Computer Science at Kent

Detecting collisions in sets of moving particles: a survey and some experiments

Colin G. Johnson and Jacqueline L. Whalley

Technical Report No: 8-02
Date: June 2002

Published by the Computing Laboratory,
University of Kent, Canterbury, Kent CT2 7NF, UK.

Detecting collisions in sets of moving particles: a survey and some experiments

Colin G. Johnson and Jacqueline L. Whalley
Computing Laboratory
University of Kent at Canterbury

June 26, 2002

Abstract

Detecting and responding to collisions between particles is an important requirement for building simulations in computational science. Due to the large number of potential collisions it is impractical to check all possibilities, so the development of algorithms which narrow down the number of possible searches to a small number is important. In this paper we review various algorithms for this task, and give results from a number of experiments which demonstrate the relative efficiency of these algorithms on a fundamental problem of detecting collisions between particles undergoing Brownian motion. The general slant of the paper is towards the development of algorithms for simulating microbiological systems.

1 Introduction

Many simulations in computational physics, chemistry and biology consist of the analysis of large numbers of moving particles. In some applications it is sufficient to consider the global movement of these particles by approximating their motion as vector fields in a continuous space. However in other simulations it is important to consider the individual particles and their interactions, for example where there are large numbers of particle types, where the spatial structures formed by the particles are significant, or where the stochastic nature of the dynamics can give rise to different results. An example of the latter is given by Arkin et al. [MA97], who use stochastic simulations to demonstrate that stochastic fluctuations in reactions in a biochemical pathway give rise to two distinct macro-level behaviours known as *lysogeny* and *lysis*.

The most significant algorithmic challenge in such simulations is detecting collisions between particles, or between particles and structures in their environment. A number of authors have discussed these algorithms, however the literature on this topic is diffuse and there is often little awareness of the work of earlier authors. The aim of this paper is to bring together some of these ideas, add some new ones and formalize some "folklore", and demonstrate the application of these ideas through a number of experiments.

2 Collision detection algorithms

Collision detection algorithms fall into two main categories. The first category consists of algorithms for the detection of collisions between objects which are "large", in the sense that they occupy a significant area or volume in the ambient space, they potentially have complex geometric structure, and we typically want to know some details of the collision/penetration, e.g. knowing which parts of the geometric structure are touching at collision time. Such algorithms are commonly used in computer-aided design [LG98], virtual reality [HLC+97] and robotics [HA92, HAFG95, Cam93, JM99]. A number of efficient systems are available for carrying out this type of collision detection, of which the I-COLLIDE program is perhaps the best known [CLMP95].

This sort of collision detection algorithm is not readily applicable to the second sort of collision detection problem, where the particles are "small" in the sense that the vast majority of the ambient space is empty and the particles typically have little complex structure, usually being represented by hard spheres.

Again there are a number of different requirements for these algorithms, depending on the application in question. Fluid dynamics applications where there are large numbers of densely packed particles [AT87] may require a different approach to problems in which particles are moving in a largely empty space and collisions are rare. The forces acting on and between the particles provide another kind of difference. Some simulations (e.g. simulations of the long-term development of galaxies under gravity) require the calculation of many long-distance interactions between different particles. Some applications can ignore these larger-scale interactions and focus solely on local interactions. The limiting case of this describes a particularly common situation where the only interactions are between particles which are in direct physical contact. Another variant in the vein is where there are forces acting on the particles by entities other than the other particles in the system; for example external gravitational or magnetic fields [SSW01].

In this paper we shall concentrate on a particular type of problem of particular relevance to biological and biochemical simulation. In this type of problem a set P of particles are moving in a volume or area, and are not allowed to pass through each other. Their motion when not colliding with other particles in P is given by brownian motion; this simulates the collision of the particles in P with other particles in the environment which are not explicitly simulated. The accuracy of the brownian motion simulation was confirmed in the experiments below using the methods described by Berg [Ber93]. The purpose of the collision detection is twofold. Firstly collision detection will be used to prevent two particles occupying the same space at the same time. Secondly the program will report a list of collisions which have occurred; this can then be used to decide whether particles associate to form other particle types, to form particle clusters, et cetera.

Many of the basic ideas which can be used to study this type of problem are also found in the study of "billiard ball" type problems [Hay96]. Billiard ball problems consist of calculating the collisions between particles which are moving under Newtonian dynamics in a space, with no changes in direction other than those directly caused by the collision with other particles. Some algorithms which have been applied to billiard ball problems can also be applied to the situation where there is a force acting on the particles depending on their location in the space.

The earliest work on this type of calculation is that of Alder and Wainwright from 1959 [AW59]. They construct a basic individual-based simulation and introduce the idea of making the timestep-length equal to the distance between collisions. They also briefly mention the idea of dividing the region into cells, but take this idea not further.

The main problem in scaling up collision detection to larger numbers of particles is dealing with the large numbers of potential collisions. Each particle can potentially collide with any other particle leading to a quadratic growth in the number of potential collisions. However if there is an upper bound on the speed of the particles, then there is a way to divide the problem down into smaller sub-problems.

The core of this solution is to break down the space containing the problem into a grid of *cells*. Assume for the purposes of this paper that the cells are squares/cubes, the corners of which form a regular lattice in the space. Instead of the particles having a position in the space relative to a single origin, they instead have a position relative to the corner (or centre, or some consistently-defined frame) of the cell in which their centre is located. When a particle moves a check is made to see whether its centre passes through any of the boundaries of the current cell, and if so the particle is transferred to the appropriate location in the other cell.

The advantage of this is that it restricts the number of collision checks which need be made. Providing the speed is bound above so that within a timestep the furthest a particle can move in one timestep is half-way across a cell minus the diameter of the particle (this statement is made more precise below), then the only particles with which it can collide are those either in the current cell or in neighbouring cells. This vastly reduces the number of potential collisions which can be followed and means that as the system is scaled, the number of potential collisions increases roughly linearly with the number of molecules, provided that the density of molecules is kept constant and the cell-size is well chosen. Early experiments on this were carried out by Rapoport [Rap80] and Erpenbeck and Wood [EW81].

One interesting perspective on this is to compare collision detection of particles in continuous and discrete spaces. Take a discrete space such as a square grid where the particles occupy intersection points. Collision detection (interpreted as detecting particles occupying neighbouring points) is easy in such a system, because of there is a small, finite number of neighbouring points which need be looked at. The decomposition of the continuous space into cells achieves a similar effect; now every point belongs to a cell, which in turn has a number of neighbours. Provided certain conditions are met looking at the neighbours is sufficient.

Another way of increasing the efficiency of these algorithms is by the use of an *event queue* [Rap80, SSW01]. In such a system we compute all the collisions which will occur in a time step, sort them in order of occurrence, and then iterate through the queue calculating the effects of each collision in turn. For the standard billiard ball problem such a queue is easy to construct; the motion of the particles can be considered to be a set of simultaneous equations of motion, with the simultaneous solutions representing points in time when the particles occupy the same point in space. Such event queues are less significant in the brownian motion model explored in this paper, as the time-horizon from the current time-point to the point at which the brownian movements will take place is short compared with the time to next collision along the current trajectory, so in most cases the particle will have made a brownian motion before it reaches the predicted event.

3 Development of a cell-based algorithm

In the above survey two main ways of representing the position of particles within the space were presented. The first of these was as a coordinate vector relative to the whole space in which the particles are moving. This makes the writing of algorithms to move the particles simple but means that there are too many collision possibilities to be checked. This algorithm in which all pairs are checked individually shall be used as a baseline for algorithm performance; call this the **naive algorithm**. The second was to represent the points relative to a cell which they occupy. This gives efficient collision detection but means that every function in which a particle is moved needs to take into account the possibility that the particle may move into a different region [KGS97]. This section discusses the development of an algorithm which combines these two representations using some features of object-oriented programming languages (a more general discussion of OO methods to similar problems is given by Norton et al. [NSD95]).

3.1 A dual representation

Firstly assume that there is a class (in the OO sense of the word), objects of which represent the particles in the system (the `particle` class). The initial stage is to divide the space in which the

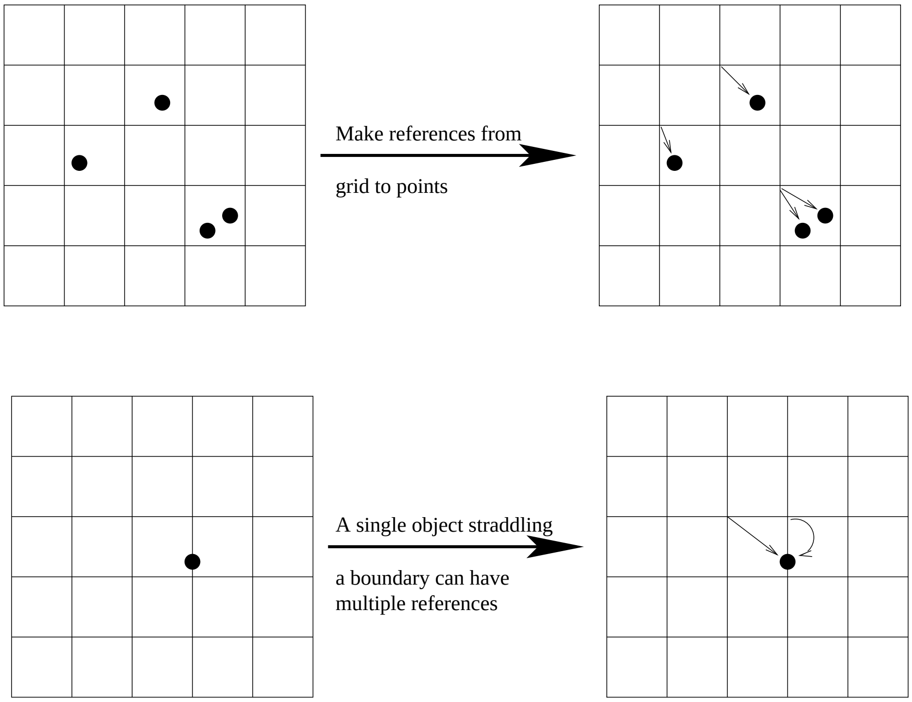

Figure 1: Clamping the objects to the corners of the grids which represent them.

particles move into a grid of squares (for a 2D problem) or cubes (for a 3D problem). We shall refer to these squares or cubes as *cells*. Associated with each of these cells is a list (e.g. a variable-length array type such as a Java `ArrayList`) of references (C-style pointers) to objects which are of the `particle` class. This shall be referred to as the `particleList` for that particular cell. If there are multiple particle types then the `particle` class can be a Java-style interface or an abstract class, implemented by each of the particle types.

Create a list containing all of the particles in the space. This shall be referred to as the `globalList`. To initialize the simulation iterate through this `globalList`, and calculate which cell(s) any part of that particle belongs to. For each cell that it belongs to add a pointer to `particleList` belonging to that cell. This process is illustrated in figures 1 and 2. Then make a list of references (C-style pointers again, or just coordinates) *back* from the object to the grid points which contain them (it is assumed that one of the attributes of the `particle` class is an appropriate list of pointers—call this the `cellList`).

This two-way linking is a powerful technique in object-oriented programming, allowing the creation of simple structures which get away from an oversimple hierarchical view where each piece of data contains other data, and not *vice versa*.

It is easy to encapsulate the updating of references between cell and particle each time a particle moves in a single method within the `particle` class. The first step would be to check whether any part of the particle has moved into any other cell. If any part of it has moved into another cell then it would iterate through its `cellList`, sending a message to each of the cells asking that the particle in question be removed from the `particleList` of that cell. Finally it checks which cells are now occupied by any part of the particle and adds a two-way link as described above.

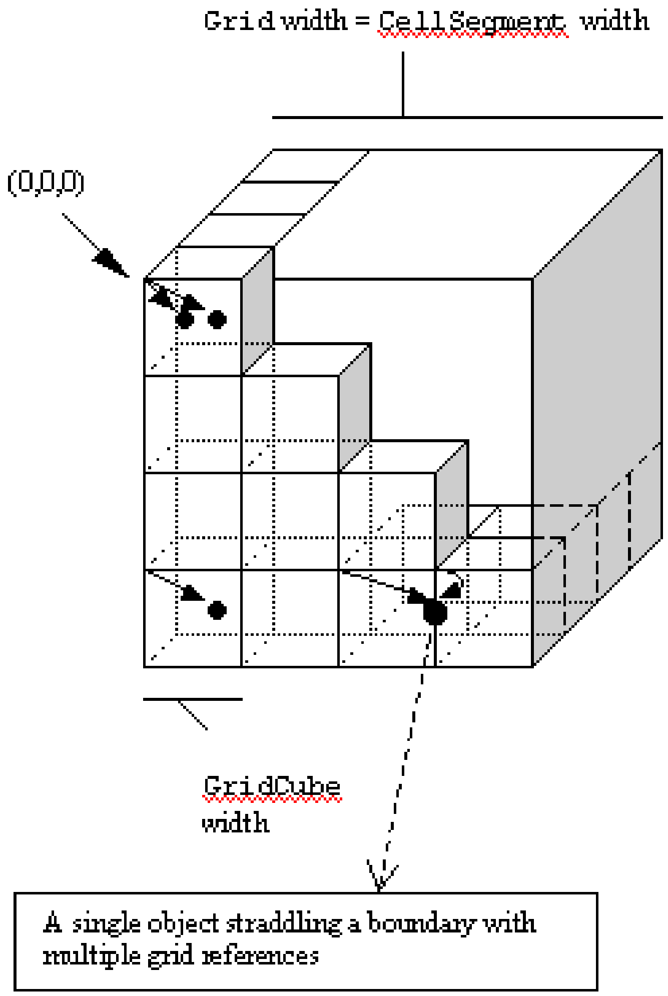

Figure 2: Subdividing the three-dimensional volume.

3.2 Applying the representation to collision detection

Once the above system has been set up, the algorithm for finding collisions is fairly simple. The algorithm iterates through the list of cells, and for each particle in its `cellList` checks whether that particle collides with any particles either in the cell itself or in one of the neighbouring cells during the current timestep. In order to ensure that this is sufficient, a relationship between the size of the particles, the size of the cells, and the maximum speed of the particles needs to hold. Let c be the minimum distance from the centre of a cell to its boundary, p_1 and p_2 the diameters of the two particles and d the distance a particle travels when it is travelling at maximum speed. Provided $c - (p_1 + p_2) < d$, all collisions will be found. It would seem to be a bad idea to increase the grid size beyond this, as this will increase the number of individual checks which need to be made. Some experimental evidence to support this is given in [GGBJ02].

3.3 Details of the collision detection

Once it has been determined that two particles are in a region where a collision could happen we need to check exactly whether they collide and if so, at what time the contact is made. This is a fairly simple exercise in algebra. Firstly let vectors $\mathbf{A}_s$ and $\mathbf{A}_e$ represent the start and end points of the centre of particle A, and $\mathbf{B}_s$ and $\mathbf{B}_e$ the same for particle B.

$$\mathbf{A}_s = \begin{pmatrix} x_{A_s} \\ y_{A_s} \\ z_{A_s} \end{pmatrix} \quad \mathbf{A}_e = \begin{pmatrix} x_{A_e} \\ y_{A_e} \\ z_{A_e} \end{pmatrix} \quad \mathbf{B}_s = \begin{pmatrix} x_{B_s} \\ y_{B_s} \\ z_{B_s} \end{pmatrix} \quad \mathbf{B}_e = \begin{pmatrix} x_{B_e} \\ y_{B_e} \\ z_{B_e} \end{pmatrix} \tag{1}$$

So the trajectory of the centre points in the time interval $[0, 1]$ is given by

$$\mathbf{P}_A(t) = \mathbf{A}_s + (\mathbf{A}_e - \mathbf{A}_s) \tag{2}$$
$$\mathbf{P}_B(t) = \mathbf{B}_s + (\mathbf{B}_e - \mathbf{B}_s) \tag{3}$$

where the spheres are at their starting position at $t = 0$, at their end position at $t = 1$, and move with constant velocity between these two points. The vector joining these two points is

$$\mathbf{P}_B(t) - \mathbf{P}_A(t) \tag{4}$$

so writing this out more fully gives

$$\mathbf{B}_s + t(\mathbf{B}_e - \mathbf{B}_s) - \mathbf{A}_s - t(\mathbf{A}_e - \mathbf{A}_s) \tag{5}$$
$$= \mathbf{B}_s - \mathbf{A}_s + t(\mathbf{B}_e - \mathbf{B}_s - \mathbf{A}_e + \mathbf{A}_s) \tag{6}$$
$$= \begin{pmatrix} x_{B_s} - x_{A_s} \\ y_{B_s} - y_{A_s} \\ z_{B_s} - z_{A_s} \end{pmatrix} + t \begin{pmatrix} x_{B_e} - x_{B_s} - x_{A_e} + x_{A_s} \\ y_{B_e} - y_{B_s} - y_{A_e} + y_{A_s} \\ z_{B_e} - z_{B_s} - z_{A_e} + z_{A_s} \end{pmatrix} \tag{7}$$

Abbreviate this by letting

$$X_1 = x_{B_s} - x_{A_s} \tag{8}$$
$$X_2 = x_{B_e} - x_{B_s} - x_{A_e} + x_{A_s} \tag{9}$$
$$\ldots \tag{10}$$

now we can rewrite the above expression as

$$\begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} = t \begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} \tag{11}$$

The length of the vector joining the two centres is therefore

$$\ell = \sqrt{(X_1 + tX_2)^2 + (Y_1 + tY_2)^2 + (Z_1 + tZ_2)^2} \tag{12}$$

Now let R represent the sum of the radii of the two circles. When the two circles are touching the distance between the centres of the two circles will equal R, i.e.

$$R = \sqrt{(X_1 + tX_2)^2 + (Y_1 + tY_2)^2 + (Z_1 + tZ_2)^2} \tag{13}$$

so expanding this expression gives

$$R^2 = X_1^2 + 2tX_1X_2 + t^2X_2^2 + Y_1^2 + 2tY_1Y_2 + t^2Y_2^2 + Z_1^2 + 2tZ_1Z_2 + t^2Z_2^2 \tag{14}$$

now we can rearrange this to give a quadratic equation in terms of t

$$t^2(X_2^2 + Y_2^2 + Z_2^2) + t(2(X_1X_2 + Y_1Y_2 + Z_1Z_2)) + (X_1^2 + Y_1^2 + Z_1^2 - R^2) = 0 \tag{15}$$

so we can solve this by standard methods; let

$$\begin{aligned} A &= X_2^2 + Y_2^2 + Z_2^2 && (16) \\ B &= 2(X_1X_2 + Y_1Y_2 + Z_1Z_2) && (17) \\ C &= X_1^2 + Y_1^2 + Z_1^2 - R^2 && (18) \\ & && (19) \end{aligned}$$

and solve using the quadratic formula.

The results can be interpreted as follows. If the values for t have an imaginary component or are real but both lie outside the interval $[0, 1]$, then no collision occurs during the current timestep. If there are two real values then the lowest of these represents the time of first contact between the two spheres. The higher of the two values can be ignored; that represents the time at which, had the spheres been allowed to pass through each other, they would last touch.

Here is a simple worked example. Consider two spheres each of radius 0.2. Sphere A moves from $(0, 0, 0)$ to $(1, 1, 0)$, whilst sphere B moves from $(1, 0, 0)$ to $(0, 1, 0)$. Solving equation 15 using Maple [HHR96] gives the solution set $\{t = 0.7, t = 0.3\}$. Plots showing (a projection of) the spheres at these two points are illustrated in figure 3.

4 Improvements to the algorithm

The previous section of the paper introduced the basic idea of an algorithm which uses a dual representation of particles; one representation which represents the particle's global position in the space, and one which represents its position relative to a division of that space into cells. This section introduces a number of ideas which demonstrate improvements on that algorithm.

4.1 Fine-grained timesteps

One problem with the above algorithm is that it allows a certain collision type to go undetected. This is illustrated in figure 4. In this type of collision one collision happens at a certain point in the timestep, however as a consequence of this a second collision occurs within the same timestep which is not detected. The chance of this occurring can be reduced by reducing the length timestep, however this rapidly becomes intractable as the timestep gets shorter.

A solution to this problem is to use variable-length timesteps. In the context of this problem a good way to think about this is to consider subdivisions of a main timestep, as that allows the

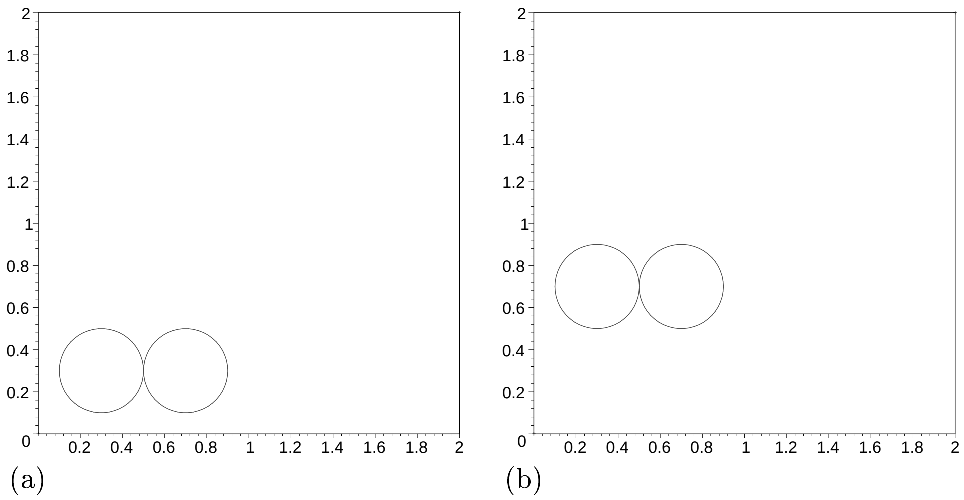

Figure 3: The spheres at their first touch (a) and theoretical last touch (b)

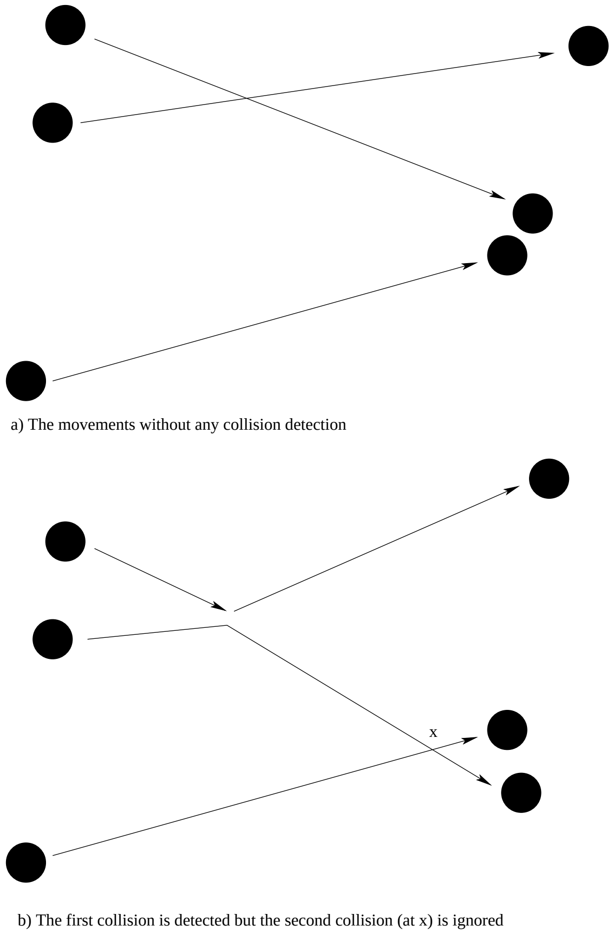

Figure 4: Two collisions in a single timestep.

recording of the current state of the system at the end of each timestep. To calculate the desired length of the timestep the algorithm above is used to calculate a list of collisions which will happen if the particles continue on their current trajectories. The earliest collision in the list is then taken, and the time when that collision occurs is used as the end of the current subdivision. However if the earliest collision occurs after the end of the current timestep, then the end of the current subdivision ends at the end of the timestep. Once this calculation has been carried out the particles are advanced to the position at the end of the subdivision and the effect of the collision on the colliding pair calculated (this is problem dependent). The next collision is then calculated from the start of the current position; this is repeated until the end of the timestep is reached.

There are a number of ways in which this procedure could potentially be refined (these ideas are not implemented in the experiments below). Firstly if collisions are occurring in parts of the space which are sufficiently far away from each other that no change in direction/speed caused by the first collision could interfere with the second (et cetera) collision, then there would be no need to recalculate the next collision at the beginning of each subdivision. Instead a miniature event queue could be created, with all the events which happen at sufficient distances from each other being queued in order of occurrence and being calculated, with only the particles whose direction has been changed by collision being checked against other particles. It is not clear whether the overhead in terms of the data structures required to keep track of which particles have changed direction by which subdivision is greater than simply recalculating all the potential collisions, particularly when the particle density is low.

A second interesting idea would be to do a conservative approximation of whether there is enough time for a collision to occur. If the event queue described in the previous paragraph is not being implemented then all that needs to be calculated is the time of the next collision. Taking the potential collisions in arbitrary order, we carry out a collision check between pairs until a collision is found. The time to this collision gives an upper bound on the time of the next collision. Now that this upper bound has been found, the remainder of the potential collisions can be checked using a two-stage process. Firstly the distance between the two particles is calculated, and then the distance which they can travel in the time given by the current upper bound is calculated. If the sum of these two distances plus the sum of the radii of the two particles is less than the distance between them, no collision can occur, so the algorithm moves onto the next particle-pair. If there is a potential collision after this approximate calculation has been carried out, then the complete collision check is done and if (1) there is a collision and (2) the collision time is lower than the current upper bound then that collision is recorded as the current earliest collision and the current upper bound is reduced to the current collision time. Once all potential collisions have been checked in this fashion, the earliest collision is known.

4.2 Reducing the number of cells checked

Another improvement which can be made is to reduce the number of neighbours which need to be checked by taking into account the direction of movement of the cell. Provided that the speed of motion within the cell is bounded as described above, then we can reduce the number of cells checked in the following way. We shall illustrate this for three dimensions, the generalization to arbitrary dimension should be clear. Define 6 regions called *halves* within the cell; two each in the x, y, and z directions. A half is all points which are to one side of the plane dividing the cell in two along its middle in the appropriate direction. If a particle is has some or all of its points in one of the two halves in a particular direction, and none of its points in the other half in the same direction, and the particle is moving "away from" the a plane normal to the axis of the half in question in the sense of moving away from the dividing plane between the two halves, then the

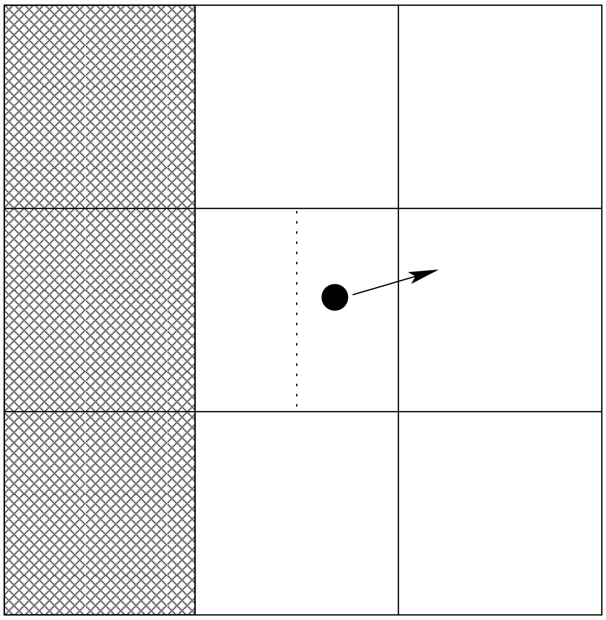

Figure 5: When the particle is sufficiently far into one half of the cell and moving away from the dividing line, some cells can be ignored.

three cubes which are on the "other" side of the half need not be examined.

This is because the only places where the particle can be is in the starting half of the original cell, in the matching half on the sides of the cell which are adjacent to the cell in the directions where the plane divides the cell in two, and the cells which are "in front of" the original half. The other particle cannot come from the cells which are "behind" the original cell, as the furthest they can get is into the one of the "other" halves of the original cell or the "other" halves of the adjacent cells.

This effect is illustrated in figure 5. It should be noted that this effect can hold in more than one direction for a single motion.

The algorithm incorporating all of the developments so far shall be called *Algorithm A*. A comparison of various versions of the algorithm is given in figure 6.

4.3 Reducing the frequency of pointer-updates

An experimental timing analysis of the above program revealed that much of the time was spent recalculating the relationships between cells and particles. Therefore it is desirable to show that the number of updates required can be reduced. In fact we can reduce it so that the cell-particle relationships need only be calculated at the beginning of each timestep, provided a certain relation between speed and cell-size holds.

Result 1 *Let d be the minimum distance from the centre of a cell to its boundary (in a square cell this is the length of a normal from the cell-edge to the centre). Let r be the maximum radius of a particle in the system. Provided that the speed of particles is bounded above so that the maximum distance a particle can travel in one timestep is $d-r$, then it is sufficient to update the links between cells and particles every timestep.*

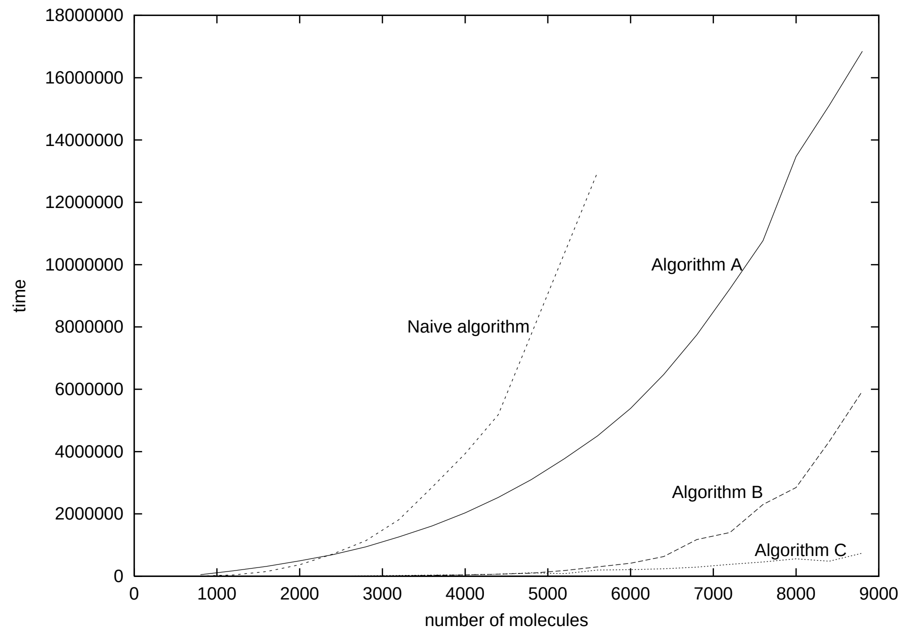

Figure 6: Comparison of various versions of the algorithm. Time is number of milliseconds of wall-clock time to execute the algorithm for a whole timestep.

Proof Consider a particle p_1 in cell c_1 at time t which is the beginning of a timestep of length 1. By time $t+1$ the particle will either still be in c_1 or in one of the immediate neighbouring cells.

If a collision happens in $[t, t+1]$ then the centre of the particle p_2 with which it collides can (trivially) be in one of three places at time t:

1. In the cell c_1
2. In one of the immediate neighbours of the cell c_1. Call the union of these cells $N(c_1)$.
3. Somewhere else.

Let us consider these cases in turn.

In case (1) both particles are at all times either in c_1 or its immediate neighbours $N(c_1)$. This is because of the speed restriction; regardless of where in c_1 the particles begin, neither of them can get out of $c_1 \cup N(c_1)$ within the timestep. Given that the (relevant) neighbours are checked by the algorithm above, all collisions will be found.

In case (2) p_1 is always in $c_1 \cup N(c_1)$ by the same argument as before. During the timestep p_2 can be in $c_1 \cup N(c_1)$, in which case any collision would be detected, or it can move outside this region, in which case there is no possibility of collision with p_1.

In case (3) no collisions can occur. p_1 starts in s_1, and the furthest that it can go (by the speed restrictions) is to the point where the point on its surface furthest from the centre of c_1 is at most half of the minimum distance across the cubes into one of the neighbouring cubes. Similarly the analogous point on particle p_2 can only get half way *into* that cube from the other side. This is illustrated in figure 7 Therefore no collision can occur in this case. ∎

This improvement has been implemented. **Algorithm B** is the same as **Algorithm A**, but with the update carried out every timestep rather than every move. Again, comparative results are given in figure 6.

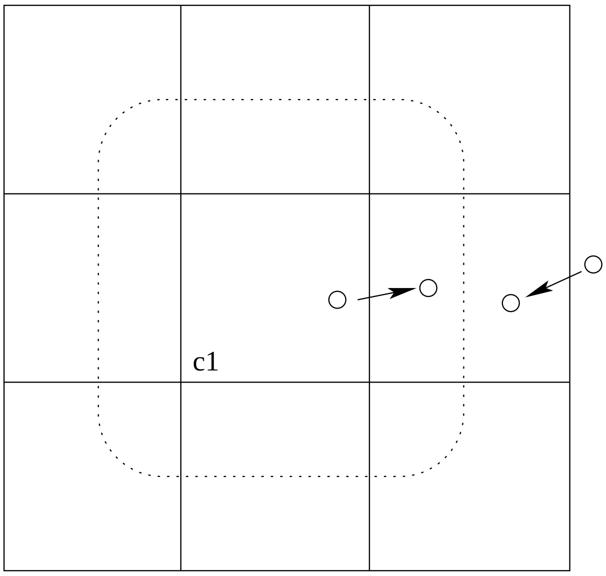

Figure 7: The particle coming from outside $N(c_1)$ cannot touch the particle coming from inside c_1 within the timestep.

4.4 Detecting collisions within slice-shaped cells

In order to improve the time-behaviour of the algorithm further a timing analysis was carried out, measuring the amount of time the algorithm spent in various blocks of code. This identified one significant bottleneck in the algorithm, at the point where the algorithm iterates round the various neighbouring cells looking for collisions. Is there a way to reduce the number of neighbouring cells which need to be examined? One way which has proven particularly successful is as follows. Instead of subdividing the volume into square cells by dividing in three directions, divide the volume into slices by dividing only in one direction (figure 8). This still divides the volume down into smaller volumes, whilst reducing the number of neighbours to two (or, when combined with the methods in section 4.2 above, one in many cases).

The algorithm with slice-shaped cells replacing cubical cells will be called called **Algorithm C**. Comparative results are given in figure 6. This is a vast improvement. It would appear that the number of particles in each cell is still sufficiently small to reduce the number of potential collisions; however the reduction in the number of neighbourhoods which need checking and the number of `cellList` updates seems to be significant. In a very densely packed volume the relative weight of these two factors is likely to be reversed.

5 Further ideas

There are a number of additional ways in which the above system could be extended. In some simulations a large number of small particles interact with a small number of large ones. This was the case in some of our recent work [WTJ02], which simulated the formation of prion aggregates in a cell; after a while a number of large clusters form. This sort of heterogeneity is not uncommon in biological modelling. To anticipate this the cell size needed to be made rather larger than was needed by the vast majority of the particles. One solution to this is to remove excessively large particles from the algorithm and to treat them differently, e.g. by applying the naive algorithm

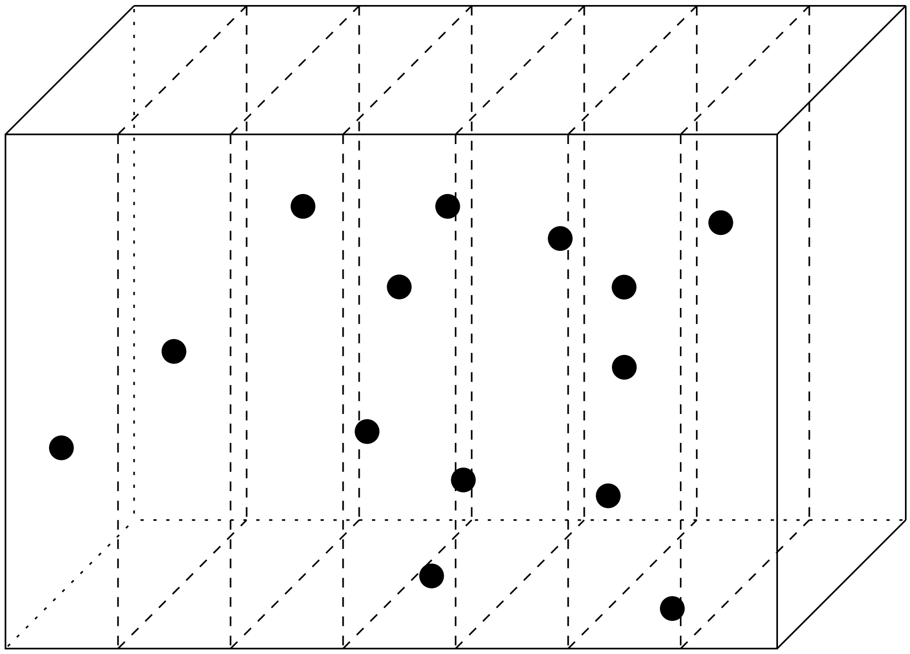

Figure 8: Slicing the volume in a single direction rather than into cubes.

to the small number of potential collisions between the large particles and all particles. A similar argument can be used to deal with fast particles which are travelling faster than the maximum speed allowed by the collision-detection algorithm.

Dealing with a more uniform set of objects of different sizes may be more difficult. One possibility would be to have multiple lattices of cells of different sizes, or to change the size of cells as the algorithm progresses.

6 Conclusions

We have surveyed algorithms for collision detection in particle fields. Based on this survey we have presented an algorithm which combines the best aspects of two approaches to this problem, and shown how this algorithm can be improved. Some applications of the algorithms in the simulation of cellular systems from biology can be found in [GGBJ02, WTJ02, Gol00, Wha01].

Thanks to Jacki Goldman and Dennis Bray for suggestions and discussion.

References

[AT87] M.P. Allen and D.J. Tildesley. *Computer Simulation of Liquids.* Clarendon Press, 1987.

[AW59] R.J. Alder and T.E. Wainwright. Studies in molecular dynamics. i. general method. *Journal of Chemical Physics*, 13(2):459–466, 1959.

[Ber93] Howard C. Berg. *Random Walks in Biology.* Princeton University Press, 1993. Expanded second edition.

[Cam93] Stephen Cameron. Using space-time for collision detection : solving the general case. In Kevin Warwick, editor, *Robotics, Applied Mathematics and Computational Aspects*, pages 403–415. Clarendon/IMA, 1993.

[CLMP95] J. Cohen, M. Lin, D. Manocha, and K. Ponamgi. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*, pages 189–195, 1995.

[EW81] J.J. Erpenbeck and W.W. Wood. Molecular dynamics calculations of shear viscosity time correlation functions for hard spheres. *Journal of Statistical Physics*, 24:455–468, 1981.

[GGBJ02] Jacki P. Goldman, William J. Gullick, Dennis Bray, and Colin G. Johnson. Individual-based simulation of the clustering behaviour of epidermal growth factor receptors. In Gary Lamont, editor, *2002 ACM Symposium on Applied Computing*. ACM Press, 2002.

[Gol00] Jacki P. Goldman. An object-oriented model of growth factor receptor clustering. Master's thesis, University of Kent, 2000. Further details at `http://www.cs.ukc.ac.uk/people/staff/cgj/research/receptors.html`.

[HA92] Y.K Hwang and N. Ahuja. Gross motion planning—a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[HAFG95] Vincent Hayward, Stephane Aubry, André Foisy, and Yasmine Ghallab. Efficient collision prediction among many moving objects. *International Journal of Robotics Research*, 14(2):129–143, April 1995.

[Hay96] Brian Hayes. The way the ball bounces. *American Scientist*, pages 331–335, July-August 1996.

[HHR96] K.M. Heal, M.L. Hansen, and K.M. Rickard. *Maple V: Learning Guide*. Springer, 1996.

[HLC+97] T. Hudson, M. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-COLLIDE: Accelerated collision detection for VRML. In *Proceedings of VRML97*, 1997.

[JM99] Colin G. Johnson and Duncan Marsh. Modelling robot manipulators using multivariate B-splines. *Robotica*, 17(3):239–247, May 1999.

[KGS97] Dong Jin Kim, Leonidas J. Guibas, and Sung Yong Shin. Fast collision detection among multiple moving spheres. In *Proceedings of the 1997 ACM Symposium on Computational Geometry (Nice, France)*, pages 373–375. ACM Press, 1997.

[LG98] M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proceedings of the 1998 IMA Conference on the Mathmatics of Surfaces*. Oxford University Press, 1998.

[MA97] H.H. McAdams and A. Arkin. Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences of the United States of America*, 94(3):814–819, 1997.

[NSD95] Charles D. Norton, Boleslaw K. Szymanski, and Viktor K. Deyck. Object-oriented parallel computation for plasma simulation. *Communications of the ACM*, 38(10):88–100, 1995.

[Rap80] D.C. Rapaport. The event scheduling problem in molecular dynamics. *Journal of Computational Physics*, 34:184–201, 1980.

[SSW01] Hersir Sigurgeirsson, Andrew M. Stuart, and Wing-Lok Wan. Collision detection for particles in a flow. *Journal of Computational Physics*, 172:766–807, 2001.

[Wha01] Jacqueline L. Whalley. Object oriented computational models of prion propagation. Master's thesis, University of Kent, 2001.

[WTJ02] Jacqueline L. Whalley, Mick F. Tuite, and Colin G. Johnson. A virtual lab for exploring the $[psi]^+$ yeast prion. In Faramarz Valafar, editor, *Proceedings of the 2002 International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*. CSREA Press, 2002.