Simulation of a many-particle system using space partitioning

Roald Frederickx
Kasper Meerts

16 october 2010

1 Scientific model

1.1 Collisions

The most basic short-range interaction that can be modeled in a many particle system is the elastic collision between hard spheres or disks. It is important that this interaction is not only addressed correctly, but also efficiently. Therefore, we want to avoid calculating computationally expensive trigonometric functions like sine and cosine and try to keep square roots to a minimum.

Collision simulations function in basically two ways, where the collisions are detected before they happen, *a priori*, or after, *a posteriori*.

In the a posteriori case, we advance the simulation by a small step each iteration and check if collisions have happened. If they did, we roll the simulation back a short while to before the collision happened and restore the situation correctly. In the a priori case, our collision detection algorithm needs to interpolate the positions of various bodies to predict whether a collision will happen or not.

A priori algorithms are more stable and correct, because the intersections of bodies that need to be fixed in the a posteriori case are unphysical. However, determining ahead of time when two bodies will collide is very difficult and it requires the collision detection algorithm to know about the physics of moving bodies.

Since we detect collisions a posteriori, we need to restore the situation to before the collision. Thus, we need to rewind each pair of intersecting spheres to before the collision. Assuming there is no external forcefield, the particles are in a state of uniform rectilinear motion. The equation we need to solve for the time t is

$$|\vec{r_1}(t) - \vec{r_2}(t)| = R_1 + R_2$$

with $\vec{r_i}$ the position vectors of the bodies and R_i their radii. Since their motion is uniform, this becomes

$$|\vec{r_1}t_0 + \vec{v_1}\Delta t - \vec{r_2}t_0 - \vec{v_2}\Delta t| = |\Delta\vec{r} + \Delta\vec{v}\Delta t| = R_1 + R_2$$

with $\Delta\vec{r}$ and $\Delta\vec{v}$ the particles' relative position and motion and Δt the sought after time difference. Squaring both sides, we find

$$(\Delta\vec{r})^2 + 2\Delta\vec{v}\Delta\vec{r}\Delta t + (\Delta\vec{v})^2(\Delta t)^2 = (R_1 + R_2)^2$$

The solutions of this quadratic equation are

$$\Delta t = \frac{-2\Delta\vec{v}\Delta\vec{r} \pm \sqrt{4(\Delta\vec{v}\Delta\vec{r})^2 - 4|\Delta\vec{v}|^2\left(|\Delta\vec{r}|^2 - (R_1+R_2)^2\right)}}{2|\Delta\vec{v}|^2}$$

As $\Delta\vec{v}\Delta\vec{r}$ is typically a negative number and $R_1 + R_2$ is always greater than $|\Delta\vec{r}|$, we take the positive square root and simplify

$$\Delta t = \frac{-\Delta\vec{v}\Delta\vec{r} + \sqrt{(\Delta\vec{v}\Delta\vec{r})^2 + |\Delta\vec{v}|^2((R_1+R_2)^2 - |\Delta\vec{r}|^2)}}{|\Delta\vec{v}|^2}$$

With this value, we can put the particles back in their original position before the collision. Now, we need to handle the collision itself.

1.1.1 One-dimensional elastic collision

Consider two particles denoted by subscripts 1 and 2. Let m_i be the masses, v_c be the the velocity of the center of mass (COM) frame, v_i the velocities before the collision, v_i' the velocities before collision in the COM frame, w_i the velocities after the collision and w_i' the velocities after collision in the COM frame. We assume the speed of both bodies is non-relativistic.

To considerably simplify the equations, we are going to transform to the intertial frame of the center of mass (COM). The COM velocity is

$$v_c = \frac{m_1 v_1 + m_2 v_2}{m_1 + m_2}$$

The resulting velocities in the new frame are

$$v_1' = v_1 - v_c = \frac{m_2}{m_1 + m_2}(v_1 - v_2)$$

and

$$v_2' = v_2 - v_c = \frac{m_1}{m_1 + m_2}(v_2 - v_1)$$

In this frame, the total momentum must be zero before and after the collision. The kinetic energy must also be conserved. Thus

$$p_1 + p_2 = q_1 + q_2 = 0$$

and

$$\frac{p_1^2}{2m_1} + \frac{p_2^2}{2m_2} = \frac{q_1^2}{2m_1} + \frac{q_2^2}{2m_2}$$

where p_i and q_i are the momenta respectively before and after the collision.

From this follows that

$$q_2 = -q_1 \qquad \text{and} \qquad p_2 = -p_1$$

which we can fill in in the energy equation

$$p_1^2 = q_1^2 \qquad \text{and} \qquad p_2^2 = q_2^2$$

The only possible solutions are $q_i = p_i$, that is no collision happened at all, or $q_i = -p_i$. This means that the velocities of both bodies are flipped.

$$w_i' = -v_i'$$

Thus we can derive the velocities in the original frame of reference

$$w_1 = w_1' + v_c = \frac{v_1(m_1 - m_2) + 2m_2u_2}{m_1 + m_2}$$

$$w_2 = w_2' + v_c = \frac{v_2(m_2 - m_1) + 2m_1u_1}{m_1 + m_2}$$

1.1.2 Two- and higher-dimensional elastic collisions

Because the colliding bodies are spheres, the only forces they can exert on each other at the contact point are according to the normal vector of each sphere at the contact point. The tangential component(s) of the momentum of each body do not change while the normal component is transformed according to the equations of an elastic collision. To get the normal vector, we simply normalise the displacement vector between the bodies

$$\vec{n} = \frac{\Delta\vec{r}}{|\Delta\vec{r}|}$$

The normal component of the velocity is

$$\vec{v_n} = \vec{v} \cdot \vec{n}$$

1.2 Spatial partitioning

Most many-particle simulations incorporate some sort of interaction between the particles dependent on their mutual distance. Of considerable interest are short-range interactions, for example when the particles represent atoms. Since a particle could hypothetically interact with every other particle, each pair needs to be tested separately. Thus $n(n-1)/2$ interactions will be considered, with n the number of particles. Most of these are useless because the particles are too far apart. This $O(n^2)$ complexity is undesirable for performance reasons. An often used technique is called **spatial partitioning**.

By splitting the world into smaller partitions, *boxes*, one only has to check for interactions between particles in nearby partitions. If we keep the particles per box a constant (x), the amount of boxes becomes n/x. The complexity of this problem reduces in this way to $O(x^2 \cdot n/x) = O(n)$, linear in the number of particles.

2 Implementation

2.1 Programming language

For this project, performance was a prime goal and thus the only contemplated programming languages were C and C++. Originally, we decided to implement the project in C++. It seemed to us that the clear distinction between the system, a partition and a particle lent itself well to an object-oriented programming language. Another advantage was the Standard

Template Library (STL) which provided us with an efficient doubly linked list. Lastly, overloading the addition and multilpication operators allowed for a natural expression of vector operations.

The nature of the problem, on the other hand, did not lend itself too well to OOP. The principles of information hiding and encapsulation added a considerable overhead to the code. Lastly, we have a lot more experience writing C when compared to C++. Given the time constraints, it was more natural to write in a language we knew better.

For these reasons, we decided to switch to C. This conversion only took an hour and the imperative paradigm seemed more fit to develop a performant program in a reasonable amount of time.

2.2 Data structures

The project needs a lot of manipulation of vector quantities, like position and velocity. For this we use the following straightforward C structure

```
1 struct Vec3 {
2         float x, y, z;
3 };
```

A particle is represented by its position and velocity. The particles needs to be contained in box so we store them in a circular doubly-linked list.

```
1 struct Particle {
2         Vec3 pos;
3         Vec3 vel;
4         struct Particle *prev, *next;
5 };
```

In this case, a box is little more than a pointer to any particle it contains. For various reasons, we keep track of the amount of particles in each box.

```
1 struct Box {
2         Particle *p;
3         int n;
4 };
```

2.3 Algorithms

To set up the initial configuration, the system is filled with the set amount of particles. A particle is placed in a random location with a random velocity from an isotropic distribution. The program then checks for collisions and if necessary, moves the particle to another region. With this approach, we can achieve a packing efficiency of $\sim$50%, compared to the theoretical maximum of $\sim$74%.

Each iteration, the simulation is advanced one timestep. This step consists of first checking for collisions with other particles, then the particles are checked for collisions with the walls and lastly every particle is moved in a straight line according to its velocity. Any particle that moves over the boundary of a partition is transfered to its new box.

```
1 function stepWorld()
```

```
        for each box:
                for each particle in the box:
                        if (the particle collides)
                                handleCollision();

        for each box at the boundary:
                for each particle in the box:
                        if (the particle collides with the wall)
                                bounceParticle();

        for each particle:
                advanceParticle();
                if (particle is not in original box)
                        transferParticle();
```

2.4 Interface

The parameters for the system are given to the program by commandline arguments. Methods to benchmark the performance of the program are also built in the software. The scripts used to gather the data are in the same repository as the code. Care was taken to only measure the time the processor spent calculating and not the time the program took to finish, which is very sensitive to external conditions.

At the start of the program the world is filled with a given amount of particles. The amount of time this procedure takes not added to the benchmark results. Then, either the system is rendered graphically for debugging purposes, or it loops a set amount of iterations. Optionally, the positions and velocities of each particle are printed. This way, we can plot a distribution of the speed or demonstrate Brownian motion, the results of which are explained in the next section.

3 Performance

4 Results

4.1 Physical results

To validate the correctness of the software, we have tested some important physical phenomena that are described by statiscal mechanics. We studied the Maxwell-Boltzmann distribution and Brownian motion. Other interesting phenomena include adsorption models, the Van der Waals corrections to the ideal gas law, diffusion, heat conduction, phase transitions, thin film phenomena, etc...

4.1.1 Maxwell-Boltzmann distribution

The Maxwell-Boltzmann speed distribution describes the probability density of the speed of a gas particle. The necessary temperature parameter was calculated from our simulation using

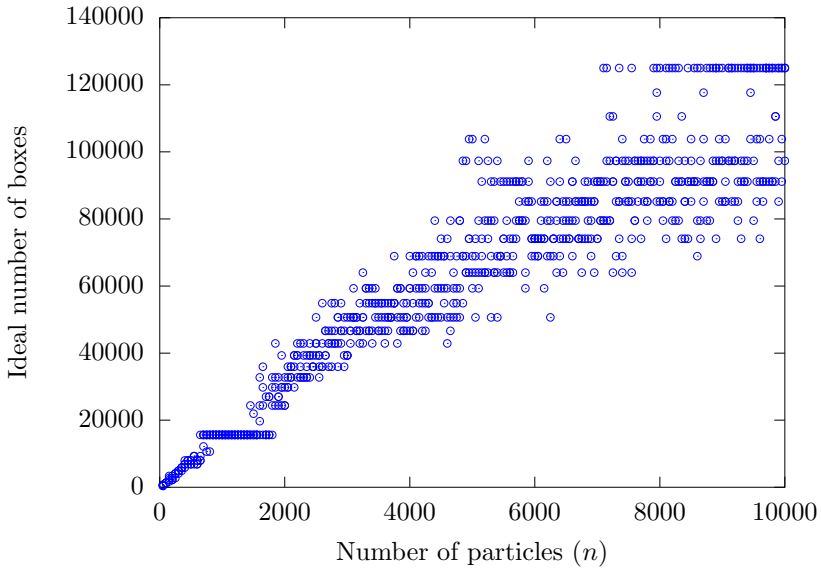

Figure 1: Ideal number of boxes for given number of particles. Worldsize $50 \times 50 \times 50$, radius 0.5.

the equipartition theorem.

$$E_{\text{avg, kin}} = \left\langle \frac{mv^2}{2} \right\rangle = \frac{3k_B T}{2}$$

With m the mass of each particle and k_B the Boltzmann constant. Both were set to unity in our simulation. Thus

$$T = \frac{\langle v^2 \rangle}{3}$$

The Maxwell-Boltzmann distribution follows

$$f(v)\,\mathrm{d}v = \left(\frac{\beta m}{2\pi}\right)^{3/2} 4\pi v^2 \exp\left(-\beta \frac{mv^2}{2}\right)\,\mathrm{d}v = \sqrt{\frac{2}{\pi T^3}} v^2 \exp\left(-\frac{v^2}{2T}\right)\,\mathrm{d}v$$

The resulting simulated distribution is shown in figure (4) together with the theoretical result. The results are clearly in perfect agreement with the predicted result.

4.1.2 Brownian motion

With a small change to the software, a single massive, huge particle was introduced in the system. Because of its relative size, space partitioning could not be used and thus we had to check each particle explicitly.

The trajectory of the particle was saved and plotted in figure (XXXBROWNXXX). Brownian motion is discernible in the path.

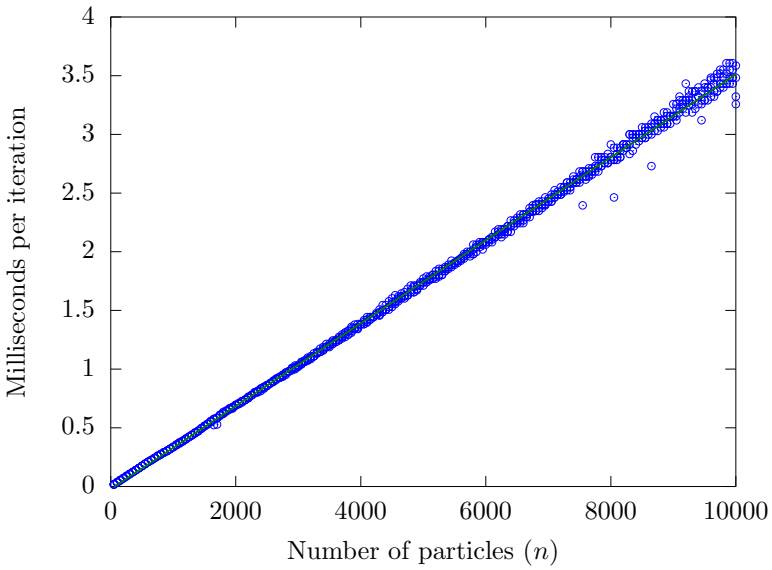

Figure 2: Linear complexity with Ideal number of boxes. Worldsize $50 \times 50 \times 50$, radius 0.5.

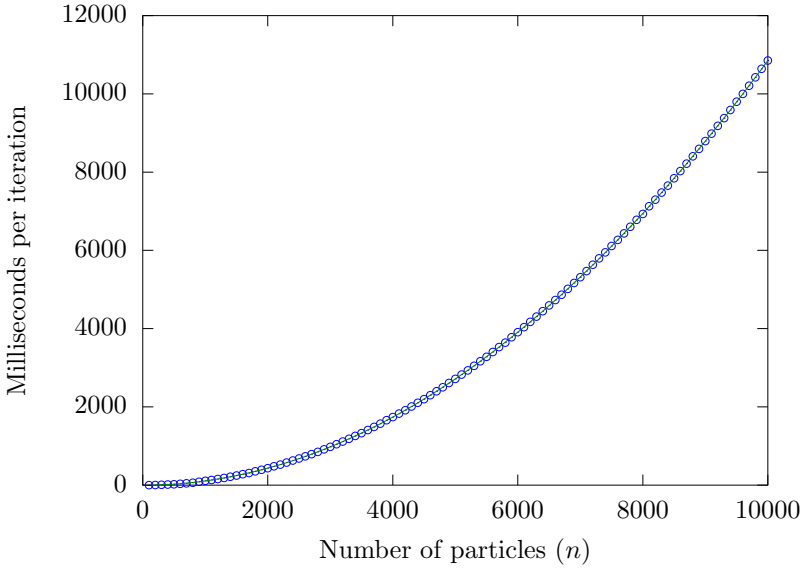

Figure 3: Quadratic complexity with one box. Worldsize $50 \times 50 \times 50$, radius 0.5.

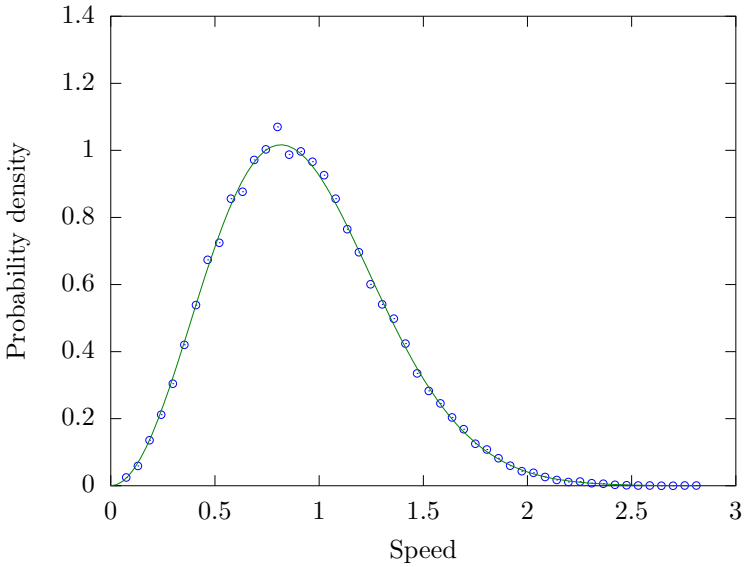

Figure 4: Speed distribution of a simulation and the expected theoretical distribution for a temperature of 1/9