a way that total momentum and energy are conserved. Typical boundary conditions are *hard walls*, where a particle bounces elastically off the walls of a container, or *periodic*, where a particle disappears at a boundary and reappears at the opposite side.

Solving (1) has wide application and has been studied by people in diverse fields, including molecular dynamics [1–4], granular flow [5, 6], and more recently in fluid suspensions [7–9]. It has also been studied by computer scientists, both in its own right in robotics and computational geometry [10, 11], and as a benchmark for parallel discrete event simulations [12–14]. Typical applications involve a large number of particles, so schemes for reducing the complexity of the simulation as a function of n are central. There is therefore considerable literature on the subject, although workers in different fields often appear not to be aware of each other's work.

The primary purpose of this paper is to identify an *efficient*[2] algorithm for collision detection among a large number of spherical particles immersed in a fluid with which they interact through exchange of momentum. This can be modeled as the system (1) where F is determined by the solution of a PDE which itself depends on $\{x_i(t)\}_{i=1}^n$ and $\{\dot{x}_i(t)\}_{i=1}^n$. The kind of applications we have in mind are for example an aerosol of solid particles or a spray of droplets in a carrier gas. We do not maintain that our approach is suitable for *all* applications involving particles immersed in a fluid. For instance, the Navier–Stokes equation for solid particles in a liquid gives rise to existence of squeeze and shear lubrication forces which demands a different numerical solution procedure [15, 16].

In order to build intuition, we consider a sequence of three classes of problems, of increasing complexity, the third of which is of the desired form. The problem classes arise by considering different forms for F, and each class is of interest in its own right. The three problem classes are:

(I) *Billiards*: The particles move in straight lines with constant velocities between collisions, so $F \equiv 0$; see Section 1.

(II) *Particle laden flow*: The particle motion between collisions is more complicated, but between collisions any two particles move independently of one another. Here F is some given function, a natural choice being that F is proportional to the difference between $\dot{x}$ and a background velocity field at x; see Section 2.

(III) *Coupled particle-flow*: Any motion of a particle affects the surrounding field, and hence the other particles. In this case, F is constructed from the solution of a PDE for the flow, which itself depends on the particle trajectories; see Section 3.

In all cases we consider elastic collisions, although other models are of interest. The nature of the algorithm will not be changed by other collision models, though particle distributions, and hence the analysis, might be.

[1] Collisions involving three or more particles can occur, but they are unstable in the sense that a small change to the particle configuration will replace them by two or more binary collisions. We deal with collisions of three or more particles as a sequence of binary collisions.

[2] Efficiency refers in general to both computational cost and memory requirements. We will mainly consider the computational cost; minimizing cost, in this case, also tends to minimize memory.

The algorithms we consider are *exact* in real arithmetic for billiards in that all collisions are detected and acted upon. For problems (II) and (III) the collision detection is exact up to small errors introduced through trajectory approximation.

As a byproduct of our studies we give a thorough literature survey for problem (I) and describe a small modification of the algorithm of Lubachevsky [14] and Marín *et al.* [4] for (I), which forms the basis of our studies of problems (II) and (III). Furthermore, we give a detailed analysis of the complexity of the resulting algorithms for (I) and give a theoretical derivation of the complexity and optimal choice of parameters, something which has been lacking in the literature. This analysis rests on Boltzmann-like assumptions on particle distributions and on an empirical observation about the behavior of the algorithm. Several authors, including Erpenbeck and Wood [2], Rapaport [3], and Lubachevsky [14], have identified the correct parameter choices empirically or on heuristic grounds, so that our analysis simply gives firm theoretical foundation to a well-known algorithm for problem (I). Our analysis of algorithms for (I) uses ideas and results from statistical mechanics, and was motivated by Kim *et al.* [10] who suggested, but did not carry through to its conclusion, this approach to the analysis. Our extension of the algorithm to problems (II) and (III) is new. Numerical solution of (III) involves solving a PDE which raises the additional issues of *numerical stability* for coupled particle-flow equations, and we investigate this important issue experimentally.

1. BILLIARDS

We begin by discussing the case when the particle motion in the absence of collisions is simple and known in advance, say $F \equiv 0$, so the particles move in straight lines with constant velocities between collisions, or $F \equiv -ge_z$, e_z a unit vertical vector, for particles moving in a uniform gravitational field. In Section 1.1 we give a historical review of the development of algorithms for such simulations, followed by a description of the details involved in the most efficient algorithm in Section 1.2. In Section 1.3 we analyze the complexity of the algorithm, and give the optimal parameter choice, supported by experimental results in Section 1.4. Our analysis of the billiards problem forms the basis for studying the more complex problems (II) and (III) in Sections 2 and 3, and proves to be useful even though the assumptions made cannot be justified for those problems.

1.1. Historical Review

To simulate the system (1) with nonzero F, the natural approach for many trained in numerical methods is to discretize time, and integrate the system over a time step Δt. Then, at the end of each time step, check whether any two particle are overlapping, and if so, assume they have collided and take appropriate measures to deal with the collision. This approach was indeed explored by Sundaram and Collins [7]. It has, however, numerous problems. For example, during a time step, a particle pair may collide, overlap, and then separate again, leaving no evidence of the collision at the end of the time step. To capture most collisions, a short time step is therefore needed, which increases the computational cost. Another problem is what to do in case of a collision; after dealing with all the overlapping particles, one would like to ensure that no two new particles are overlapping. But eliminating the overlap of a particle pair in some way might result in overlap between one of the two particles and another particle in the system. This appears therefore not to be the correct approach to the problem.

A different approach is suggested by considering first the case $F \equiv 0$, in which the particles move in straight lines between collisions. In that case, we can actually compute

the exact time of a collision between any two particles. Consider two spherical particles whose positions at time t are given by

$$x_1(t) = q_1 + v_1 t, \quad \text{and} \quad x_2(t) = q_2 + v_2 t,$$

where q_1 and $q_2 \in \mathbb{R}^d$ are their positions at time 0, and v_1 and $v_2 \in \mathbb{R}^d$ are their constant velocities. Furthermore, denote their radii by r_1 and r_2, respectively. They will collide at time t_c if and only if the distance between their centers equals the sum of their radii, i.e., if $\|x_1(t_c) - x_2(t_c)\| = r_1 + r_2$. Now square both sides and let $\Delta v = v_1 - v_2$, $\Delta x = q_1 - q_2$ and $\sigma = r_1 + r_2$ to get

$$\|\Delta v\|^2 t_c^2 + 2(\Delta v \cdot \Delta x) t_c + \|\Delta x\|^2 = \sigma^2. \tag{2}$$

Hence the collision time t_c is simply a root of a quadratic. If the particles are not overlapping at time 0, and this equation has two solutions, then the smaller solution is the time of their next collision. Otherwise, if the equation has no solution, the particles will not collide if they move along the same straight line with constant velocity indefinitely. Note that even if the particles are moving in a uniform gravitational field, the formula for the collision time is the same since their relative motion is linear.

Simple Algorithm

This observation suggests the following algorithm to simulate the system (1) in the billiards case $F \equiv 0$, or the uniform gravity case $F \equiv -ge_z$:

Step 1. Compute the time of the next collision in the system, t_m.
Step 2. Advance all particles in the system up to time t_m.
Step 3. Change the state of the two colliding particles.

Then repeat these three steps up to the required time. In simulation terminology, this algorithm is termed *event driven* since it advances the system from event, that is collision, to event. Alder and Wainwright [1] were the first to describe an event driven computer simulation of hard spheres moving along straight lines between collisions, and their starting point was this simple algorithm.

In the applications we have in mind the number of particles, n, is often large, so a key question regarding computational cost is how the computing time of an algorithm scales with n. We will therefore discuss the complexity of all proposed algorithms as n varies. In this section we will often give only a heuristic discussion of complexity, but give a detailed analysis for the optimal algorithm in Section 1.3. We use the standard notation in analysis of algorithms [17, chap. 1], with $f(n) = \mathcal{O}(g(n))$ meaning that there exists a constant $c > 0$ such that for all large n, $f(n) < cg(n)$, with $f(n) = \Omega(g(n))$ meaning that there exists a constant $c > 0$ such that $f(n) > cg(n)$ for all large n, and with $f(n) = \Theta(g(n))$ meaning that $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$.

From the outset we note that an event driven algorithm appears to need to perform at least as many operations as the total number of collisions, n_c, in the simulated time interval $[0, T]$, so an optimal algorithm has complexity $\Omega(n_c)$.[3]

[3] Indeed, if the ordered times at which collisions occur is a required output then, by embedding a sorting problem within collision detection, it is possible to argue that, in many models of computation, the algorithm has complexity $\Omega(n_c \log n)$ [18].

To analyze the complexity of the simple algorithm above, note that to find t_m in the first step, one could compute the collision times for every particle pair in the system, using Eq. (2) for each pair, and select the minimum. This requires one calculation of a collision time for each particle pair, for a total of $n(n-1)/2$ calculations. Each calculation involves a few additions ($3d-2$), subtractions ($2d+3$) and multiplications ($3d+3$), one division, and one square root, but as in customary in the analysis of algorithms, we ignore the actual number and only analyze how the number scales with n; accordingly we say that Step 1 takes $\Theta(n^2)$ calculations. For Step 2 we change the state of each of the n particles so there are $\Theta(n)$ calculations. Finally, in the third step we change the state of only two particles, requiring a constant number of calculations, independent of n, denoted $\Theta(1)$. Therefore, simulating n_c collisions with this algorithm takes on the order of $\Theta(n_c n^2)$ calculations; clearly very far from the desired optimum $\Omega(n_c)$, and likely to put severe limitations on the size of systems tractable for simulation.

This does not mean that this simple algorithm should not be used. For very small systems, say $n < 100$, it is likely to perform better than any of our later suggestions, and given its simplicity it might be the method of choice for even larger systems. For the applications we have in mind however, this algorithm is not an option.

Saving Collision Times–The Event Queue

Alder and Wainwright [1] studied this problem for molecular dynamics simulations. They noted that most of the collision times computed in Step 1 on two consecutive iterations will be the same. A single collision is not likely to affect collisions between distant particles in the near future. Saving the computed collision times would result in drastic savings in computing time. Only collision times for the two particles involved in the collision need to be recomputed and their old times discarded. This way, only $2n-3$ particle pairs, $n-1$ for one particle and $n-2$ for the other, need to be examined in Step 1, except when computing the very first collision time, giving total cost of $\Theta(n_c n)$, or so it seems.

This method, however, raises the important issue of how to maintain the list of the saved collision times, called the *event queue*. After each collision, we need to determine which collision will occur next, in other words which particle pair has the smallest collision time. A simple way to carry this out is to store the computed collisions, i.e., which particle pair is involved and the time of collision, without any particular order, and search through the list every time an event occurs. If all $n(n-1)/2$ collision pairs are kept, this requires $\Theta(n^2)$ calculations, which brings the complexity back up to $\Theta(n_c n^2)$. This issue was not addressed by Alder and Wainwright [1], but we return to it later as addressing it will clearly be a central ingredient in efficient algorithms.

The Cell Method

At this point we note that it seems wasteful to compute, in Step 1, future collision times for every single particle pair in the system; each particle will only participate in one collision before it changes its course and thereby renders all its previously computed collision times invalid. Since a particle is more likely to collide with another that is in its close vicinity than one that is far away, it is natural to consider only collisions between close particles.

Alder and Wainwright [1] suggested dividing up a cube containing all the particles into a grid of small cubes, called *cells* from now on, and assigning each particle to the unique cell

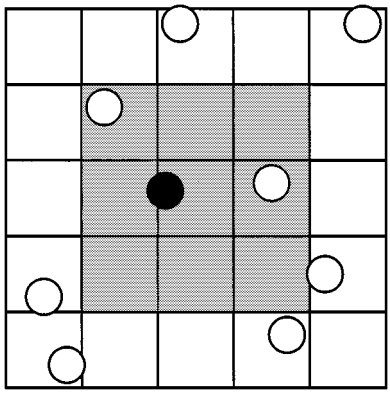

FIG. 1. The black particle only computes collision times with particles in the $3^2 = 9$ shaded cells.

containing its center. Then collisions are only considered between particles in neighboring cells of the grid (see Fig. 1), at the expense of keeping track of which cell a particle is in. That is, in addition to collisions, transfers between cells must be detected for each particle.

This changes Step 1 in the algorithm to

Step 1′. Compute the time of the next event, meaning a collision or a transfer, in the system, t_m,

and Step 3 to

Step 3′. Handle the event; that is change the state of the two particles in the event of a collision, or update the cell structure in the event of a transfer.

This detection of transfers ensures that no collisions are overlooked; two colliding particles must be in neighboring cells at the moment they collide, and once a particle changes cells the algorithm examines all particles in the neighboring cells for possible collisions.

If the number of cells is m^d (where d is the dimension of the space), the number of pairs examined in Step 1 is reduced to $2 \cdot 3^d n/m^d$ on the average, assuming the particles are uniformly distributed. The finer the grid, the fewer pairs need to be examined per collision; however, refining the grid increases the number of transfers to be detected and handled. This suggests that there is an optimal choice of the cell size, and in Section 1.3 we find, under mild statistical assumptions, how that optimal cell size scales with n. Since only particles in neighbouring cells are considered for collisions, the side length of a cell,

$$L = \frac{D}{m}, \tag{3}$$

where D is the side length of a cube containing all the particles, can be no smaller than the diameter of the largest particle in the system; see Fig. 2.

Alder and Wainwright [1] did not implement this scheme, as it requires quite a lot of computer memory which was a scarce resource at the time. Furthermore, for the size of systems they were simulating, less than 500 particles, it is not likely to have had a major impact on performance. However, for the size of problems accessible on today's computers it is a central to efficient algorithms.

We have now identified the main ingredients of the final algorithm, and what follows is mostly fine tuning. The three primary data structures to be maintained are

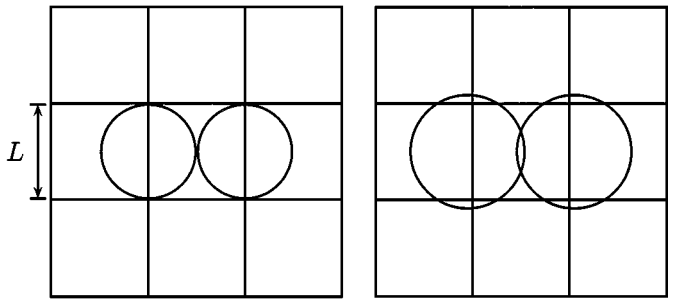

FIG. 2. A cell can be no smaller than the diameter of a particle, $L > 2r$. LEFT: $L = 2r$; the two particles do not belong to adjacent cells, and are not touching. Right: $L < 2r$; the two particles do not belong to adjacent cells, but are overlapping.

The Particle Information which consists of the position, x, and velocity, v, of each particle, along with any other information needed, such as its radius, r, in case of particles of different sizes.

The Event Queue which is a collection of events, each of which has an event time and the information necessary to handle (or carry out) the event, such as the two particles involved in a collision, or the cell a particle will transfer to.

The Cell Structure which is a collection of cells, each of which has a list of the particles belonging to it.

The algorithms we discuss differ in how the event queue is implemented and how many events are put in it, and to a lesser extent how the cells are stored and utilized. Below we identify a good implementation of the queue that allows the operations needed to be carried out in as few operations as possible.

Delaying the Update

So far we have introduced two schemes to reduce the computations done in Step 1. If the cell size can be chosen so that only a constant number of particles are examined for collisions, and if the event queue can be implemented efficiently, it seems that the cost of Step 1 can be made largely independent of n, and we make this precise later. On the other hand, the innocent looking Step 2 still costs $\Theta(n)$ calculations, and has thus become the bottleneck. Erpenbeck and Wood [2] noted that this step only needs to be carried out for the particles involved in the event, reducing the cost of Step 2 to constant per event. Less importantly, since transfers do not change the path of a particle, there is no need to carry this step out in case of a transfer. This means that the position, x, and velocity, v, stored for a particle now stands for its position and velocity at the time of its last collision (as opposed to the time of the last event in the system). For each particle we therefore need to keep additionally the time of its last collision, t_c. Since the particle motion between collisions is linear, we can obtain the position of a particle at any time t as simply $x + (t - t_c)v$.

Implementing the Event Queue

We now turn to the one remaining issue of maintaining the event queue set up in Step 1′. Neither Alder and Wainwright [1] nor Erpenbeck and Wood [2] mentioned how to do this. At each event, we need to determine which event occurs next, so the data structure for

the events should allow extracting the next event, i.e., the one with the smallest time, and inserting and removing events as the simulation proceeds. That is, we need an efficient implementation of a priority queue. Rapaport [3] suggested using a binary search tree [17, chap. 13] to implement the queue, allowing the aforementioned operations to complete in $\Theta(\log s)$ steps on average for a queue of size s, assuming that the tree is randomly built; such randomness has been observed empirically for MD hard sphere simulations [3]. Alternatively, one can *ensure* that the operations have complexity $\mathcal{O}(\log_2 s)$ by using a balanced tree, such as a red black tree [17, chap. 14], as suggested by Kim *et al.* [10, 11].

Using an efficient implementation of the event queue along with the cell structure therefore makes the cost of Step 1 $\mathcal{O}(\log n)$ for each event. We saw that, by delaying the update, Step 2 involves a constant number of operations per event, and Step 3 only required a fixed number of operations to begin with. We have thus managed to bring the total cost of the algorithm down to $\mathcal{O}(n_e, \log n)$, where n_e is the total number of events over the course of the simulation. Now $n_e \geq n_c$, as n_e includes transfers, so we cannot yet conclude that this algorithm will have complexity $\mathcal{O}(n_c \log n)$, unless the cell size can be chosen to make n_e grow no faster than n_c asymptotically. In Section 1.3 we will make the complexity analysis more systematic and rigorous, and see how to achieve this.

One Event per Particle

Our analysis has conformed to the standard practice of ignoring constants. In practice, the constants do effect the running time of the algorithm, so we finally describe one modification that does not affect the asymptotic complexity, but greatly reduces the constant.

In the algorithm described so far, several collisions and one transfer are scheduled per particle. Lubachevsky [13] noted that all but one or two of these will eventually be removed from the event queue since once a particle is involved in a collision, all subsequent computed events for that particle become invalid. It therefore seems appropriate to only keep one event per particle, and this is what Lubachevsky [13] does.

It is true that a particle will not necessarily engage in the first collision foreseeable at the current time, since its proposed partner might earlier engage in a collision with a third party. Some savings in computing time might therefore result from storing more than one event per particle. However, scheduling only one event per particle results in a smaller event queue, and allows simpler data structures to be used efficiently for the event queue, such as a heap [17, chap. 7] or a complete binary tree [17, chap. 5]. Heaps, which are binary trees with the property that every node has a smaller value than its children, are known to be excellent implementations of priority queues, and so it is our choice of data structure for the event queue. In addition to being very efficient for priority queues, heaps are also incredibly simple, and can be implemented efficiently in less than 30 lines of code.

Another scheme, suggested by Marín *et al.* [4], is not to discard all but the next foreseeable event for a given particle, but store them all at the nodes of an event queue, with the queue ordered by the next foreseeable event for the particle. This also fixes the size of the event queue, but each node in the event queue now consists of a list of events. Through experiments, Marín *et al.* [4] find that this yields a significant improvement in efficiency.

We adopt a slightly different scheme, keeping only the next transfer and the next collision for every particle, which gives improvements in efficiency similar to those in [4]. Since a transfer doesn't change the path of a particle, a previously computed collision still remains

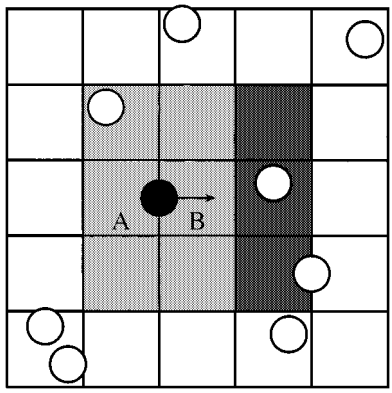

FIG. 3. The black particle just transferred from cell A to B and only computes collisions with particles in the new, dark-shaded neighboring cells, since it has previously computed collisions with the particles in the light-shaded cells.

valid after a transfer. Keeping the next foreseeable collision along with the transfer reduces the number of collision checks in the event of a transfer by a factor of $2/3$, since the particle involved does not need to recompute collision times with particles in all the neighboring cells, but only the new neighboring cells, as illustrated in Fig. 3. Once a particle is involved in a collision on the other hand, all subsequent events become invalid, so keeping more than one collision is not likely to improve the efficiency.

1.2. The Algorithm

The algorithm developed above is based on the simple algorithm presented at the outset, with several ways of reducing the computations done at each step. We now describe the details of the ideas used to reduce the cost. All but one of these schemes were presented in [13]; our main contribution is to the analysis of the algorithm and its extensions to the particle-field problems in subsequent sections.

The algorithm maintains three data structures. Much of the last section was devoted to identifying what information should be kept in each and how it should be implemented. To summarize:

The Particle Information is an array with one element for each particle in the system, with each element consisting of the position, velocity, and the time of the last collision of the corresponding particle.

The Event Queue is a heap containing one node for each particle in the system. Each node stores information on both (1) the next foreseeable collision of the corresponding particle, that is the collision time and some identification of the other particle involved, and (2) the next foreseeable transfer, that is the transfer time and some identification of the two cells. The nodes are ordered (or *keyed*) by the smaller of the two event times. There is a one-to-one correspondence between the events in the queue and the particles in the system. Each collision is therefore represented twice in the event queue, once for each particle involved in the collision, and which event is handled first is arbitrary.

The Cell Structure is an array with one element for each cell, each element containing a list of particle indices which enumerate the particles belonging to that cell. These lists can be implemented as linked lists or arrays.

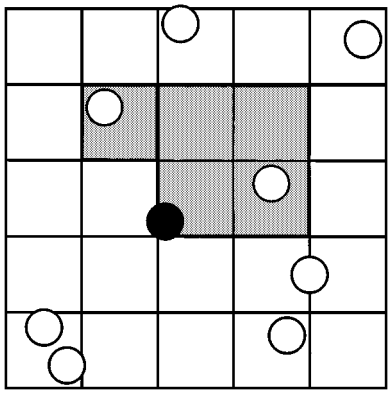

FIG. 4. Initially, the black particle only needs to check for collisions with particles in the shaded cells.

To start the simulation we have to initialize these three data structures. The particle information is initialized with the initial positions and velocities, and the last collision time set to zero. To initialize the cell structure, we compute the cell location of each particle and insert it into the appropriate list in the cell array. To set up the event queue, we need to check every particle pair in adjacent cells for collisions and compute a transfer time for every particle. Checking particles in all surrounding cells of a given particle for a collision would result in double checking every pair, so we only have to check particles in half of the surrounding cells (see Fig. 4), and only a part of the particles in the same cell. When a transfer time and a collision time has been computed for each particle, we create a heap from the n events.

Then we perform the following steps until we reach the desired final time:

S1. Find the next event in the event queue.
S2. Handle the event.
S3. Compute the next transfer time for the particle corresponding to the event.
S4. Compute the next collision time with particles in appropriate neighboring cells.
S5. Adjust the position of the event and its new partner's event in the event queue.
S6. Return to Step S1.

The smallest element of a heap is always at the top, so Step S1 consists simply of looking at the top element of the heap.

If the event is a transfer, Step S2 consists of moving the particle between cells, that is removing the particle from the list of one cell, and adding it to the list of another. For a collision, it consists of changing the states of the two particles involved in the collision, for example as described below for an elastic collision. Furthermore, to avoid changing the states of the particles again when the partner gets to handle the event, we change the collision event of the partner to a special event, which we call a *check*. This event, when handled, has no effect on the particle *state* but, as for collisions, forces the particle to recompute its next collision time with particles in all neighboring cells. Thus, handling a check event consists of nothing at all, but it will trigger the execution of Steps S3–S5. We will also find a further use for this event below. We now have three types of events: collisions, transfers, and checks.

In an elastic collision the particles involved change the magnitude of their momenta along the line of contact in such a way that momentum and energy are conserved. If we let

$p_i^- = m_i \dot{x}_i(t_c-)$ and $p_i^+ = m_i \dot{x}_i(t_c+)$ be the momentum of particle i immediately before and after a collision at time t_c, then an elastic collision between particles 1 and 2 is such that

$$p_1^+ = p_1^- + ad, \quad \text{and} \quad p_2^+ = p_2^- - ad,$$

where

$$a = \frac{2(m_1(p_2^- \cdot d) - m_2(p_1^- \cdot d))}{m_1 + m_2}$$

is the net exchange of momentum between the particles, and

$$d = \frac{x_1(t_c) - x_2(t_c)}{\|x_1(t_c) - x_2(t_c)\|}$$

is a unit vector in the direction of contact.

Computing a transfer event in Step S3 consists of finding the intersection of a line with d hyper-planes, which amounts to solving d linear equations, and selecting the smallest.

In Step S4, computing collision times involves solving the quadratic equation (2). The word *appropriate* refers to the fact that which cells to consider depends on the type of the event; see Fig. 1 for a collision and a check, and Fig. 3 for a transfer. For each computed collision time, the algorithm compares it to (1) the smallest time computed for the particle involved so far, and (2) the collision time of the partner particle, and keeps it only if it is smaller than both. When all collision times have been computed, the particle involved notifies its newly found partner, if any, to adjust its event time. A subtle point is that a third party, the partner's old partner, now has a collision time that is invalid. The easiest way to deal with this complication is to change the third party's collision event to the special check event described above, thereby cancelling the collision but still forcing the particle to recheck for collisions at the time of the event. This operation does not affect the third party's location in the priority queue since its event time remains the same.

After Steps S3 and S4 the particle involved in the event has updated its event time, so its position at the top of the event queue is incorrect, and has to be corrected in Step S5. Furthermore, if the particle involved in the event scheduled a new collision, it has notified its new partner who has in response changed its collision time, and so its position in the event queue is also invalid and needs to be repositioned. Both of these operations on heaps are described in [17, chap. 7].

So far we have ignored the boundary conditions, and proper modifications have to be made to handle them. For elastic walls we add one more event, a *wall collision*, which we check for in Step S4. Since a wall collision changes the path of a particle, we only keep either a particle collision or a wall collision for each particle. For periodic boundary conditions we modify the collision check routine to check for a collision with the nearest periodic image of each particle, and let the cells at opposite edges be adjacent;[4] see Fig. 5. In addition, each time we update the position of a particle we check whether the particle has left the domain, and if so add the domain length to, or subtract it from, the appropriate coordinates.

[4] For this to work, the number of cells, m^d, must be at least 3^d.

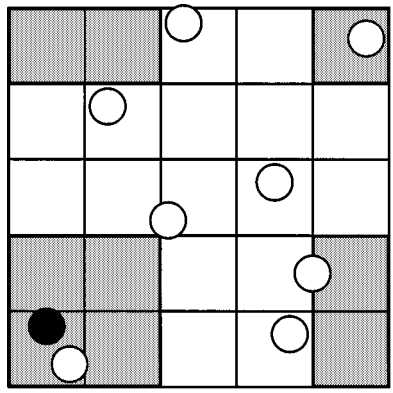

FIG. 5. For periodic boundary conditions, the black particle needs to check for collisions with the nearest periodic images of the particles in the shaded cells.

1.3. Complexity

Suppose we want to simulate a system of n particles over a time period $[0, T]$. How does the computing time of the algorithm increase as n increases? If n is large, this is clearly a key question regarding computational cost. As we noted earlier, an optimal algorithm will have cost scaling like the number of collisions. In this section we analyze in detail the complexity of the algorithm developed above, and derive the optimal choice of cell size. Our analysis is motivated by Kim *et al.* [10], who suggested using results from statistical mechanics to estimate the complexity, although they did not carry this program to its conclusion.

Obviously the behavior, and thereby the cost, of a collision detection algorithm will depend on the configuration of the particles in space and time. For the billiards case, statistical mechanics provides a set of assumptions about the *statistics* of the particle positions and velocities over space and time which, while remaining unproven, are strongly supported on empirical and theoretical grounds. We therefore start with a brief discussion of the relevant results from statistical mechanics which underpin our analysis.

The Maxwell–Boltzmann Distribution

Take $\Delta x > 0$ and $\Delta v > 0$ small and define the number density of particles per unit volume f such that $nf(x, v, t)(\Delta x)^d(\Delta v)^d$ is the total number of particles in the cube[5] $[x, x + \Delta x]$, and whose velocities lie in the cube $[v, v + \Delta v]$ at time t. As n gets larger, f becomes smoother, and we can think of approximating it with a continuous density. Along these lines, Boltzmann [19] treated a large collection of particles as a continuum,[6] and showed that for any initial distribution $f(x, v, 0)$, f approaches in the course of time the Maxwell–Boltzmann distribution

$$f(x, v) = C \exp\left(-\frac{\|v\|^2}{2\beta^2}\right),$$

[5] For $x \in \mathbb{R}^d$ and $\Delta x > 0$, $[x, x + \Delta x]$ denotes the cube with lower left corner at x, and side lengths Δx.

[6] Boltzmann's analysis applies to a wide class of interaction potentials for the particles, including the hard sphere potential.

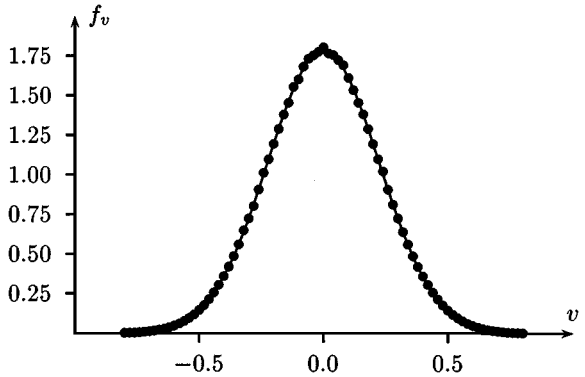

FIG. 6. Equilibrium velocity distribution of a single particle in a hard sphere simulation (dots) and the Maxwell distribution (solid).

where C is a normalization constant, and β is determined by the total energy of the particles [20]. This means that no matter what the initial configuration of the particles is, if we look at the spatial and velocity distributions of the particles at a single instance in time, after some transition period, we will find that

A1. The particle positions are independent and uniformly distributed over accessible positions;

A2. The particle velocity components are independent and Gaussian with mean zero and variance β^2;

A3. The spatial and velocity distributions are independent of one another.

For the second conclusion, we used that

$$\exp\left(-\frac{\|v\|^2}{2\beta^2}\right) = \prod_{i=1}^{d} \exp\left(-\frac{v_i^2}{2\beta^2}\right).$$

These results are obtained by treating the collection of particles as a continuum, and are not true for any finite n. Nonetheless, for all practical purposes, the Maxwell–Boltzmann distribution is an excellent approximation for the particle distribution after a short time if n is large. Figure 6 shows the velocity distribution of a single particle in time over several million collisions, generated by running the algorithm described in Section 1.2 with $n = 1000$. Even for this small number of particles, the agreement with the Gaussian prediction is excellent.[7]

For our analysis, we will therefore assume that n is large and the particles have the Maxwell–Boltzmann distribution at all times; that is, we take A1–A3 above as *assumptions*. This includes the assumption that the initial configuration satisfies A1–A3, but since almost any initial configuration will rapidly evolve to the Maxwell–Boltzmann distribution, this assumption is not too restrictive. Our analysis is accordingly average case analysis, averaging over initial conditions taken from the Maxwell–Boltzmann distribution. We expect however, and observe experimentally, that because of ergodicity, single realizations will give rise to similar complexity. Unfortunately these three simple assumptions do not suffice for bounds

[7] To make the connection between the velocity distribution of a single particle over time and the velocity distribution of the collection of particles at a particular instance in time, we are assuming ergodicity and independence of different particles.

on the expected complexity of the algorithm, and we will add a fourth assumption, A4, below; we postpone its statement as it involves, contrary to assumptions A1–A3, some details of the algorithm.

Operation Count

We start by counting the number of operations in of each the Steps S1–S5, ignoring constants as before. We have informally been through most of this in Section 1.1, but here we make the treatment more precise.

S1. A single operation.

S2. Constant number of operations in the event of a collision or a transfer, no operation in the event of a check.

S3. Constant number of operations.

S4. Constant number of operations for each particle in the neighboring cells, for a total of $n_s(i) - 1$ operations, where $n_s(i)$ is the total number of particles, at the occurrence of this event (labelled i), in the 3^d cells surrounding and including the cell containing this event.

S5. At most $\log n$ operations.

The total number of operations over the course of the simulation is therefore

$$\mathcal{O}\left(\sum_{i\in\text{events}} (1 + n_s(i) + \log n)\right). \tag{4}$$

The first term comes from Steps S1–S3 (getting the event, handling it, and computing a transfer), the second from Step S4 (computing collisions), and the third from Step S5 (adjusting the positions in the event queue).

Average Number of Operations

The expression (4) depends on the number of events, and how the particles are distributed throughout the domain as the events occur. We now compute its average under the statistical assumptions A1–A3. For a function X of the particle positions and velocities we denote by $\mathbb{E}X$ the average, or expected value, of X over an ensemble of simulations obeying A1–A3.

Under assumption A1 the expected number of particles in a cell at any fixed instant in time is $n_0 L^d$, where $n_0 = n/D^d$ is the particle number density, and $L = D/m$ is the side length of a cell. Now $n_s(i)$ is the number of particles in the 3^d cells surrounding the event i *at the occurrence of event i*, which is *not* a fixed instant in time. For instance, if the event is a collision, we know that $n_s(i)$ is at least two, namely the two colliding particles. In the event of a transfer, we know that $n_s(i)$ is at least one, namely the particle being transferred. We are tempted to conclude that the expected value of $n_s(i)$ is increased by no more than two particles,

$$\mathbb{E}n_s(i) \le 3^d n_0 L^d + 2.$$

However, in a region of high particle number density, collisions are more frequent than in a region of low particle number density. Reversing the argument, we could argue that the occurrence of a collision in a region is, on average, an indicator of higher particle number density; i.e., the expected number of particles in a cell at a collision, $\mathbb{E}n_s(i)$, is higher than

the expected number of particles in a cell at a fixed instant in time, $n_0 L^d$, not only by the 2 particles involved in the collision but possibly by a factor.

Below we will see that we take the limit $n \to +\infty$ in such a way that the total volume fraction occupied by the particles is fixed, $nr^d/D^d = C$, so $r/D = Cn^{-1/d}$ (with a different constant C). In a cell of side length L we can fit at most $C(L/r)^d = Cn_0 L^d$ particles (with a yet different constant C), which is therefore a firm upper bound on the number of particles in a cell. In particular,

$$\mathbb{E}n_s(i) \leq Cn_0 L^d$$

for some constant $C \geq 3^d$ independent of n.

Using the law of iterated expectation, we get

$$\mathbb{E}\left[\sum n_s(i)\right] = \mathbb{E}\left[\mathbb{E}\left[\sum n_s(i) \mid n_e\right]\right] = \mathbb{E}\left[\sum \mathbb{E}[n_s(i) \mid n_e]\right],$$

since given n_e the number of terms in the summation is fixed, and equal to n_e. Now $n_s(i)$ is independent of how many events there are in total,

$$\mathbb{E}[n_s(i) \mid n_e] = \mathbb{E}n_s(i) \leq Cn_0 L^d,$$

so

$$\mathbb{E}\left[\sum n_s(i)\right] \leq \mathbb{E}\left[\sum Cn_0 L^d\right] = \mathbb{E}[n_e Cn_0 L^d] = Cn_0 L^d \mathbb{E}n_e.$$

The other two terms in the sum in (4) are independent of i, so the expected total number of operations is

$$\mathcal{O}((1 + n_0 L^d + \log n)\mathbb{E}n_e). \tag{5}$$

To continue we need to determine how $\mathbb{E}n_e$ depends on n and L. Now $n_e = n_c + n_t + n_{ch}$ where n_c is the number of collisions, n_t is the number of transfers, and n_{ch} is the number of checks, so we proceed to determine the average value of each term.

Number of Collisions

Under assumptions A1–A3, arguments from statistical mechanics [21, pp. 461–471] give that for a dilute system of particles the average number of collisions, n_c, in a time period $[0, T]$ is

$$\mathbb{E}n_c = \mathbb{E}\|v_i - v_j\|\sigma_c n_0 nT, \tag{6}$$

where $n_0 = n/D^d$ is the particle number density, $\|v\|$ is the Euclidean norm of v, $\|v\|^2 = \sum_{i=1}^d |v|^2$ so $\mathbb{E}\|v_i - v_j\|$ is the mean relative speed of two particles, and σ_c is a collision cross section of two particles; we have $\sigma_c = a_d r^{d-1}$, in particular $\sigma_c = 2r$ for $d = 2$ and $\sigma_c = 4\pi r^2$ for $d = 3$.

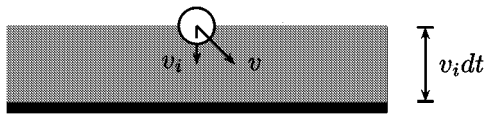

FIG. 7. Expected number of crosses over a plane.

Number of Transfers

To express n_t in terms of n and L, first consider how many particles on average cross a plane perpendicular to one of the coordinate axes in a time interval of length dt. A particle with velocity v_i perpendicular to the plane[8] will pass it if it is closer to it than $v_i dt$, and is traveling in the right direction; see Fig. 7. Under assumptions A1–A3, the expected number of particles passing the plane in time dt is thus $|v_i|\,dt f(v_i)\,dv_i n_0 D^{d-1}$, where $f(v_i)\,dv_i$ is the density of particles with velocity v_i along the axis perpendicular to the plane, and $n_0 D^{d-1}|v_i|\,dt$ is the number of particles in a slab of thickness $|v_i|\,dt$.

Integrating over v_i and t then gives the total number of passes as $\mathbb{E}|v_i|Tn/D$. The cells can be thought of as composed of $m = D/L$ planes in each dimension so multiplying by m and summing over i gives the expected total number of transfers as

$$\mathbb{E}n_t = \mathbb{E}\|v\|_1 nT/L, \tag{7}$$

where $\|v\|_1$ denotes the 1-norm of v, $\|v\|_1 = \sum_{i=1}^{d} |v_i|$.

Number of Checks

To count the number of checks, n_{ch}, recall that we use them for two purposes. For the first purpose, a check is always introduced at a collision. For the second, a check will be introduced in the event of a transfer or a collision if and only if the new partner had a scheduled collision. One might therefore be tempted to conclude that there is at most one check introduced in the event of a transfer and at most two at a collision, so $n_{ch} \le 2n_c + n_t$. However, the handling of a check might itself introduce another check, so no immediate bound in terms of the other two events is obvious. In fact, this issue is raised in [22] and [23].

In practice n_{ch} is usually far less than $2n_c + n_t$, typically $n_{ch} \approx 1.1n_c$, so we make in addition to assumptions A1–A3 the following reasonable assumption:

A4. The expected number of checks is bounded by a constant C, independent of n, times the expected number of transfers and collisions, $\mathbb{E}n_{ch} \le C(\mathbb{E}n_c + \mathbb{E}n_t)$.

Since we are ignoring constants we can therefore combine $\mathbb{E}n_{ch}$ with $\mathbb{E}n_c + \mathbb{E}n_t$, that is drop it altogether.

Complexity

Combining the expressions (7) for n_t and (6) for n_c with (5) we get the average complexity of the algorithm as

$$\mathcal{O}((1 + \log n + n_0 L^d)(\mathbb{E}n_c + \mathbb{E}n_t)) = \mathcal{O}\left((1 + \log n + n_0 L^d)\left(\sigma_c n_0 + \frac{1}{L}\right)\beta T n\right), \tag{8}$$

[8] In this paragraph the subscript i refers to a component of the velocity vector; everywhere else, v_i labels the velocity of particle i.

where we have replaced $\mathbb{E}\|v_i - v_j\|$ and $\mathbb{E}\|v\|_1$ by β for simplicity, for if the particles have the Maxwell–Boltzmann velocity distribution, then

$$\mathbb{E}\|v_i - v_j\| = \int\int \|v_1 - v_2\| f(v_1) f(v_2)\, dv_1\, dv_2 = \sqrt{2}\mathbb{E}\|v\| = \sqrt{2\,d}\beta, \quad \text{and}$$

$$\mathbb{E}\|v\|_1 = \int \|v\|_1 f(v)\, dv = \frac{1}{\sqrt{2\pi}}\mathbb{E}\|v\| = \sqrt{\frac{d}{2\pi}}\beta.$$

Choice of Units

For hard sphere molecular dynamics it is customary to choose the units of mass, length, and time such that the unit mass is the mass of a single particle, the unit length is the diameter of a particle, $\sigma = 2r$, and the unit energy is $m\beta^2$. Then the unit time is σ/β. There are only two free parameters in this system, and with the units chosen in this way, it is convenient to choose the particle number density n_0, and the number of particles, n. In these units, we can therefore write (8) as

$$\mathcal{O}\left((1 + \log n + n_0 L^d)\left(n_0 + \frac{1}{L}\right)Tn\right).$$

These units are natural for hard sphere molecular dynamics, but we will study the more general case of particles moving in a velocity field. For that problem, the more natural length unit is the length scale of the velocity field, typically the size of the domain, D. The natural time scale is such that the unit velocity is a typical field velocity. Usually the velocity of a single particle will be close to the field velocity, and so β is a natural unit velocity. With this choice of length scale $n_0 = n$, so n and n_0 are not different parameters. For our parameters we take n and the volume fraction of the particles, $\rho = n\sigma_v$, where $\sigma_v = b_d r^d$ is the volume of a single particle; thus $\sigma_v = \pi r^2$ in 2D, $\sigma_v = \frac{4\pi}{3}r^3$ in 3D. Then $r = (\rho/nb_d)^{1/d}$ and therefore $\sigma_c = a_d(\rho/nb_d)^{1-1/d} = C\rho^{1-1/d}n^{1/d-1}$ so we can rewrite (8) as

$$\mathcal{O}\left((1 + \log n + nL^d)\left(\rho^{1-1/d}n^{1/d} + \frac{1}{L}\right)Tn\right). \tag{9}$$

We take the limit $n \to \infty$ in such a way that β, D, and ρ are fixed.

In what follows we shall work in these units, and so we note from (6) that the total number of collisions is

$$n_c = \rho^{1-1/d}n^{1+1/d}T. \tag{10}$$

Each collision involves two particles, so each particle has $2n_c/n$ collisions on average during the time interval $[0, T]$. The average time between successive collisions of a single particle, the *mean collision time*, is therefore

$$\tau_c = \frac{Tn}{2n_c} = \frac{1}{2}\rho^{1/d-1}n^{-1/d}. \tag{11}$$

Optimal Cell Size

The complexity (9) is, as expected, dependent on the choice of the cell size L. As the cell size is decreased the second factor increases, which reflects the fact that more transfers have

to be detected, but the first factor decreases, which reflects the fact that fewer particle pairs need to be examined for collisions at each event. A natural question is therefore whether the cell size can be chosen to make the complexity close to n_c.

Balancing the two terms 1 and nL^d in the first factor gives $1/L^d \sim n$, and balancing the two terms in the second factor gives the same scaling. With this choice of L, the first factor in (9) is $\Theta(\log n)$, the second factor is $\Theta(n^{1/d})$, and the product of the three factors is $\Theta(n^{1+1/d} \log n) = \Theta(n_c \log n)$. To summarize, we have the following:

Conclusion

Under the assumptions A1–A4, and for a fixed volume density and kinetic energy, the average case complexity of the algorithm is $\mathcal{O}(n_c \log n)$ if the total number of cells is proportional to the number of particles, that is $m^d = \Theta(n)$.

Thus, with this choice of cell size scaling with n the algorithm is optimal, since the cost would appear to be $\Omega(n_c \log n)$ in reasonable models of computation.

1.4. Experiments

The analysis in the preceding section is based on statistical assumptions which are not proven for any finite n although, as mentioned, they are widely accepted in the statistical physics literature. In this section we validate the analysis through a variety of experiments with the algorithm. All the experiments are performed at fixed volume density and kinetic energy as n increases.

2D Billiards

We run the algorithm in two dimensions with n varying from 5,000 to 100,000 in increments of 5,000, while keeping the volume density, ρ, fixed at 15%.

We try different values of $m = 1/L$ for each n in order to explore how the running time varies with m. In Fig. 8 (left) we plot the running time versus m for $n = 5000$. There is clearly a minimum around $m = 90$. We do this for each value of n and find the value of m that gives the least running time, m_{opt}, and plot in Fig. 8 (right) the result. Regression on $\log m = \log a + b \log n$ gives $a = 0.7476$ and $b = 0.5620$, quite close to the theoretical prediction $m \sim \sqrt{n}$.

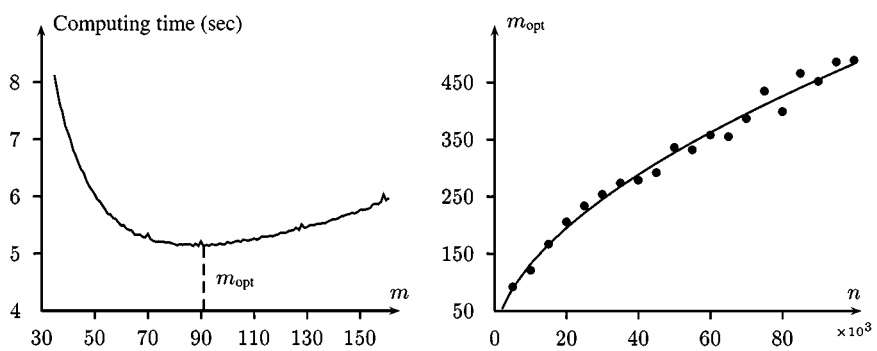

FIG. 8. 2D billiards, $\rho = 15\%$. Left: Computing time in seconds vs. m for $n = 5000$. Right: m_{opt} vs. n in thousands (dots) and the fitted curve an^b, with $a = 0.7476$ and $b = 0.5620$ (solid); this compares well with the predicted value $b = \frac{1}{2}$.

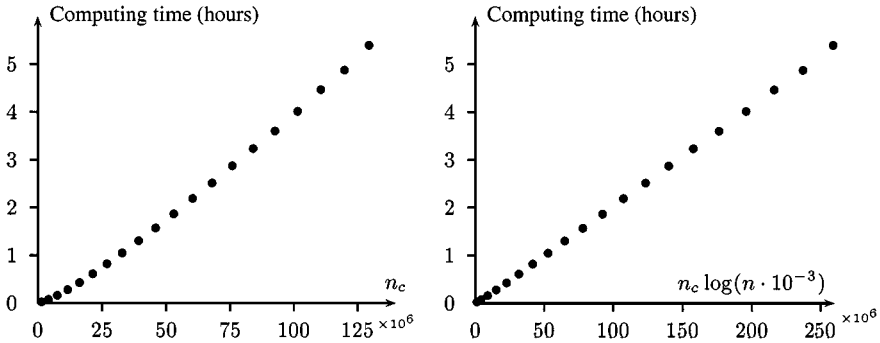

FIG. 9. 2D billiards, $\rho = 15\%$. Left: Computing time in hours vs. n_c in millions. Right: Computing time in hours vs. $n_c \log n$ with n_c in millions and n in thousands. The computing time appears linear in $n_c \log n$, and the best fit of the form $a(n_c \log n)^b$ has $a = 0.02$ and $b = 1.0017$.

In Fig. 9 we show how the computing time varies with n_c (left) when the optimal cell size is used. It appears slightly super linear, and Fig. 9 (right) shows the computing time versus $n_c \log n$ and it appears to be perfectly linear. Indeed, fitting a curve of the form $a(n_c \log n)^b$ gives $b = 1.0017$.

3D Billiards

We repeat the preceding experiment in three dimensions, with $\rho = 15\%$ as before. Fig. 10 left shows how the running time varies with $m = 1/L$ for $n = 50000$. It appears monotone in m. The minimum is at $m = 55$, in which case the cell size L equals the particle diameter. The algorithm wants to use smaller cells, but the restriction $2r < L$ (see Fig. 2) forbids that. This is a result of the high particle density. The modification suggested by Kim *et al.* [10, 11], that is to check only particles in the same cell for collisions and allowing a particle to belong to multiple cells, could slightly improve the efficiency of the algorithm in this case. The power law fit of m to an^b gives $b = 0.3382$, which is very close to the prediction $\frac{1}{3}$, but in this case it is simply due to the fact that $m \sim \frac{1}{r} \sim \sqrt[3]{n}$. From Fig. 11 we see that the cost is still near linear in $n_c \log n$, as is confirmed by regression; fitting the cost to $a\,(n_c \log n)^b$ gives $a = 0.6488$ and $b = 1.0488$.

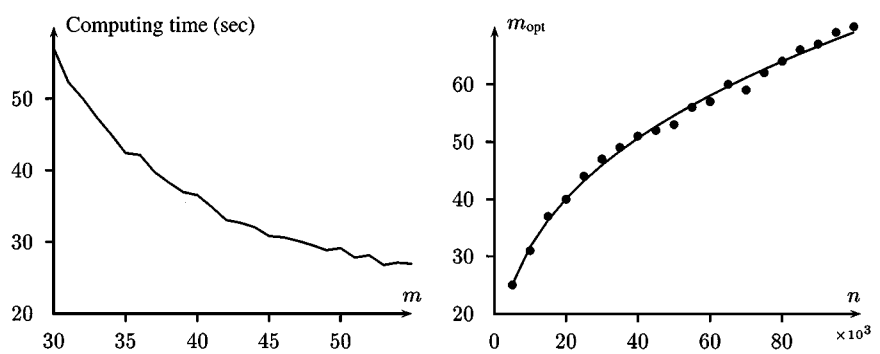

FIG. 10. 3D billiards, $\rho = 15\%$. Left: Computing time in seconds vs. m for $n = 50,000$. Right: m_{opt} vs. n in thousands (dots) and the fitted curve an^b, with $a = 0.3406$ and $b = 0.3382$ (solid); this compares favorably with the prediction $b = \frac{1}{3}$.

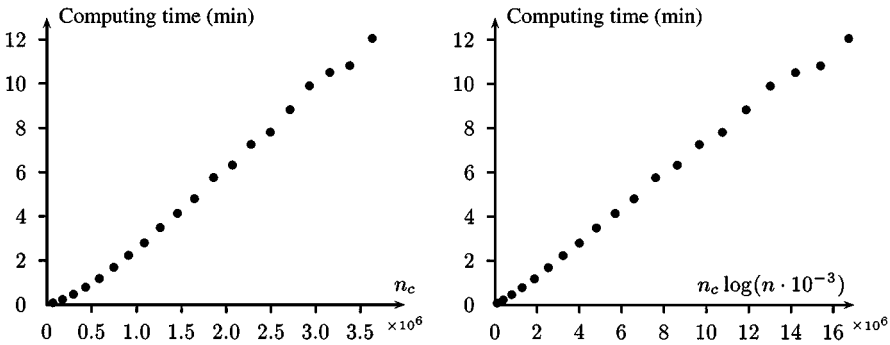

FIG. 11. 3D billiards, $\rho = 15\%$. Left: Computing time in minutes vs. n_c in millions. Right: Computing time in minutes vs. $n_c \log n$. The best fit of the form $a(n_c \log n)^b$ has $a = 0.6488$ and $b = 1.0488$.

To further test our conclusions from Section 1.3, we redo the experiment in 3D with lower density, $\rho = 1\%$. Figure 12 (left) shows how the running time varies with the cell size for this lower density. There is a clear minimum around $m_{\text{opt}} \approx 50$, and repeating this for different number of particles and recording for each n the optimal m results in the plot in Fig. 12 (right).

Using the optimal cell size we plot in Fig. 13 the cost of the algorithm versus n_c (left) and $n_c \log n$ (right). Again it appears to be linear in $n_c \log n$, and regression gives an exponent very close to 1: $a = 2.5322$ and $b = 1.0372$.

It is interesting to compare our algorithm to the state of the art at the time of the earliest algorithm. Alder and Wainwright in 1959 [1] report that for a 500-particle system, their algorithm running on an IBM 704 calculator could handle 500 collisions per hour. For a 5000 particle system, the current algorithm running on a Pentium III PC handles about 60 million collisions per hour, which is around 16,000 collisions per second.

2. PARTICLE LADEN FLOW

In principle, the algorithm described in Section 1.2 can be used to simulate any system of particles whose trajectories, in the absence of interaction with other particles, are known

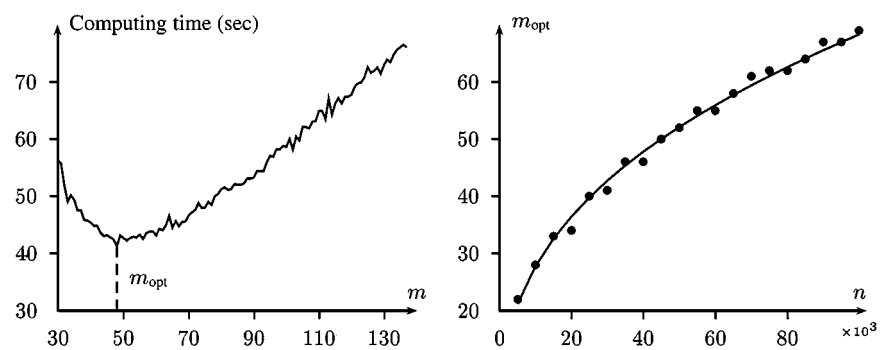

FIG. 12. 3D billiards, $\rho = 1\%$. Left: Computing time in seconds vs. m for $n = 50{,}000$. Right: m_{opt} vs. n in thousands (dots) and the fitted curve an^b, with $a = 0.7626$ and $b = 0.3904$ (solid); the predicted value is $b = \frac{1}{3}$.

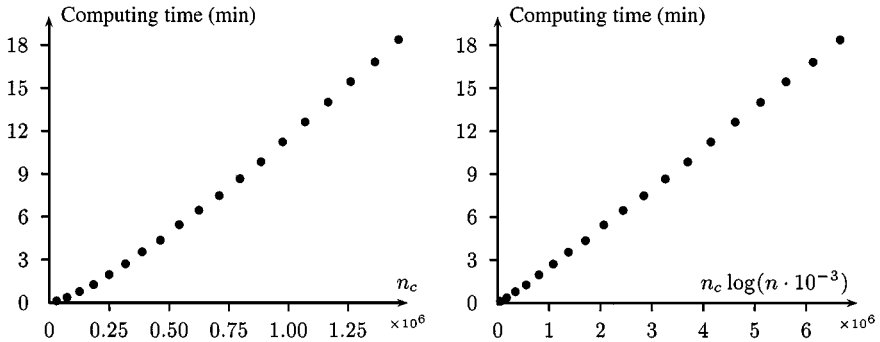

FIG. 13. 3D billiards, $\rho = 1\%$. Left: Computing time in minutes vs. n_c in millions. Right: Computing time in minutes vs. $n_c \log n$. The best fit of the form $a(n_c \log n)^b$ has $a = 2.5322$ and $b = 1.0372$.

in advance. All that is required is a way of computing the next collision time between any pair of particles, assuming they do not collide with other particles. Our aim however is to handle the more complicated particle trajectories of fluid suspensions, where the particles are immersed in a fluid, allowing for interchange of momentum and energy between the particles and the fluid. In such systems, the particle motion affects the surrounding fluid, so the trajectories cannot be integrated independently indefinitely, even in the absence of collisions. A simpler situation arises when only the fluid affects the immersed particles, and not vice versa, often termed particle laden flow, which is the object of study in this section.

A commonly used model for the effect of the fluid on the immersed particles is Stokes's law [24], and nonlinear corrections of it [25, pp. 16]. This law states that the force exerted by a fluid on an immersed particle is proportional to the relative velocity of the field and the particle, the radius of the particle, and the fluid viscosity. In dimensionless form, Stokes's law can be written

$$\tau \ddot{x}(t) = u(x(t), t) - \dot{x}(t), \tag{12}$$

where $\tau \propto r^\gamma$ is the so-called particle time-constant, $\gamma = 2$ in 3D [24, p. 229] and $\gamma = 2$ with log-correction in 2D [24, p. 246]. When, in Section 2.3, we do experiments we choose $\gamma = 1$ in 2D. (We are primarily concerned with the complexity of the algorithm when applied to nontrivial particle trajectories, so the experiments will still give useful information, despite nonphysical choice of exponent γ.) Furthermore, we will take the limit $n \to \infty$ in such a way that the particle volume density, ρ, is fixed, so $r \propto \frac{1}{\sqrt{n}}$.

In particle laden flow, even though the particle trajectories in the absence of collisions are in principle known, finding the next collision time of two particles whose trajectories are given by a differential equation is in general expensive computationally. In this section, we employ the algorithm from Section 1 on short incremental time intervals in which we can accurately approximate the particle motion (piecewise) linearly. Such an algorithm is, in any case, forced upon us for the problems in Section 3 where the velocity field depends on the particles. The modified algorithm is detailed in Section 2.1. We analyze the complexity of the modified algorithm in Section 2.2, but we emphasize that our statistical assumptions are far from being justifiable in this more general setting. However, we perform numerical experiments in Section 2.3 to test our conclusions and find that the statistical assumptions nonetheless lead to useful predictions. We find that, for driven flow problems, the scaling $m \sim n^{1/d}$ is optimal whenever the number of collisions grows with n at least as fast as in billiards.

2.1. The Algorithm

Given the developments in Section 1, a natural way to simulate (1) with F given by (12), is to apply the billiards algorithm, S1–S6, on short time intervals. That is, introduce a time step Δt, assume that the particle motion is linear, i.e., the particles move in straight lines with constant velocities, over each time step up to a collision, and apply the algorithm to the linear paths. If a particle pair collides during a time step, then in Step S2 *integrate* (as opposed to simply advance as in billiards) the paths of the two particles involved in the collisions up to that time, and then handle the collision. At the end of the time step, integrate all the particle paths from the time of their last collision, if any, or from the beginning of the time step if none. This work if care is taken in two respects.

Consistency of the Numerical Integrator and the Interpolation

First, the numerical integrator and the interpolation for the collision detection have to be consistent,[9] meaning that applying the integrator on a shorter time step than Δt will give the *same* particle position as the interpolation employed for the collision detection. Otherwise, the particle positions at the time of collision, as computed by the numerical integrator, might be such that the particles are overlapping, which will cause difficulties for the collision detection algorithm. For example, for the quadratic formula (2) to correctly predict when two particles are *touching*, the integrator used has to be consistent with the assumption that the motion is linear within a time step. In other words, it has to be linear in Δt for the position.

False Predictions

Secondly, even though two particles are *touching*, they will not necessary *collide* in the next instant; only if their velocities are such that they are approaching each other will they collide; see Fig. 14. Thus, before handling the collision, we must check that the particles are indeed colliding. If they are, we handle it in the usual manner, but otherwise ignore it.

This issue is not to be confused with the fact that a numerical integrator will not get the particle paths correct, and thereby give "false collisions." The only way to get no such false collisions is by computing the true trajectories of the particles exactly. However, if we assume that the motion *is* linear over each time step up to a collision, the quadratic formula for (2) will give false predictions for collisions, as indicated in Fig. 14, which should be dealt with as described above.

We reiterate that numerical errors introduced by the integrator will inevitably cause collisions to be added or missed. But apart from such errors, our algorithm does not miss (or add) a collision, no matter how large a time step or small cells are used, provided of course that the cell size is larger than a particle diameter. The algorithm is therefore "exact" in this sense; the detection of transfers ensures that particles that come close to each other at any time are checked for a possible collision.

Numerical Integrator

We use linear interpolation in a form useful when the particle time constant τ is small, and Eq. (12) becomes stiff. The consistency requirement makes it difficult to use a fully

[9] We are not using the term consistency in the standard sense applied to finite difference schemes.

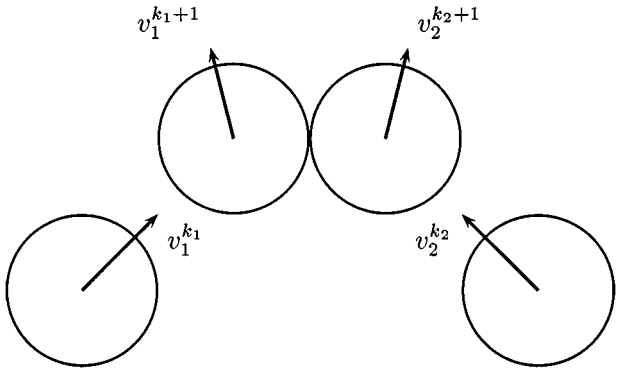

FIG. 14. False prediction for a collision between two particles moving in a velocity field. At the start of the time step the two particles have velocities $v_1^{k_1}$ and $v_2^{k_2}$. Based on constant velocities, the algorithm predicts a collision within the next time step. Integrating the particles toward the collision time reveals that they are indeed *touching* but, because of the effect of the velocity field, their velocities should be changed in such a way that they are not *colliding*.

implicit integrator; to handle this stiffness we use the *linearly implicit* integrator,

$$x^{k+1} = x^k + \Delta t v^k, \tag{13}$$

$$v^{k+1} = v^k + \frac{\Delta t}{\tau}(u(x^{k+1}, t^{k+1}) - v^{k+1}). \tag{14}$$

Since u is considered given in this section, this means (x^{k+1}, v^{k+1}) is uniquely determined from (x^k, v^k) for any time step $\Delta t > 0$. This scheme predicts position linear in Δt and is hence consistent with the collision detection formula (2).

Quadratic Interpolation

Alternatively, we could use an integrator that is quadratic in Δt for the particle positions. This approach has been used for molecular dynamics simulations with mixed hard-core and soft potentials [26]. An argument similar to the one that lead to Eq. (2) then gives that the next time any two particles are touching is the smallest positive root of a quartic. In general we still get false predictions, unless a specific numerical integrator is used. In particular, the integrator

$$\begin{aligned} x^{k+1} &= x^k + \Delta t v^k + \frac{1}{2}\Delta t^2 a^k, \\ v^{k+1} &= v^k + \Delta t a^k, \\ \text{where } a^k &= \frac{1}{\tau}(u(x^k, t^k) - v^k), \end{aligned}$$

will eliminate the false predictions altogether. The reason is that this scheme amounts to assuming constant acceleration within a time step, for which the quartic formula will correctly predict not only when two particles are touching, but also when they are colliding.

Time Step Initialization

Applying the billiards algorithm on small time steps means that we potentially have to set up the data structures at the beginning of every time step instead of just at the beginning of the simulation as in the billiards case; that is *very* often if the time step is small. The particle information is "initialized" by integrating the path of each particle from the time of its last collision or the beginning of the time step as we suggested initially. The cell structure is updated every time there is a transfer so at the end of a time step it has correct information on which cells the particles belong to and hence does not require initializing. The event queue, on the other hand, contains no useful information at all at the end of a time step, and will therefore need to be set up again at the beginning of each time step. The set-up cost of the event queue, which we deliberately ignored in the billiards case, will therefore enter our complexity analysis.

Piecewise Linear Paths

To avoid incurring this set-up cost at every time step we could, instead of applying the collision detection algorithm over a single time step, integrate the particle paths over a few, say k, time steps, store the computed trajectories, and apply the algorithm to the piecewise linear paths. We can then use Eq. (2) on each piece. It is then natural to ask how far should we integrate in time, that is how large should k be chosen? Larger k means less frequent set-up of the event queue. However, once a particle is involved in a collision its previously computed path becomes invalid, so integrating too far ahead in time is clearly bound to waste computational time. This suggests that there is an optimal k for which the running time of the algorithm is the least, and we will indeed see, theoretically in Section 2.2 and experimentally in Section 2.3 that, if the time step is considerably smaller than the mean collision time, there is an optimal $k > 1$.

This algorithm is a bit more complicated to implement than applying the collision detection at every time step, since each particle has to keep an array of states instead of a single state. Furthermore, this idea is not easily applicable to the more general case of coupled particle-fluid problems discussed in Section 3, so we focus primarily on the original scheme, that is with $k = 1$.

Previous Work

Sundaram and Collins [7] describe a similar approach they used to collect collision statistics in particle-laden turbulent flow. As described above, they discretize the trajectory of the particles and assume linear motion within a time step. They also employ a cell structure as described above, but instead of detecting transfers between cells, they *assume* a bound on the velocity of the particles, $v_{\max}\Delta t \leq L/2$, which ensures that only particles in adjacent cells can possibly collide within a time step. This assumption cannot be justified *a priori*.

They also report on their experience with using Verlet lists [27], and "overlap detection" instead of collision detection. Verlet lists are prominent in soft sphere simulations, in which the particles interact through a smooth potential, for example the Lennard–Jones potential [27]. Each particle keeps a list of its nearest neighbors, which is updated every few time steps. When used for hard sphere simulations, a bound on the velocity of each particle is needed. Sundaram and Collins [7] conclude that Verlet lists are less efficient

than the cell method they used, and that overlap detection is in most cases not sufficiently accurate.

It should be noted that to guarantee the algorithm of Sundaram and Collins [7] accounts for all collisions, even in the billiards case, an extremely short time step may be needed. For the billiard problem, as an example, the only known a priori upper bound on the particle velocity is when all the kinetic energy is contained in a single particle. That is, if the average particle speed is β, then $v_{\max}^2 = n\beta^2$ and hence the method of Sundaram and Collins [7] requires $\Delta t \leq L/(2\beta\sqrt{n})$. For billiards, the mean free collision time is proportional to $n^{-1/d}$, so if $d > 2$, the maximum time step to guarantee not missing collisions in an algorithm without transfer detection is an order of magnitude smaller than the mean free collision time.

Detecting transfers as well as collisions, which is not done in the Sundaram and Collins [7] approach, fixes this problem without a major increase in computational cost. Furthermore, since the restriction $v_{\max}\Delta t \leq L/2$ then no longer applies, it allows the use of smaller cells which also potentially reduces the cost.

2.2. Complexity

We now analyze the complexity of this modified algorithm. We assume that the statistical hypotheses A1–A4 we used in the billiards case remain true. This assumption often fails, but we shall see that the theory does have useful predictive capabilities.

First consider applying the collision detection over a single time step. The operation count (4) from Section 1.3 then applies to each time step, where the sum is now over events in the time interval $[t, t + \Delta t]$, and with the additional task of setting up the event queue at the beginning of every time step. The number of operations required for this setup task is

$$\mathcal{O}\left(\sum_{i=1}^{n}(1 + n_s^*(i)) + n\log n\right). \tag{15}$$

Here, similar to $n_s(i)$ in Section 1.3, $n_s^*(i)$ is the number of particles in the $(3^d + 1)/2$ cells surrounding and including the cell particle i is in, which are shaded in Fig. 4. The term $n\log n$ is the cost of setting up a heap of n elements.

Under assumption A1 in Section 1.3, namely that the particles are uniformly distributed throughout the domain at each instant in time, the average of (15) is

$$\mathcal{O}((1 + n_0 L^d)n + n\log n), \tag{16}$$

where L is the side length of a cell, $n_0 = n/D^d$ is the particle number density, and D is the side length of a cube containing all the particles, so $n_0 L^d$ is the average number of particles per cell. We do not expect the particles to be uniformly distributed in this more general setting, but anticipate though that the average number of particles per cell will be proportional to $n_0 L^d$. The average of (4) is still (5), where $\mathbb{E}n_e$ is the expected number of events in a time interval of length Δt.

In addition, at the beginning of each time step we need to integrate the path of each particle, costing $\Theta(n)$ operations. Since this term is already included in the above expression, we can safely ignore it.

Adding (16) and (5) we get the cost of each time step as

$$\mathcal{O}((1 + \log n + n_0 L^d)[\mathbb{E}n_e + n]), \tag{17}$$

where now $\mathbb{E}n_e$ is the expected number of events in a time interval of length Δt.

In Section 1.3 we used assumptions A1–A3 to write $\mathbb{E}n_t$ and $\mathbb{E}n_c$ in terms of n and m and assumption A4 to bound $\mathbb{E}n_{ch}$. These assumptions are clearly not justified in this more general setting. We anticipate however that the total number of transfers will still be proportional to βmnT, that is given by Eq. (7); this is born out in experiments. The total number of collisions will on the other hand not necessarily be given by Eq. (6).[10] We assume that it is proportional to some power of n,

$$\mathbb{E}n_c \sim \beta \Delta t n^{1+\alpha}. \tag{18}$$

For billiards we saw that $\alpha = 1/d$. For the number of checks, $\mathbb{E}n_{ch}$, we again use assumption A4.

Plugging (7) with $T = \Delta t$ and (18) into (17) we get

$$\mathcal{O}\left((1 + \log n + n_0 L^d)\left(\left(n^\alpha + \frac{1}{L}\right)\beta n \Delta t + n\right)\right),$$

and summing over all $T/\Delta t$ time steps, that is multiplying by $T/\Delta t$, we get the total complexity as

$$\mathcal{O}\left((1 + \log n + nL^d)\left(n^\alpha + \frac{1}{L} + \frac{1}{\beta \Delta t}\right)\beta T n\right).$$

Since $n_0 \sim n$ and $m = 1/L$, in our chosen units we can write this as

$$\mathcal{O}\left(\left(1 + \log n + \frac{n}{m^d}\right)\left(n^\alpha + m + \frac{1}{\Delta t}\right) Tn\right),$$

Choosing $m^d = \Theta(n)$ keeps the first factor $\mathcal{O}(\log n)$, so the cost is

$$\mathcal{O}\left(\left(n^\alpha + n^{1/d} + \frac{1}{\Delta t}\right) n \log n\right). \tag{19}$$

The mean free collision time, that is the mean time between successive collisions of a single particle, is

$$\tau_c = \frac{T_n}{n_c} = \frac{\beta}{n^\alpha}.$$

Also, L/β is the time it takes a particle to travel one cell length. We therefore notice from (19) that in order for the overhead added by the time stepping not to be dominant we need $\Delta t = \Omega(\tau_c)$ and $\Delta t = \Omega(L/\beta)$, that is *the time step should not be much smaller than either the mean time between collisions or the time it takes to travel a single cell length*. For particle laden flow, the size of the time step is determined by the time scale of the particle motion and hence we are not free to choose it to optimize the complexity. The above analysis therefore indicates that if the characteristic time scale of the particle motion is much smaller than the mean collision time, the dominant cost is checking for collisions and transfer and setting up the priority queue at the beginning of each time step.

[10] Indeed, one use of a collision detection algorithm like this is collection of collision statistics, such as the total number of collisions, in particle-laden flow.

Therefore, if $\alpha \geq 1/d$ and we choose $\Delta t = \Omega(n^{-1/d})$ (for example, keep Δt fixed), the complexity is $\mathcal{O}(n_c \log n)$. If on the other hand $\alpha < 1/d$, the complexity is $\mathcal{O}(n^{1+1/d} \log n)$ which is not optimal.

Piecewise Linear Paths

Now consider applying the collision detection less frequently than at every time step, say every k time steps. Then the event queue need only be set up $T/(k\Delta t)$ times, instead of $T/\Delta t$ times. Of course, this comes at the expense of more costly collision and transfer checks; finding the next transfer time of a particle or next collision time of two particles whose trajectories are piecewise linear with k pieces costs up to k times more than before. We proceed as before, using the analysis from Section 1.3 with $T = k\Delta t$, and add the previously mentioned set-up cost every k time steps, for a total cost of

$$\mathcal{O}\left((k + \log n + knL^d)\left(n^\alpha + \frac{1}{L} + \frac{1}{k\beta\Delta t}\right)\beta T n\right). \tag{20}$$

With the choice of cell size $L = \Theta(n^{-1/d})$, the optimal k is then such that both $k\Delta t = \Omega(\tau_c)$, that is, $k\Delta t$ is asymptotically larger than the mean free collision time, and $k = \mathcal{O}(\log n)$, if both are possible. Whether or not this is possible depends on how the number of collisions scales with n and how Δt is chosen. If for example Δt is chosen on the order of τ_c, then this is always possible since then the first condition is $k = \Omega(1)$. If Δt is fixed and $n_c = \Theta(n^{1+\alpha})$ then $\tau = \Theta(n^{-\alpha})$ so the first condition is $k = \Omega(n^{-\alpha})$ and hence $k = \mathcal{O}(\log n)$ as long as $\alpha \geq 0$ ($\alpha < 0$ means that the number of collisions decays as $n \to +\infty$). Of course the proper choice of k depends on the constants, which we have ignored, but a rule of thumb is to choose k such that $k\Delta t$ is on the order of the mean collision time.

2.3. Experiments

As before, we compare the analytical results from the preceding section with timings obtained from running the algorithm. All the experiments are performed with fixed particle volume density and zero initial kinetic energy as n increases. We restrict our attention to 2D particle laden flow with elastic collisions. We place the particles in a 2D incompressible velocity field $u = \nabla^\perp \psi = (\frac{\partial \psi}{\partial x_2}, -\frac{\partial \psi}{\partial x_1})$, where ψ is the stream function.

Taylor–Green Flow

We first use the Taylor–Green flow [28, 29], which is a solution to the forced 2D Navier–Stokes equations in the unit square with periodic boundary conditions and the initial conditions shown in Fig. 15, namely

$$\psi(x, 0) = \frac{1}{2\pi} \sin 2\pi x_1 \sin 2\pi x_2.$$

To get a time-independent flow from this initial condition it is necessary to set the force to $f(x) = \nu \nabla^\perp \psi(x, 0)$. In Appendix A.1 we show that then $\psi(x, t) = \psi(x, 0)$, so

$$\begin{aligned} u_1(x, t) &= \sin 2\pi x_1 \cos 2\pi x_2, \\ u_2(x, t) &= -\cos 2\pi x_1 \sin 2\pi x_2. \end{aligned}$$

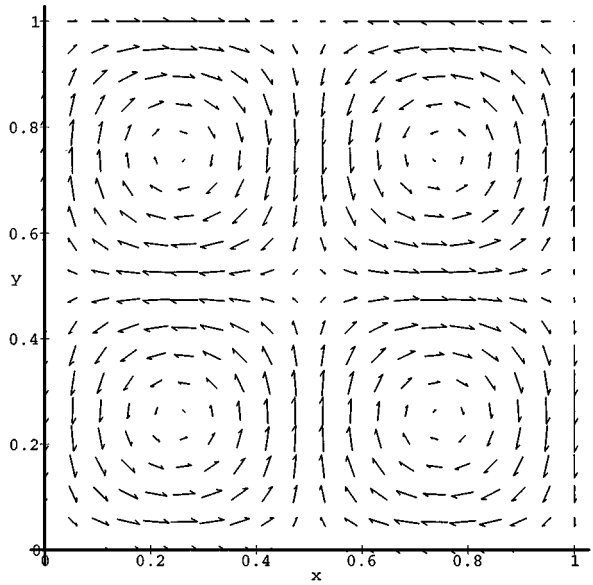

FIG. 15. The initial conditions for the Taylor–Green flow.

The point of this velocity field is to test the algorithm when there is an equilibrium particle distribution which is nonuniform in space. Figure 16 (left) shows a typical initial particle configuration used for the experiments. The particles quickly start spiraling outwards and after a few time units they are distributed as on the right; as expected, the particles are not uniformly distributed in space. Also, the spatial and velocity distributions are now highly correlated; the particle velocity is small near the four saddles at the center and at the corners, and larger between them. That is, our assumptions for the complexity analysis are certainly not satisfied.

Adaptive Cell Size Selection

The time-dependence of the particle distribution suggests that using a fixed cell size throughout the simulation is perhaps not the most efficient strategy. Initially, when the particles are distributed as on the left in Fig. 16, relatively large cells are probably most efficient, whereas in equilibrium, when the particle distribution is as on the right, much smaller cells should be used. We therefore also include experiments with the following simple adaptive scheme. We monitor the running time of the algorithm, and every few time steps we decrease the cell size. We do this until the running time ceases to decrease. Then we start increasing the cell size again until the running time ceases to decrease. We continue this throughout the simulation, always heading in the same direction as long as the running time is decreasing. Below we compare the performance of this method to the performance of keeping m fixed throughout the simulation. Our main focus is, however, on keeping m fixed throughout the simulation, so unless otherwise stated, that is the method we use.

Results

We run the algorithm with n ranging from 10,000 to 200,000 in increments of 10,000. We start with the particles at rest and distributed as in Fig. 16 (left), use a time step $\Delta t = 10^{-2}$,

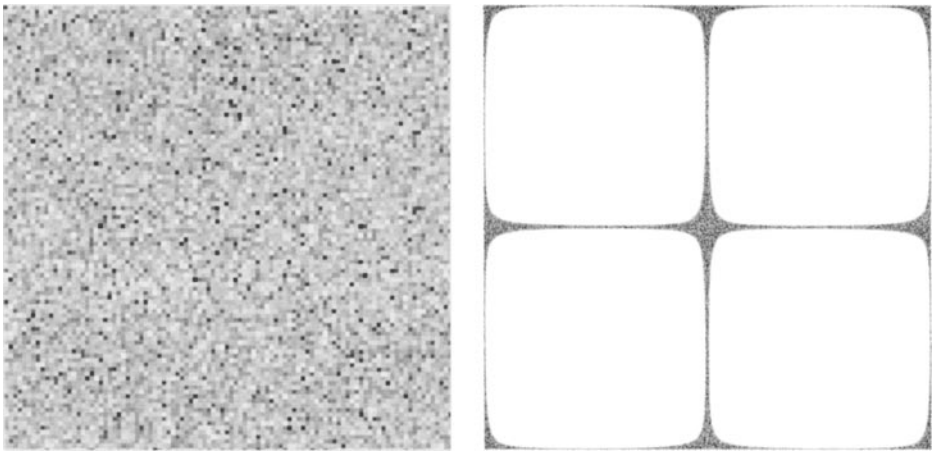

FIG. 16. Initial (left) and equilibrium (right) particle distribution for particle laden Taylor–Green flow with 50,000 particles.

and run for 10 time units at which time the particles are distributed as shown in Fig. 16 (right).

Figure 17 (left) shows how the total number of collisions, n_c, increases with the number of particles. A fit of the form an^b gives $b = 1.61$, which is slightly larger than $1 + 1/d = 1.5$, and thus we predict that by choosing $m \sim \sqrt{n}$ the cost will scale like $n_c \log n$. We take $m = 3(\sqrt{n})$, either fixed throughout the simulation or as the starting value for the adaptive scheme. Figure 17 shows how the total number of transfers increases with the product of m and n. A fit of the form $n_t = a(mn)^b$ gives that $b = 1.02$ so the number of transfers is very nearly linear in mn as we predicted.

Figure 18 (left) shows how the computing time increases with the number of collisions. It appears slightly super-linear, and a fit of the form $C = an_c^b$ indeed gives $b = 1.18$. After dividing the cost by $\log n$, Fig. 18 (right), the best fit is $C \sim n_c^{1.13} \log n$.

From Fig. 18 we see that the simple adaptive scheme performs better than the fixed scheme, though it does not change the asymptotic running time. The simulation of 200 thousand particles took a bit less than 48 hours of computing time and resulted in almost 1 billion collisions, giving an average of 20 million collisions an hour, or 5500 collisions per second.

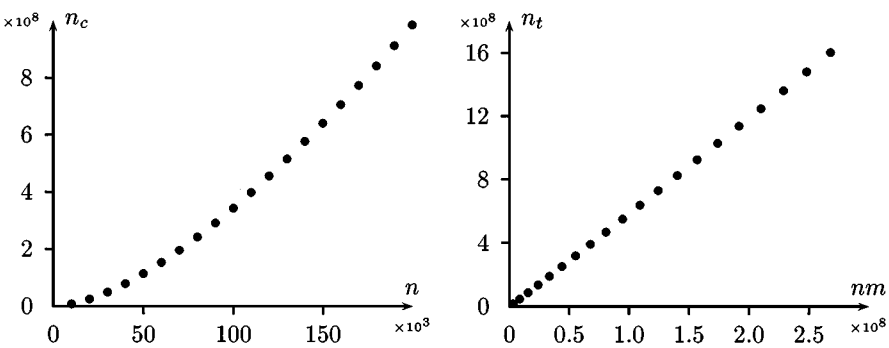

FIG. 17. Number of events for particle laden Taylor–Green flow. Left: Number of collisions in 100 millions vs. number of particles in thousands. The best fit is $n_c = 2.85n^{1.61}$. Right: Number of transfers in 100 millions vs. nm in 100 millions. The best fit is $n_t = 4.32(nm)^{1.02}$.

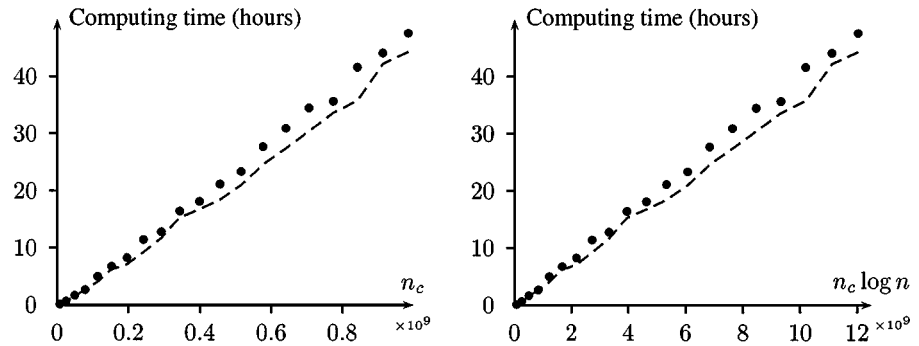

FIG. 18. Computing time for particle laden Taylor–Green flow for fixed m (dots) and the adaptive scheme (dashed). Left: Computing time in hours vs. number of collisions in billions. The best fit is $C \sim n_c^{1.19}$. Right: Computing time in hours vs. $n_c \log n$ in billions. The best fit is $C \sim n_c^{1.13} \log n$.

Synthetic Turbulence

In order to gain insight into the behavior of colliding particles in a 2D turbulent velocity field, we create an incompressible flow field with prescribed mean spectral properties; these can be chosen to match theoretical predictions or empirical observations about energy scaling laws in turbulence. This is done by setting up a linear stochastic PDE for the stream function ψ, as described in Appendix 2. The point is that particles in this velocity field only reach a *statistical* equilibrium, and their configuration is nonuniform in both space and time.

For the experiments we choose the spectrum of the velocity field to be the Kárman–Obukhov spectrum [30, pp. 112],

$$\varepsilon_k \sim \|k\|^2 (1 + \|k\|^2)^{-7/3},$$

shown in Fig. 19, which was introduced to study Kolmogorov turbulence. Here ε_k is the mean energy in wavenumber k (see Appendix 2). We have also experimented with the Kraichnan spectrum [30, pp. 113],

$$\varepsilon_k \sim \|k\|^2 \exp(-\|k\|^2),$$

shown in Fig. 19, obtaining identical results regarding the complexity of the algorithm.

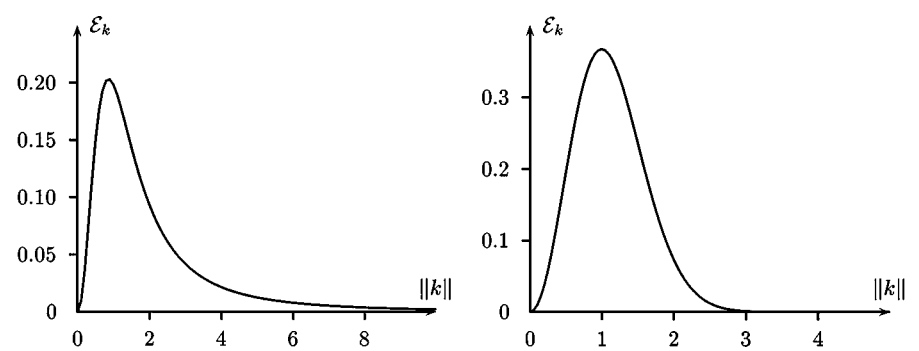

FIG. 19. Spectra used for experiments with synthetic turbulence. Left: Kárman-Obukhov spectrum. Right: Kraichman spectrum.

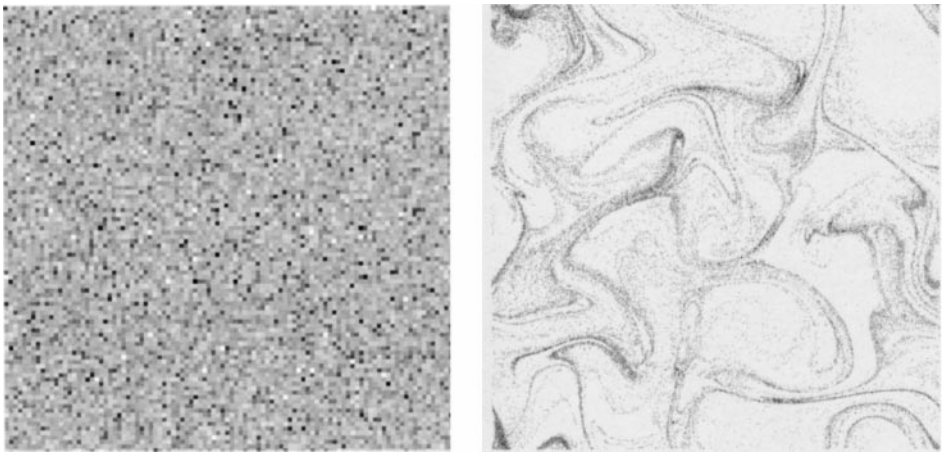

FIG. 20. Initial (left) and typical (right) particle distribution for particle laden synthetic turbulence.

Results

We run the algorithm with n ranging from 10,000 to 200,000 in increments of 10,000. We start with the particles at rest and uniformly distributed in space. We use a time step $\Delta t = 10^{-2}$ and run for 10 time units, at which time the particles are distributed as in Fig. 20 (right). Note that the distribution is not uniform in space.

Figure 21 (left) shows how the total number of collisions, n_c, increases with the number of particles. This is different from both billiards and the Taylor–Green case, and appears to be linear in n. A fit of the form an^b to the second half of the data gives $b = 0.82$, which is far less than $1 + 1/d = 1.5$. We therefore do not expect the algorithm to be optimal on this problem. By choosing $m \sim \sqrt{n}$, our analysis predicts that the cost will scale like $n^{3/2} \log n$, which is far greater than $n_c \log n$. We again take $m = 3(\sqrt{n})$, either fixed throughout the simulation or as the starting value for the adaptive scheme. Figure 21 (right) shows how the total number of transfers increases with the product of m and n. A fit of the form $n_t = a(mn)^b$ gives that $b = 1.04$ so the number of transfers is still very nearly linear in mn.

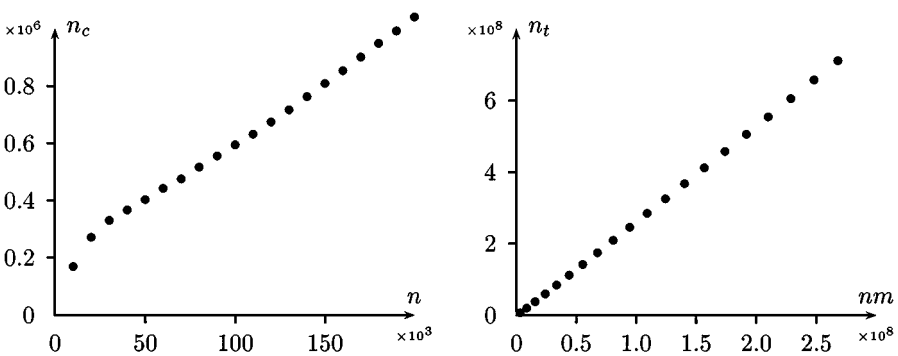

FIG. 21. Number of events for particle laden synthetic turbulence. Left: Number of collisions in millions vs. number of particles in thousands. A fit of the form an^b to the second half of the data gives $b = 0.82$. Right: Number of transfers in 100 millions vs. nm in 100 millions. The best fit is $n_t = 1.39n^{1.04}$.

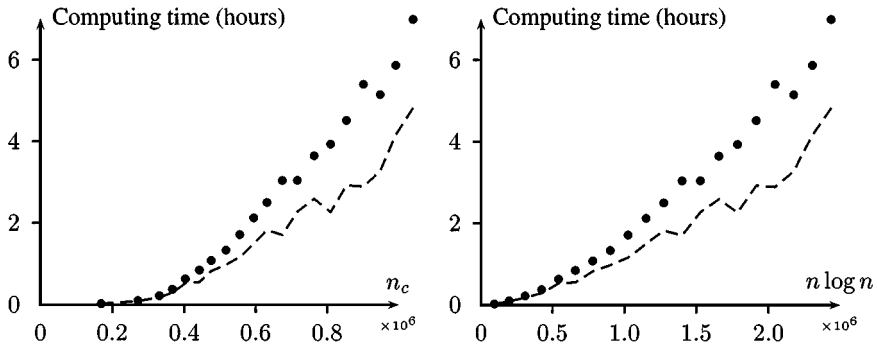

FIG. 22. Computing time for particle laden synthetic turbulence for fixed m (dots) and the adaptive scheme (dashed). Left: Computing time in hours vs. number of collisions in millions. Right: Computing time in hours vs. $n \log n$ in millions. The best fit from the second half of the data is $C \sim n^{1.54} \log n$.

Figure 22 (left) shows how the computing time increases with the number of collisions and as expected it is far from linear. Figure 22 (right) shows the cost divided by $\log n$ versus n. The best fit, using only the second half of the data, is $C \sim n^{1.54} \log n$, quite close to the prediction of $C \sim n^{3/2} \log n$. Recall that Fig. 21 shows that the number of collisions is sublinear in n; thus, as predicted, the algorithm is far from optimal in this case.

From Fig. 22 we again see that the simple adaptive scheme performs better than the fixed scheme, though it does not change the asymptotic running time. The simulation of 200 thousand particles using the adaptive scheme took a bit less than five hours of computing time and resulted in more than 1 million collisions, giving an average of a mere 200 thousand collisions an hour, or 55 collisions per second.

Optimal Time Step

In Section 2.2 we concluded from Eq. (19) that in order for the overhead added by the time stepping not be dominant, the time step should be chosen not much smaller than the mean free time. We test this conclusion experimentally by running the algorithm with different time step sizes. We use Taylor–Green flow with 10,000 particles and the same setup as before. For this set-up, there are about 7.5 million collisions, giving a mean collision time of

$$\tau_c = \frac{nT}{2n_c} \approx 7 \times 10^{-3}.$$

Figure 23 (left) shows the cost versus size of time step.

We see that the cost is not sensitive to the size of the time step, as long as it is large enough; however, once the time step becomes small compared to the mean free collision time, the cost is inversely proportional to the size of the time step. This is precisely what is predicted by (19).

Piecewise Linear Paths

In the experiments above we chose $\Delta t = 10^{-2}$, which is on the order of τ_c. Our analysis in Section 2.2 indicates that we would not benefit from applying the collision detection less frequently with this time step, that is use $k > 1$. Recall that for $k > 1$ our complexity analysis gives (20) rather than (19) for the cost.

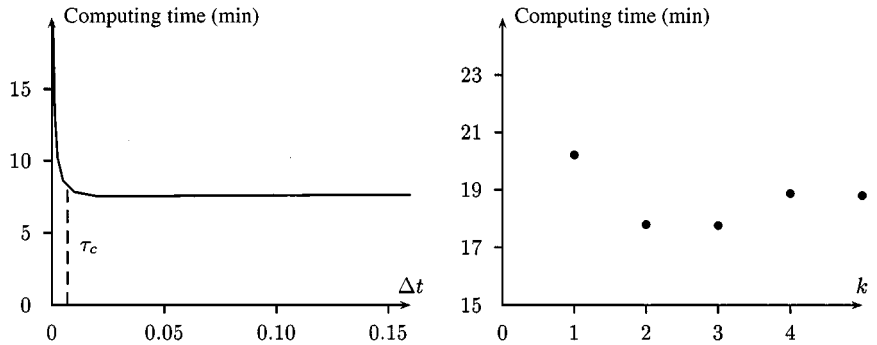

FIG. 23. Computing time in seconds vs. Δt for particle laden Taylor–Green flow, $n = 10000$. $\tau_c = nT/2n_c = 7 \times 10^{-3}$ is the mean free collision time. Left: Computing time vs. Δt. Right: Computing time vs. k with $\Delta t = 10^{-3}$.

To test the conclusions of (20), we use a smaller time step, $\Delta t = 10^{-3}$. Figure 23 shows how the running time varies with k.

There is a minimum at $k = 3$, which means that it is optimal to apply the collision detection over an interval of three time steps, which is about half the mean free time. For $n = 10000$, $\log n$ is quite small, so the restriction $k = \mathcal{O}(\log n)$ kicks in early and prevents $k\Delta t$ from being closer to τ_c. We see that if the mean collision time is much larger than the time step used for integration, this idea can reduce the cost quite significantly.

3. COUPLED PARTICLE-FLOW

In particle laden flow, the fluid exerts a force on the immersed particles. By Newton's third law, the particles then exert an equal and opposite force on the fluid. In some applications, this back-coupling is thought to be important. The exact solution of such fluid-particle flows requires the solution of the Navier–Stokes equations with a free moving boundary corresponding to the surface of the particles. The numerical solution of the resulting equations is tractable for small and moderate values of n [31] but for large n this approach is computationally intractable. A simplified model of this back-coupling effect consists of adding to the continuum balance laws point sources of mass, momentum, and energy [25, pp. 7–23]. In this setting, any motion of a particle affects the surrounding fluid, and hence affects other particles instantaneously. Therefore, the particle paths cannot be integrated independently of each other indefinitely in the absence of collisions; small time steps must be used and approximate independence invoked.

In Section 3.1 below we describe how we apply the algorithm described in Sections 1 and 2 to such coupled particle-fluid flows. Now there is the additional task of solving a PDE with a force term consisting of n delta functions. A key question regarding computational cost is the relative cost of solving the PDE and performing the collision detection. In Section 3.2, we assess the complexity of the two parts of the algorithm, the collision detection and the numerical solution of the PDE, and compare the two. In Section 3.3 we verify the analysis through experiments. We find that, under a natural limiting process, the choice $m \sim n^{1/d}$ is optimal for these coupled problems, no matter how the number of collisions scales with n; this contrasts with the driven flow case.

For the experiments, we take for simplicity u as the solution to the diffusion equation[11]

$$\frac{\partial u}{\partial t}(x,t) = \nu \Delta u(x,t) + f(x,t) - \alpha \sum_{i=1}^{n} (u(x_i(t),t) - \dot{x}_i(t))\delta(x - x_i(t)), \tag{21}$$

with periodic boundary conditions in the unit square, $\Omega = [0,1] \times [0,1]$, for some chosen $f : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2$. Here ν and α are dimensionless constants. This is coupled to the particles obeying (12) together with elastic collisions. For fluid suspensions, the relevant PDE is the Navier–Stokes equation, but our main purpose is to understand the complexity of coupled particle-fluid algorithms, for which (21) is adequate. As before, we will take the limit $n \to \infty$ in such a way that ρ is fixed, and then $\alpha = r \propto \frac{1}{\sqrt{n}}$ and for Stokes's' law $\tau \propto \frac{1}{\sqrt{n}}$. This scaling is chosen so that formal arguments indicate that the sum of delta functions in (21) tends to a smooth correction to the PDE for u in the limit $n \to +\infty$.

3.1. The Algorithm

To detect collisions we proceed as for particle laden flow, with the sole addition of solving the PDE (21). At any time t, we solve the PDE over the time interval $[t, t + \Delta t]$, given the particle positions at time t. Then we invoke the collision detection algorithm over the time interval $[t, t + \Delta t]$ as described in Section 2.1, integrating the particle paths numerically as necessary with the scheme (13) and (14), with u frozen at the previously computed solution of the PDE at time $t + \Delta t$.

To solve the PDE (21), we use a method implicit in the diffusion term and linearly implicit in the delta sources as described in Appendix 3. Our choice of method (A.6) for solving the PDE rather than (A.5) is dictated by an interesting numerical instability in (A.5), shown in Fig. 1 in Appendix 3, which results when many particles cluster together. The implicit method appears to cure this instability.

3.2. Complexity

In addition to the cost of collision detection, we now have the cost of solving a PDE. We consider a situation where the number of particles scales like the number of mesh points and, since we will use a method implicit in the diffusion term, we will take the time step to scale like the space step. Thus, if $N = (\Delta x)^{-d}$ is the total number of mesh points, we take $n = \Theta(N)$ and $\Delta t = \Theta(N^{-1/d})$.

In some circumstances, such as for the simple diffusion equation (21), a Fourier based solver can be used to solve the PDE, and the resulting complexity is $\Theta(N \log N + n) = \Theta(N \log N)$, per time step. We study such Fourier-based methods as they minimize the cost of solving the PDE for an implicit method, and allow us to assess the additional relative cost of collision detection in a worst case setting; for PDEs where Fourier methods cannot be employed we anticipate a lower relative cost for collision detection.

To solve the linear equations arising in the PDE, we use the conjugate gradient (CG) method, preconditioned by the solution with explicit treatment of the delta source terms. This preconditioning can be performed using the FFT in $\Theta(N \log N)$ operations. Hence,

[11] In dimensions 2 or more, it will be difficult to make sense of this model without regularizing the delta function; otherwise the velocity field will be unbounded at the particle locations. However, the issue of the computational cost of fully coupled particle-flow models can be addressed through (21), relying on spatial discretization to regularize the delta-singularities.

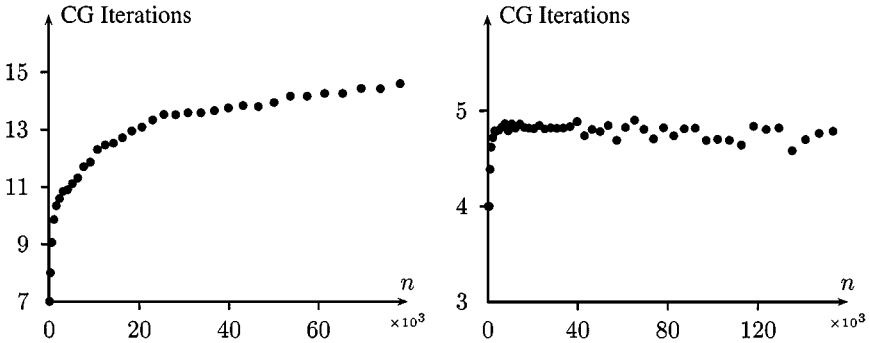

FIG. 24. Average number of CG iterations per time step vs. n. Left: Taylor–Green forcing. Right: Stochastic forcing.

if the number of CG iterations is bounded independently of N, then the total cost of the linear solver is $\Theta(N \log N)$. In practice, only a few CG iterations are used each time step and the number of iterations is roughly independent of N as demonstrated in Fig. 24 for the experiments of Section 3.3. The graphs show the total number of CG iterations over the entire simulation divided by the total number of time steps, as $n = \Theta(N)$ increases.

In summary, the cost of solving the PDE over the time interval $[0, T]$ is

$$\Theta\big(N^{1+1/d} \log N\big). \tag{22}$$

To estimate the cost of the collision detection, we make the same statistical assumptions as in Sections 1.3 and 2.2; note that, as in Section 2.2, these statistical assumptions are of limited validity. The cost is then, from (18) and (19) with $\Delta t = \Theta(n^{-1/d})$,

$$\mathcal{O}\big(\big(n_c + n^{1+1/d}\big) \log n\big). \tag{23}$$

Adding (22) and (23) the total complexity is

$$\mathcal{O}\big(\big(n_c + n^{1+1/d}\big) \log n\big), \tag{24}$$

since $n = \Theta(N)$, that is the number of mesh points and the number of particles are kept proportional as they are increased. In particular, if the number of collisions grows *slower* with n than in billiards, that is $n_c = \mathcal{O}(n^{1+1/d})$, the cost of collision detection does not add asymptotically to the cost of solving the coupled problem. If the number of collisions grows *faster* than in billiards, the collision detection is more expensive but optimal. So the combined algorithm is optimal, even in situations where the collision detection algorithm is not optimal for particle-laden flow. Under our statistical assumptions, the choice $m \sim n^{1/d}$ therefore appears optimal for coupled problems.

3.3. Experiments

As before, we perform a few numerical experiments to validate the analysis in the previous section. All the experiments are performed with fixed particle volume density and zero initial kinetic energy as n increases and the scalings detailed after Eq. (21). We use two different forces f. On the one hand, we let f be the Taylor–Green flow from Section 2.3, $f = \nabla^{\perp}\psi$

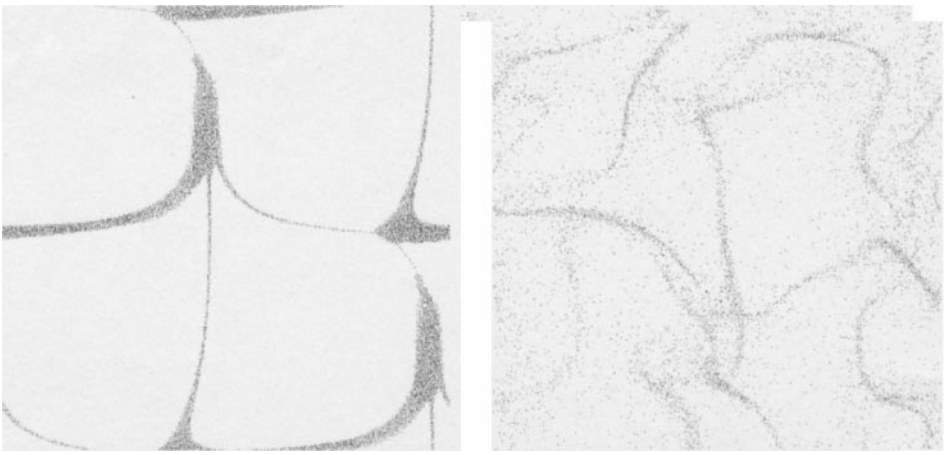

FIG. 25. Particle distribution for the coupled problem at time $t = 10$ with 10,815 particles. Left: Taylor–Green forcing. Right: Stochastic forcing.

with $\psi(x) = \frac{1}{2\pi} \sin 2\pi x_1 \sin 2\pi x_2$, and the stochastic force used in Appendix 2 to generate synthetic Kárman–Obukhov turbulence, $f = \frac{dW}{dt}$, on the other hand.

Without the back-coupling (no delta functions in (21)), the steady state of u with Taylor–Green forcing is the Taylor–Green velocity field we used in Section 2.3, so we might expect a similar particle distribution as before. To ease direct comparisons between the two experiments, we use the same parameter values as in Section 2.3. Starting from an initially uniform particle distribution, after two time units the distribution is the same as in the laden case, but around five time units it breaks up, and after 10 time units the particle distribution is as shown in Fig. 25 (left). This shows that the particles are having a significant effect on the flow and back-coupling is important. Figure 25 right shows the particle distribution at time $t = 10$ for stochastic forcing.

Results

To assess the relative cost of the collision detection and the numerical solution of the PDE, we run the algorithm with n, N, and Δt varying jointly so that $n = N$ and $\Delta t = 1/\sqrt{N}$, as described above.

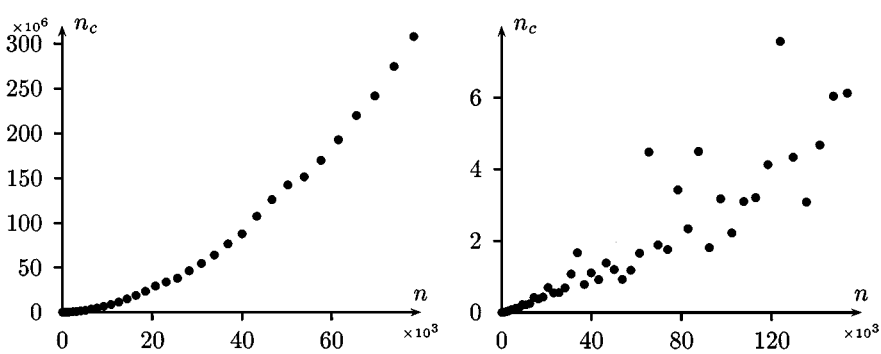

FIG. 26. Number of collisions in millions vs. number of particles. Left: Taylor–Green forcing. The best fit is $n_c = 0.24n^{1.87}$. Right: Stochastic forcing. The best fit is $n_c = 0.78n^{1.33}$. Different realizations of the stochastic force are used for different particle numbers which explains the large variation in the number of collisions.

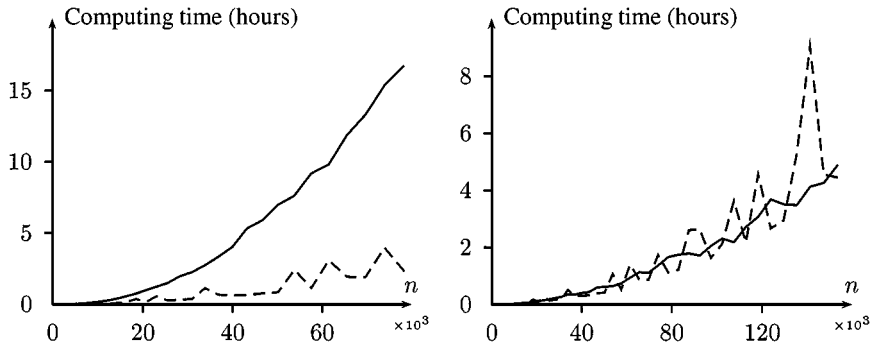

FIG. 27. Computing time spent in collision detection, C_1 (solid), and in solving the PDE, C_2 (dashed), in hours vs. n for the coupled problem. Left: Taylor–Green forcing. Best fit is $C_1 \sim n^{1.92} \log n$ and $C_2 \sim n^{1.68} \log n$. Right: Stochastic forcing. The best fit is $C_1 \sim n^{1.67} \log n$ and $C_2 \sim n^{1.63} \log n$.

The relative cost of the two contributions depends on the number of collisions. Figure 26 shows the number of collisions for Taylor–Green forcing (left) and stochastic forcing (right). A fit of the form $n_c = an^b$ gives $b = 1.87 > 1.5$ for Taylor–Green and $b = 1.33 < 1.5$ for the stochastic forcing. We therefore expect the collision detection to be asymptotically more expensive in the former case, but the cost of the two contributions to be comparable in the second case. This is indeed confirmed in Fig. 27 which shows the cost versus the number of particles.

Both experiments indicate that the cost of the two contributions is well predicted by the heuristic analysis in Section 3.2. Furthermore, if the number of collisions grows slower with n than in the billiards case, the two contributions are comparable. It is important to note however, that we have kept the cost of solving the PDE to a bare minimum; for more complex PDEs we anticipate that the cost of the PDE solver will be greater, making the relative cost of collision detection less.

4. CONCLUSIONS

In this paper we have presented and analyzed a collision detection algorithm for a large number of particles moving in a velocity field. We have

- Given an average case analysis of the complexity of the algorithm in the billiards case, under reasonable empirical assumptions, arriving at the observed fact that the optimal choice of cell size is to have a constant number of particles per cell and that the algorithm is optimal to within a logarithmic factor.
- Extended the event driven cell-based algorithm, developed by computational chemists and computer scientists for the billiards problem, to particle-laden flow and coupled particle-flow problems.
- Given numerical evidence to show that the analysis of the billiards algorithm gives useful predictions for optimal cell-scaling and complexity for problems where Boltzmann-like statistics do not prevail, such as particle laden flow and coupled particle-flow problems, and that the collision detection algorithm is optimal if and only if the number of collisions grows at least as fast with n as it does in billiards.
- Shown that for coupled particle-field simulation where the number of mesh points and particles are commensurate, our algorithm for collision detection is either optimal (when the number of collisions grows at least as fast with n as in billiards) or can be included in such simulations without increasing the asymptotic growth of the cost.

• Identified, and cured, an interesting numerical instability arising in coupled particle-field problems.

APPENDIX 1. THE TAYLOR–GREEN FLOW

Let $u = \nabla^\perp \psi = (\frac{\partial \psi}{\partial x_2}, -\frac{\partial \psi}{\partial x_1})$ be the velocity field, where ψ is the stream function. Let ω be the vorticity, $\omega = \nabla \times u = \nabla \times \nabla^\perp \psi$, so $\omega_3 = -\Delta\psi$. Now take the curl of the Navier–Stokes equation,

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\nabla_p + \nu \Delta u + \nu f,$$

where $f(x) = 4\pi^2 u(x, 0)$, and use

$$\nabla \|u\|^2 = 2u \cdot \nabla u + 2u \times (\nabla \times u) = 2u \cdot \nabla u - 2\omega \times u$$

to get

$$\frac{\partial \omega}{\partial t} + \nabla \times (\omega \times u) = \nu \Delta \omega + 4\pi^2 \nu \omega(x, 0).$$

With the initial conditions

$$\omega_1(x, 0) = \omega_2(x, 0) = 0 \quad \text{and} \quad \omega_3(x, 0) = -\Delta\psi(x, 0) = 4\pi \sin 2\pi x_1 \sin 2\pi x_2$$

the nonlinearity $\nabla \times (\omega \times u)$ vanishes for all time, and taking the Fourier transform gives the Taylor–Green [28, 29] solution

$$\omega_3(x, t) = 4\pi \sin 2\pi x_1 \sin 2\pi x_2$$
$$\text{so that} \quad \psi(x, t) = \frac{1}{2\pi} \sin 2\pi x_1 \sin 2\pi x_2.$$

APPENDIX 2. SYNTHETIC TURBULENCE

In this appendix we describe how to generate a two-dimensional incompressible turbulent velocity field u, periodic in the unit square, with the properties of being homogeneous, stationary, isotropic, and Gaussian [30, pp. 108–113; 32].

To ensure incompressibility of u, we work with the stream function ψ and set $u = \nabla^\perp \psi$. We take ψ as the solution to the following stochastic PDE:

$$d\psi + \nu A \psi \, dt = dW. \tag{A.1}$$

Here A is a linear operator with eigenfunctions $\{e_k\}_{k \in K}$, and eigenvalues $\{\alpha_k\}_{k \in K}$. We take $A = -\Delta$ in the unit square with periodic boundary conditions so

$$K = 2\pi \mathbb{Z}^2 \setminus \{(0, 0)\}, \quad e_k(x) = e^{1k \cdot x}, \quad \text{and} \quad \alpha_k = \|k\|^2.$$

W is a Q-Wiener process,

$$W(x, t) = \sum_{k \in K} \sqrt{\lambda_k} \beta_k(t) e_k(x),$$

for some operator Q with $Qe_k = \lambda_k e_k$, where $\{\beta_k\}_{k\in K}$ is an i.i.d. sequence of standard complex valued Brownian motions. Q is the covariance operator of W, and its spectrum, $\{\lambda_k\}_{k\in K}$, is chosen so that the velocity field u has the desired energy spectrum; see [32]. For a rigorous interpretation of this equation, see [33].

Expanding ψ in eigenfunctions of A,

$$\psi(x,t) = \sum_{k\in K} \hat{\psi}_k(t) e_k(x),$$

we get the following Ornstein–Uhlenbeck stochastic differential equations for the Fourier coefficients:

$$d\hat{\psi}_k + \nu\alpha_k \hat{\psi}_k\, dt = \sqrt{\lambda}_k\, d\beta_k, \quad k \in K. \tag{A.2}$$

The solutions can be expressed as

$$\hat{\psi}_k(t) = e^{-\nu\alpha_k t}\hat{\psi}_k(0) + \sqrt{\lambda}_k X_k(t), \tag{A.3}$$

where

$$X_k(t) = \int_0^t e^{-\nu\alpha_k(t-\tau)}\, d\beta_k(\tau)$$

is a complex valued Gaussian process with independent increments and variance

$$\int_0^t e^{-2\nu\alpha_k(t-\tau)}\, d\tau = \frac{1}{2\nu\alpha_k}(1 - e^{-2\nu\alpha_k t}). \tag{A.4}$$

Also, if $k \neq k'$, X_k and $X_{k'}$ are independent. Letting $t \to \infty$ in (A.3) we get that the stationary distribution of $\hat{\psi}_k$ is Gaussian with variance $\lambda_k/2\nu\alpha_k$. Now the Fourier transform of the velocity field $\hat{u}$ is $\hat{u}_k = (-ik_2\hat{\psi}_k, ik_1\hat{\psi}_k)$, so the energy spectrum of u is

$$\varepsilon_k = \mathbb{E}\|\hat{u}_k\|^2 = \|k\|^2\mathbb{E}|\hat{\psi}_k|^2 = \|k\|^2\frac{\lambda_k}{2\nu\alpha_k} = \frac{\lambda_k}{2\nu};$$

hence we choose $\lambda_k = 2\nu\varepsilon_k$ to achieve the spectrum ε_k.

A.2.1. Implementation

To generate the velocity field on an $N_1 \times N_2$ grid and at discrete times $\{j\Delta t\}_{j=0}^{J}$, for some $\Delta t > 0$, we proceed as follows. We use Eq. (A.3) for the $N_i - 1$ lowest modes in each dimension, that is with

$$k \in \{(2\pi j_1, 2\pi j_2) | j_i = -N_i/2 + 1, \ldots, N_i/2 - 1\},$$

where we assume for simplicity that N_1 and N_2 are even. This gives $\hat{\psi}_k((j+1)\Delta t)$ given $\hat{\psi}_k(j\Delta t)$:

$$\hat{\psi}_k((j+1)\Delta t) = e^{-\nu\alpha_k\Delta t}\hat{\psi}_k(j\Delta t) + \sqrt{\lambda}_k(X_k((j+1)\Delta t) - X_k(j\Delta t)).$$

The expressions $X_k((j+1)\Delta t) - X_k(j\Delta t)$ are independent (for different k and j) complex valued Gaussian random variables with variance given by (A.4) with $t = \Delta t$. We take $\hat{\psi}_k(0)$ from the stationary distribution, that is Gaussian with variance $\varepsilon_k/\|k\|^2$. Finally, we use the discrete Fourier transform [34] to obtain the values of ψ on an $N_1 \times N_2$ grid from its Fourier coefficients; see [32].

APPENDIX 3. PDE SOLVERS

In this section we describe the numerical method used to solve the PDE (21).

A.3.1. Discretization of the PDE

The discretization of the PDE (21) for u is based on the finite element method since the δ function term can easily be dealt with in the variational formulation. Denote the bilinear finite element subspace by V^h, and the set of nodal basis functions supported on the square elements by $\{\phi_j^h\}_{j=1}^N$. Let $u^h(x,t) = \sum_{j=1}^N \mu_j(t)\phi_j^h(x)$ and $\mu = (\mu_1, \ldots, \mu_N)^T$, $f(x,t) = \sum_{j=1}^\infty \beta_j(t)\phi_j^h(x)$ and $\beta = (\beta_1, \ldots, \beta_N)^T$, and $d_l = (\phi_1(x_l(t)), \ldots, \phi_N(x_l(t)))^T$.

We apply the finite element method and, for simplicity, lump the mass matrix and replace it by the identity; we also approximate the stiffness matrix by the five-point finite difference stencil for the periodic Laplacian, $\mathcal{A}^h$. We define the matrix $\mathcal{D}^h(x(t))$ by

$$\mathcal{D}_{ij}^h = \frac{1}{\Delta x^2} \sum_{l=1}^n \alpha \phi_i(x_l)\phi_j(x_l),$$

where $x = (x_1, \ldots, x_N)$. Thus, we have

$$\dot{\mu} + \nu\mathcal{A}^h\mu + \mathcal{D}^h(x)\mu = \beta + \frac{1}{\Delta x^2} \sum_{l=1}^n \alpha \dot{x}_l\, d_l.$$

The particle paths $(x, \dot{x})$ are integrated using the algorithm in Section 2.1.3.. Given this, a natural linearly implicit approximation for μ is

$$\frac{\mu^{k+1} - \mu^k}{\Delta t} + \nu\mathcal{A}^h\mu^{k+1} + \mathcal{D}^h(x^k)\mu^k = \beta^k + \frac{1}{\Delta x^2} \sum_{l=1}^n \alpha \dot{x}_l^k\, d_l. \tag{A.5}$$

Further implicitness can be introduced as follows:

$$\frac{\mu^{k+1} - \mu^k}{\Delta t} + \nu\mathcal{A}^h\mu^{k+1} + \mathcal{D}^h(x^k)\mu^{k+1} = \beta^k + \frac{1}{\Delta x^2} \sum_{l=1}^n \alpha \dot{x}_l^k\, d_l. \tag{A.6}$$

In (A.5) an instability occurs when many particles are close together and the effective force on the fluid is large at some points in the spatial domain. Treating the delta sources in a linearly implicit fashion, as in (A.6), cures the instability. Figure 28 demonstrates this instability for the coupled Taylor–Green problem.

A.3.2. Implementation

The solution of the coupled PDE-ODE system, (12) and (21), consists of three steps. At each time step:

1. Compute the right-hand side of (A.5) or (A.6).
2. Solve the linear system (A.5) or (A.6).
3. Solve the ODE (12).

Step 3 is described in Section 2.1. Step 1 is straightforward using the property of bilinear basis functions on square elements. All that is required is to distribute information located at x_l^k to the four nearest nodes.

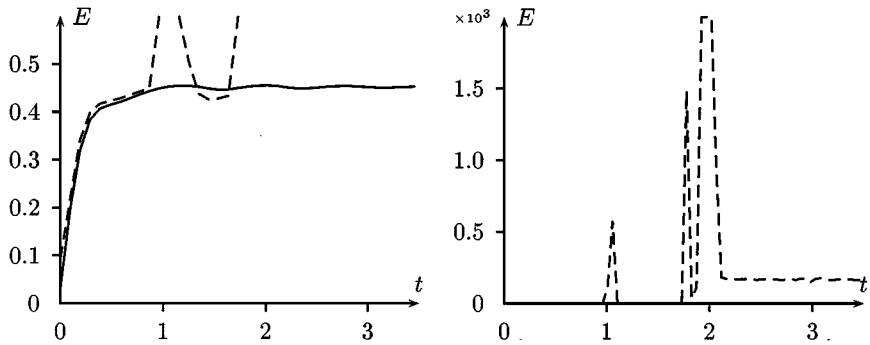

FIG. 28. Fluid enegry, $E(t) = \frac{1}{2}\int u(x,t)^2\,dx$, for coupled Taylor–Green flow with 10,814 particles. Solved using (A.6) (solid) and (A.5) (dashed). Left: Energy on a small scale for both methods (A.6)(solid) and (A.5)(dashed). The methods are initially comparable, but (A.5) blows up around time $t = 1$. Right: Energy on a larger scale for (A.5). The energy repeatedly blows up and decreases again.

In the explicit approach (A.5) of Step 2, the matrices on the left-hand side can be diagonalized by the discrete Fourier matrix and hence can be inverted efficiently using fast Fourier transform (FFT) [34], and the complexity is $\Theta(N\log(N))$. Moreover, we do not need to form the matrices explicitly.

In the semi-implicit approach (A.6), however, the matrix $\mathcal{D}^h(x^k)$ cannot be diagonalized by the Fourier matrix. Furthermore, the bandwidth of $\mathcal{A}^h$ is $\Theta(N)$ since periodic boundary conditions are used, and hence (banded) Gaussian elimination can be very expensive. However, note that the rank of $\mathcal{D}^h(x^k)$ is equal to n, the number of particles. If we use conjugate gradient (CG) with

$$\mathcal{L}^h \equiv \mathcal{I}^h + \nu\Delta t\mathcal{A}^h$$

as preconditioner, then CG will take at most $n+1$ iterations to converge. In practice, only a few iterations are required to converge (see Fig. 24). The inversion of the preconditioner $\mathcal{L}^h$ can be done by use of the FFT, as in the explicit case.

ACKNOWLEDGMENT

We thank Paul Tupper for reading early versions of the manuscript and providing numerous useful suggestions. We also thank an anonymous referee for helpful suggestions concerning our complexity analysis.

REFERENCES

1. B. J. Alder and T. E. Wainwright, *J. Chem. Phys.* **31**, 459 (1959).
2. J. J. Erpenbeck and W. W. Wood, Molecular dynamics techniques for hard-core systems, in *Statistical Mechanics B*, edited by B. J. Berne, volume 6 of *Modern Thoretical Chemistry*, (Plenum Press, New York, 1977), Ch. 1, pp. 1–40.
3. D. C. Rapaport, *J. Comput. Phys.* **34**, 184 (1980).
4. M. Marín, D. Risso, and P. Cordero, *J. Comput. Phys.* **109**, 306 (1993).
5. S. Luding, E. Cleément, A. Blumen, J. Rajchenbach, and J. Duran, *Phys. Rev. E* **49**, 1634 (1994).
6. S. Luding, H. J. Herrmann, and A. Blumen, *Phys. Rev. E* **50**, 3100 (1994).
7. S. Sundaram and L. R. Collins, *J. Comput. Phys.* **124**, 337 (1996).

8. S. Sundaram and L. R. Collins, *J. Fluid Mech.* **335**, 75 (1997).
9. W. C. Reade and L. R. Collins, *J. Fluid Mech.* **415**, 45 (2000).
10. D.-J. Kim, L. J. Guibas, and S.-Y. Shin, Fast collision detection among multiple moving spheres, in *Proceedings of the Thirteenth Annual Symposium on Computational Geometry, Nice, France, 1997*, ACM, Vol. 13, pp. 373–375.
11. D.-J. Kim, L. J. Guibas, and S.-Y. Shin, *IEEE Trans. Visualization Comput. Graph.* **4**, 230 (1998).
12. P. Hontalas, B. Beckman, M. DiLoreto, L. Blume, P. Reiher, K. Sturdevant, L. V. Warren, J. Wedel, F. Wieland, and D. Jefferson, Performance of the Colliding Pucks simulation on the Time Warp operating systems (Part 1: Asynchronous behavior & sectoring), in *Distributed Simulation*, edited by B. Unger and R. Fujimoto, Simulation Series (SCS, 1989), Vol. 21, pp. 3–7.
13. B. D. Lubachevsky, *J. Comput. Phys.* **94**, 255 (1991).
14. B. D. Lubachevsky, Simulating colliding rigid disks in parallel using bounded lag without Time Warp, in *Distributed Simulation*, edited by D. Nicol, Simulation Series (SCS, 1990), Vol. 22, pp. 194–202.
15. H. M. Schaink, P. A. Nommensen, R. J. J. Jongschaap, and J. Mellema, *J. Chem. Phys.* **113**, 2484 (2000).
16. L. E. Silbert and J. R. Melrose, *J. Rheol.* **43**, 673 (1999).
17. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms* (The MIT Press, Cambridge, MA, 1990).
18. M. Dyer, Personal communication, 2000.
19. L. Boltzmann, *Vorlesungen über Gastheorie* (Ambrosius Barth, Leipzig, 1912).
20. G. E. Uhlenbeck and G. W. Ford, *Lectures in Statistical Mechanics*, Lectures in Applied Mathematics (Am. Math. Soc. Providence, 1963), Vol. I.
21. F. Reif, *Fundamentals of Statistical and Thermal Physics*, McGraw-Hill Series in Fundamentals of Physics, (McGraw-Hill, New York, 1965).
22. D. C. Rapaport, *J. Comput. Phys.* **105**, 367 (1993).
23. B. D. Lubachevsky, *J. Comput. Phys.* **105**, 369 (1993).
24. G. K. Batchelor, *An Introduction to Fluid Dynamics* (Cambridge Univ. Press, Cambridge, UK, 1967).
25. A. A. Amsden, P. J. O'Rourke, and T. D. Butler, *KIVA-II: A Computer Program for Chemically Reactive Flows with Sprays* (Los Alamos National Laboratory, Los Alamos, NM, 1989).
26. Y. A. Houndonougbo, B. B. Laird, and B. J. Leimkuhler, *Mol. Phys.* **98**, 309 (1999).
27. L. Verlet, *Phys. Rev.* **159**, 98 (1967).
28. G. I. Taylor, The decay of eddies in a fluid, in *Scientific Papers of G. I. Taylor*, Cambridge Univ. Press, Cambridge, UK. 1960), Vol. 2, pp. 190–192.
29. G. I. Taylor and M. S. Green, *Proceedings of the Royal Society* **158**, 499 (1937).
30. J. García-Ojalvo and J. M. Sancho, *Noise in Spatially Extended Systems* (Springer-Verlag, New York, 1999).
31. B. Maury and R. Glowinski, Fluid-particle flow: a symmetric formulation, in *C.R. Acad. Sci. Paris, Number* **324**, 1079 (1997).
32. H. Sigurgeirsson and A. Stuart, *Particles in synthetic turbulence: A random dynamical system*, in preparation.
33. G. Da Prato and J. Zabczyk. Stochastic equations in infinite dimensions, in *Encyclopedia of Mathematics and its Applications* (Cambridge Univ. Press, Cambridge, UK, 1992), Vol. 44.
34. M. Frigo and S. G. Johnson, The fastest Fourier transform in the west, available at http://www.fftw.org/.