Simulation of a many-particle system using space partitioning

Roald Frederickx
Kasper Meerts

5 November 2010

Abstract

In many research areas, one has the need to simulate a many-particle system. Short-range interactions are of considerable interest, for example when the particles represent atoms or molecules. The amount of pairs of particles that possibly need to be checked is of the order of $O(n^2)$. But the dynamics of a particle depend only on its close neighbours. Thus we implement a technique based on *spatial partitioning* where the world is divided in a fixed amount of partitions and particles only interact with other particles in nearby partitions. Simulations show that the complexity is reduced to $O(n)$, a dramatic improvement. A few phenomena are simulated and studied, e.g. the Maxwell-Boltzmann speed distribution and Brownian motion.

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 (3D) or disks (2D). It is important that this interaction is not only addressed correctly, but also efficiently. Therefore, one tries to avoid calculating computationally expensive trigonometric functions like sine and cosine and tries to keep square roots to a minimum.

Collision simulations function in basically two ways, the collisions can be detected before they happen, *a priori*, or afterwards, *a posteriori*.

In the a posteriori case, the simulation gets advanced by a small time step each iteration. If a collision took place, the simulation gets rolled back a short while to the exact moment of impact. The collision then gets handled and the situation is restored correctly. In the a priori case, the 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. Therefore we opted for the other method.

1.1.1 Rewinding time

Since we detect collisions a posteriori, we need to restore the situation to the exact time the collision took place. This entails moving each pair of intersecting sphere back in time.

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

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

The solutions of this quadratic equation are

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

As $\Delta\vec{v}\cdot\Delta\vec{r}$ 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

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

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

1.1.2 One-dimensional elastic collision

Consider two particles denoted by subscripts 1 and 2. Let m_i be the masses, v_c the velocity of the center of mass (COM) frame, v_i the velocities before the collision, v_i' the velocities before the collision in the COM frame, w_i the velocities after the collision and w_i' the velocities after the 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 inertial 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 before and after the collision, respectively.

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 new velocities in the original frame of reference

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

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

1.1.3 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 in the direction of the normal vector. 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

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

The change of this component can be calculated by plugging it into the equations above.

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 neutral 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 superfluous 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, or *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 thus reduces to $O(x^2 \cdot n/x) = O(n)$, linear in the number of particles.

2 Implementation

The code of this project will be hosted at `http://github.com/kmmeerts/ManyParticles` for the forseeable future.

2.1 Programming language

For this project, performance was a prime goal and thus the only programming languages considered 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 multiplication operators allowed for a natural expression of vector operations.

The performance critical nature of the problem, on the other hand, did not quite lend itself to the object-oriented programming paradigm. 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 switched to C.

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

```
struct Vec3 {
        float x, y, z;
};
```

A particle is represented by its position and velocity. The particles needs to be contained in a box so we store them in a circular doubly-linked list. This enables us to easily iterate over all the particles in a box.

```
struct Particle {
        struct Vec3 pos;
        struct Vec3 vel;
        struct Particle *prev, *next;
};
```

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.

```
struct Box {
        struct Particle *p;
        int n;
};
```

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, tries another location until it has found an empty spot.

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 transferred to its new box.

To find out whether a particle intersects another particle, we test it against every other particle in the same box. As this is the common case, this step is performed first. If this search yields no results, we check against particle in neighbouring boxes. Note that it is important that the box size is at least two times the radius of each particle for we could miss some collisions otherwise. A possible, unimplemented performance improvement would be only checking the boxes that intersect the particle.

The law of conversation of momentum allows to make another small performance improvement. The equations for the elastic collision are solved for one of the particles. Instead of doing the same for the other particle, the change in momentum of the first particle is subtracted from the second particle.

```
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. This overhead is not included in the benchmark results. Then, either the system is rendered graphically (using OpenGL) 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 given in the next section.

3 Performance

3.1 Testing platform

All tests were run on a laptop with an Intel Core2 Duo P8700 processor and 4 GB of dual channel DDR2 800 MHz RAM running Gentoo Linux on a 2.6.33 kernel with the kernel timer ticking at 1000 Hz. All CPU time measurements were accurate to 1 ms.

The code was compiled with version 4.4.5 of the GNU C Compiler using the -O3 optimization flag and building for the native architecture. Debugging options were omitted for the benchmark runs.

3.2 Filling the world

Our algorithm initializes the world by trying to insert a particle in a random position. If this particle collides with an existing particle, a new random location is tried, otherwise the particle is added.

Of course, there exists a limit to the amount of spheres that can be packed in a container. If the total volume of the spheres exceeds the available volume, it is of course impossible to place them in this container. Moreover, even if the total volume of the spheres is less than the available volume, there is no guarantee that the spheres will fit. Because of the round shape of the spheres, there will always be some space that goes to waste.

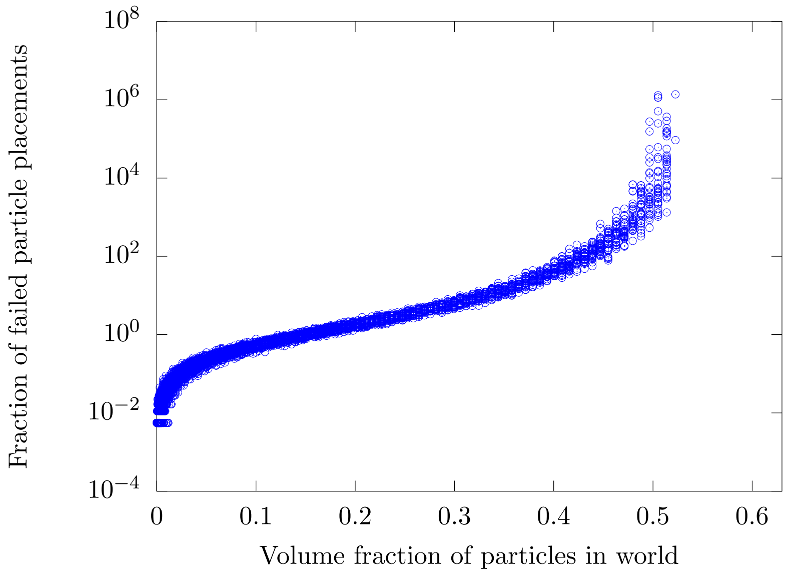

Figure 1: Fraction of failed particle placements w.r.t. packing density. Note the logarithmic scale on the vertical axis.

The problem of trying to fit as many spheres as possible in a fixed container is known as *sphere packing*. It is proven[2] that the most ideal sphere packing has a volume fraction of $\pi/\sqrt{18} \approx 0.74$. This happens when the spheres are placed on a regular grid called "face centered cubic". Our implementation, however, fills the world in a random, irregular way. This is a closely related problem known as *random close packing*. Another theorem[4] states that the packing in such a random way cannot exceed approximately 63%. This is the absolute maximum, assuming the randomly placed particles are “shaken” as much as possible to let them settle down and reach the lowest possible volume.

Our algorithm did no such “shaking”, and could only fill the world up to a density of 52%. A graph of the failed particle placements w.r.t. the packing density is given in figure 1. Notice the asymptotical behaviour near the 52% density. Above this density, the algorithm will loop forever, trying to insert a particle in a random position, but it will never succeed in doing so.

A render of the maximum density that our algorithm could achieve is shown in figure 2.

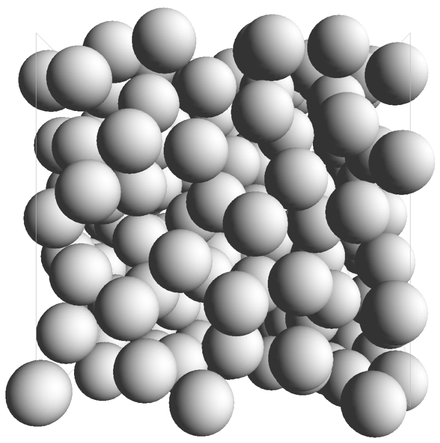

Figure 2: Render of a world with the maximum amount of particles.

3.3 Collisions: naive approach

The performance of a naive many-particle system with simple collisions was evaluated and the results are shown in figure 3. This naive approach was simulated by using just one box for the entire world. The quadratic complexity of the problem in function of the number of particles is clearly visible.

3.4 Space partitioning

To ameliorate the above result, the ideal number of boxes needs to be determined. The performance of a typical system in function of the number of partitions is given in figure 4. The performance peaks around the ideal number. We have determined the optimum number of boxes for a large amount of particles and graphed it in figure 5.

In that test, each particle has a radius of 0.5 and the worldsize was kept constant at $50 \times 50 \times 50$. The number of particles ranged from 0 to 10 000. With the maximum number of particles, the world was filled for about 4%, expressed as the fraction of volume of all particles with respect to the volume of the world.

From the clear linear trend in this plot, it follows that — for our implementation — the ideal number of boxes is roughly ten times the number of particles. This means that, on average, there will only be one particle in every tenth box. Thus we can conclude that detecting a collision is *much* harder than dealing with the overhead of iterating over the boxes.

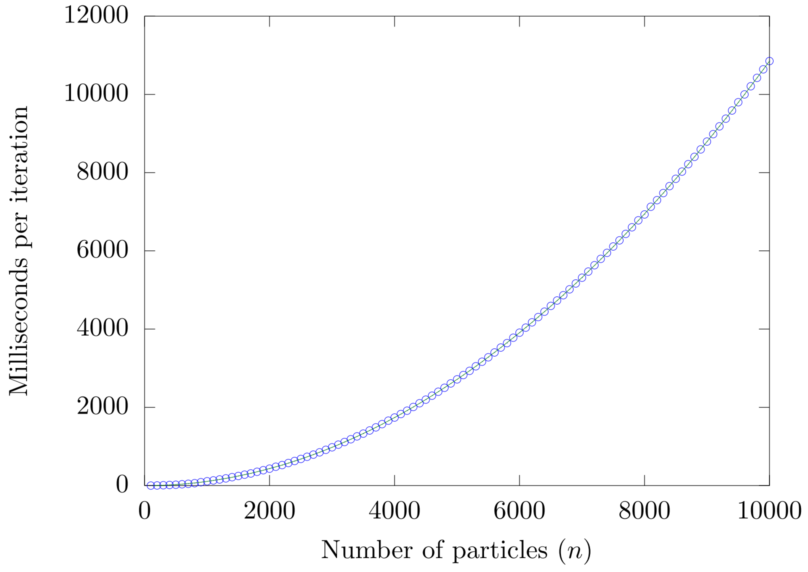

Figure 3: Quadratic complexity with one box. Worldsize $50 \times 50 \times 50$, radius 0.5.

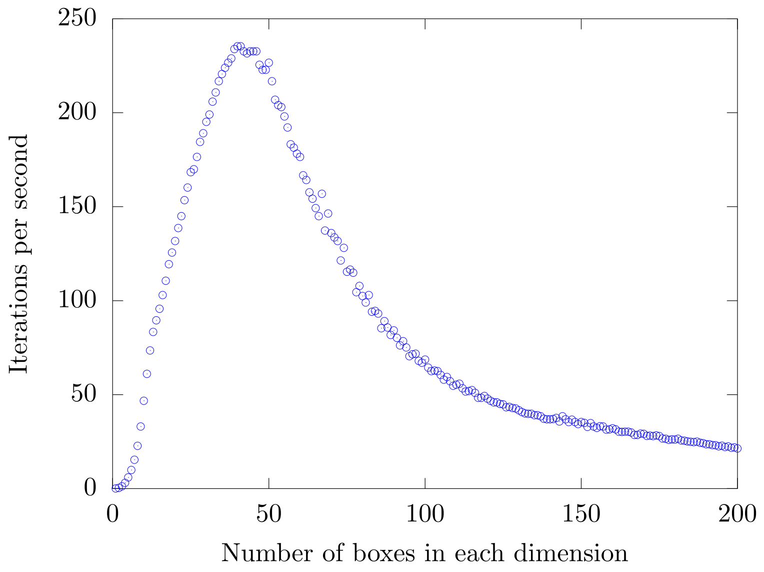

Figure 4: Effect of number of boxes on performance, for a fixed number of particles. Worldsize $50 \times 50 \times 50$, radius 0.1, 10 000 particles.

Another interesting feature of this curve is the anomaly found at around 1 000 to 2 000 particles. Here, the ideal number of boxes is constant and takes the value 15 625, or 25^3. Note that this means that every box has a size of exactly $2 \times 2 \times 2$. We believe that the preference for this amount of boxes comes from the fact that a computer can easily do arithmetic with numbers that are powers of two. Thus this anomaly has more to do with the hardware running the algorithm than the algorithm itself. Note that there also seems to be a plateau around

8 000 particles and more. This occurs at 125 000 boxes, or exactly 50^3. Each box now has size $1 \times 1 \times 1$, and calculations (divisions, multiplications) with the number 1 can be done quite efficiently indeed.

Recall that for the maximum number of particles chosen, only 4% of the available space is filled. However, we could not extend the graph to the right with more particles, since this would mean using boxes with a width smaller than 1. Given that the chosen radius of a particle was 0.5, this would mean that two particles could collide that are not in adjacent boxes. This collision would not get detected and the results would be useless.

The only way to circumvent this is using a smaller radius per particle. This was done in figure 6, where a radius of 0.1 was chosen.

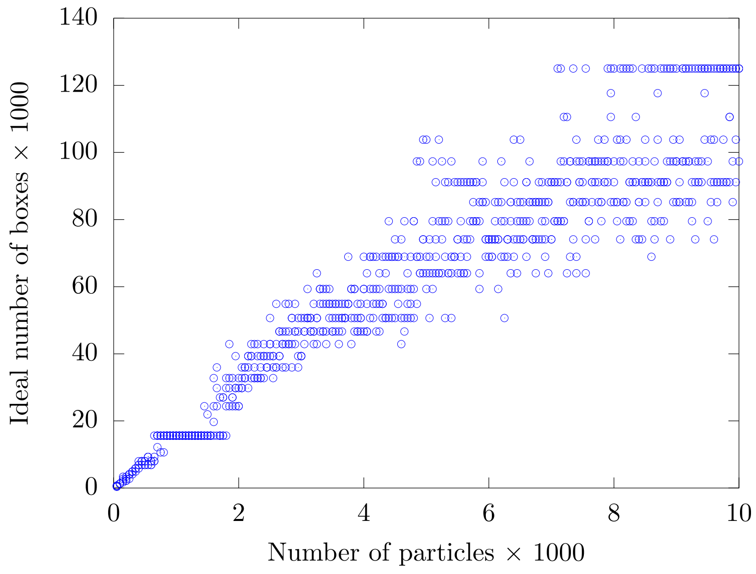

Figure 5: Ideal number of boxes for given (small) number of particles. Worldsize $50 \times 50 \times 50$, radius 0.5.

Note that the ideal case of $1 \times 1 \times 1$ sized boxes continues up to 20 000 particles (the plateau at $2 \times 2 \times 2$ sized boxes is not visible anymore in that figure).

Also note that the ideal number of boxes has changed from 10 per particle in the previous graph to around 7 boxes per particle for large numbers of particles. This is likely caused by the added memory demand for the extra particles and boxes and is once again more of a limitation of the hardware than of the algorithm. Indeed, modern computers make heavy use of caching data from memory on fast caches near the processor. When the working set of a program exceeds this available space, performance will decrease. Hence it makes sense to limit the amount of memory accesses when dealing with a large working set (a lot of particles). This translates into using less boxes.

The results for an even greater amount of particles are plotted in figure 7. The particle number reaches 1.2 million. This would be completely intractable using a naive approach. Once again, a plateau is visible, this time when using $200 \times 200 \times 200$ boxes. Also, the average number of boxes per particle seems to have stabilized in the vicinity of 10 again.

Lastly, the actual time complexity is plotted for the various number of particles. Each

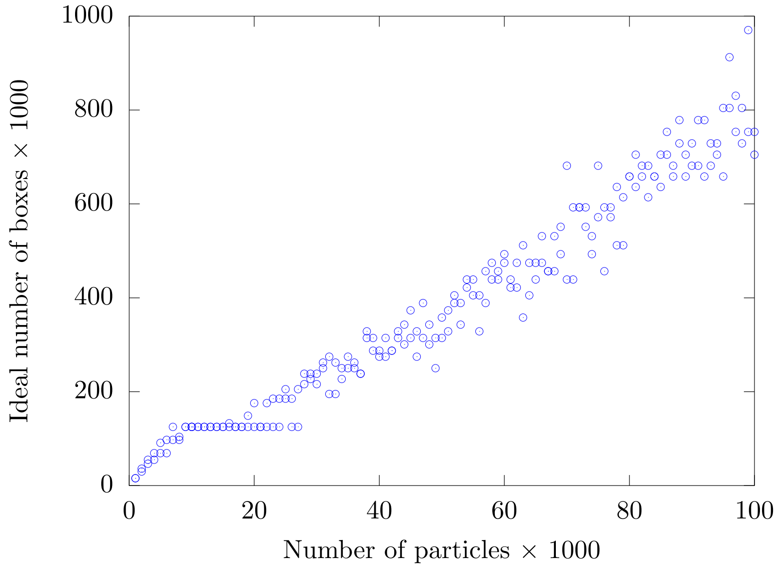

Figure 6: Ideal number of boxes for given number of particles. Worldsize $50 \times 50 \times 50$, radius 0.1.

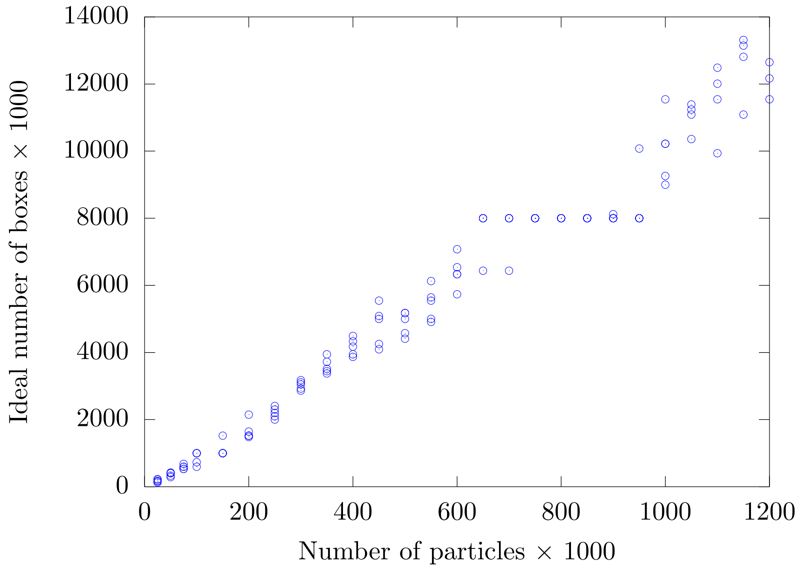

Figure 7: Ideal number of boxes for given (large) number of particles. Worldsize $50 \times 50 \times 50$, radius 0.1.

simulation was done using the ideal number of boxes calculated above. For the case of particles with a radius 0.5, the results are shown in figure 8. It is clear that the time complexity has gone from quadratic for the naive approach to linear when using space partitioning.

In figure 9, this graph gets extended for more particles (while reducing the radius of the particles to 0.1). Note the deviation from the linear trend. This can once again be ascribed

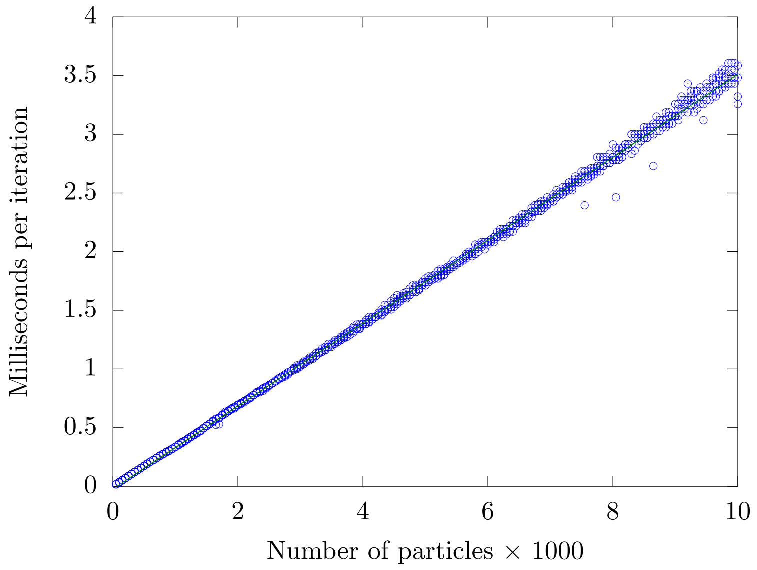

Figure 8: Linear complexity with ideal number of boxes. Worldsize $50 \times 50 \times 50$, radius 0.5.

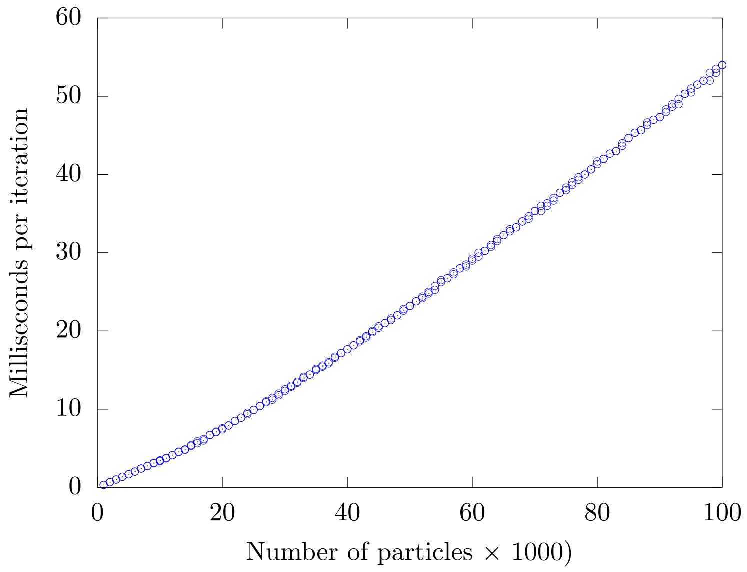

Figure 9: Small deviation from linear complexity with ideal number of boxes for large amount of particles. Worldsize $50 \times 50 \times 50$, radius 0.1.

to the hardware running the algorithm. The formal derivation for the linear time complexity assumed a constant fraction of particles in a box, x. We found above that for a small amount of particles, the ideal x-value was about $1/10$. For a large amount of particles, however, this value increased to about $1/7$ for 100 000 particles above. Thus this x depends on the number of particles and so the theoretically expected linear curve turns out to be slightly superlinear in practise.

Going to even higher particle numbers, this anomaly vanishes again, as seen in figure 10. Note that at the point of 1.2 *million* particles, every iteration still only takes about 1 second on the testing machine. For comparison, this would take 5 *years* with the naive model of figure 3.

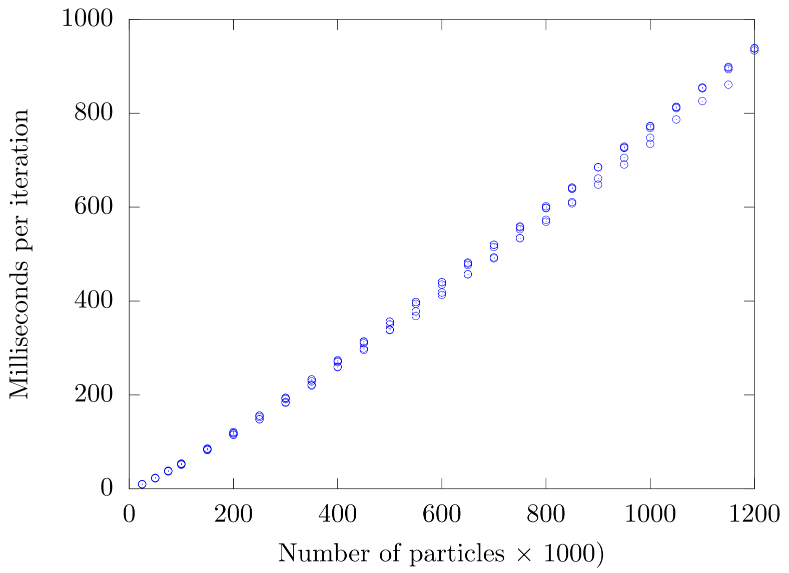

Figure 10: Stabilisation of linear complexity with ideal number of boxes for *very* large amount of particles. Worldsize $50 \times 50 \times 50$, radius 0.1.

3.5 Conclusion

In theory, space partitioning allows us to reduce the time complexity of a many-particles simulation from $O(n^2)$ to $O(n)$.

In our example, the ideal number of boxes to maximize the number of iterations per second turns out to be around 10 times more than the amount of particles. Due to the nature of the underlying hardware, this varies slightly as the memory demand grows larger. This also means that the ideal complexity that can be achieved in practice is slightly superlinear for some intermediate regimes. It must be noted, however, that this superlinear complexity is nowhere near as bad as the quadratic complexity of the naive implementation, and that it turns linear again for large numbers of particles.

4 Applications

To validate the correctness of the software, we have tested some important physical phenomena that are described by statistical 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 and capacity, phase transitions, thin film phenomena, etc...

4.1 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 the equipartition theorem.

$$\langle E_{\mathrm{kin}} \rangle = \left\langle \frac{mv^2}{2} \right\rangle = \frac{3kT}{2}$$

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

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

The well known Maxwell-Boltzmann distribution[3] reads

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

with T given in function of the expectation value of the squared velocity above.

The velocity of our particles got initialized as a vector of fixed length 1, pointing in a random direction. Thus, the inital speed distribution has the shape of a delta distribution around the speed 1. Through collisions between particles, these reach thermal equilibrium. While doing so, the sharp peak in the speed distribution relaxes to approximate the Maxwell-Boltzmann distribution. Figure 11 depicts this relaxation. The parameters for this simulation were: a worldsize of $48 \times 48 \times 48$, 4000 particles with radius 0.2 and a timestep of 0.01. Plots were made after 5 000, 10 000 and 15 000 iterations. The distribution converges the theoretical Maxwell-Boltzmann distribution. The peak in the middle of the plot is a remnant of the initial delta distribution of the speeds.

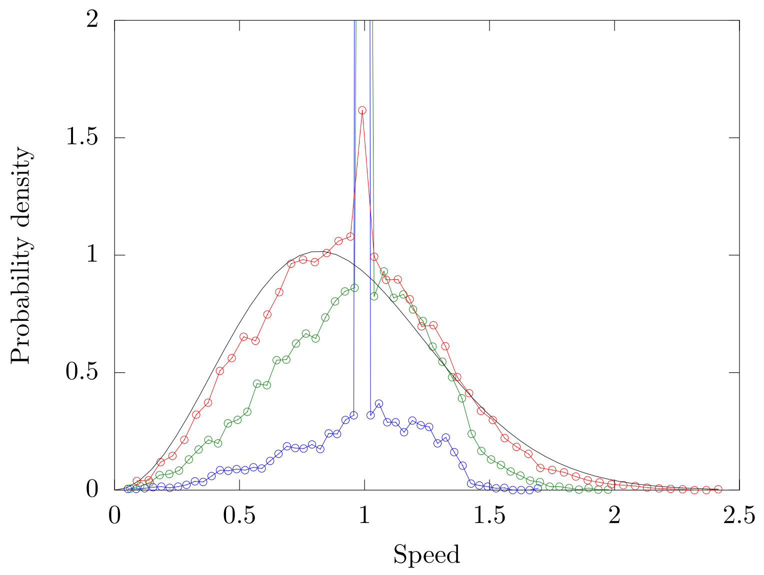

Figure 11: Relaxation of the speed distribution after 5 000 (blue), 10 000 (green) and 15 000 (red) iterations. The black curve is the theoretical equilibrium distribution

The average of three simulated distributions is shown in figure 13 together with the theoretical result. The parameters used were: a worldsize of $60 \times 60 \times 60$, 8000 particles with radius 0.2 and a relaxation time of 15 000 iterations with timestep 0.05.

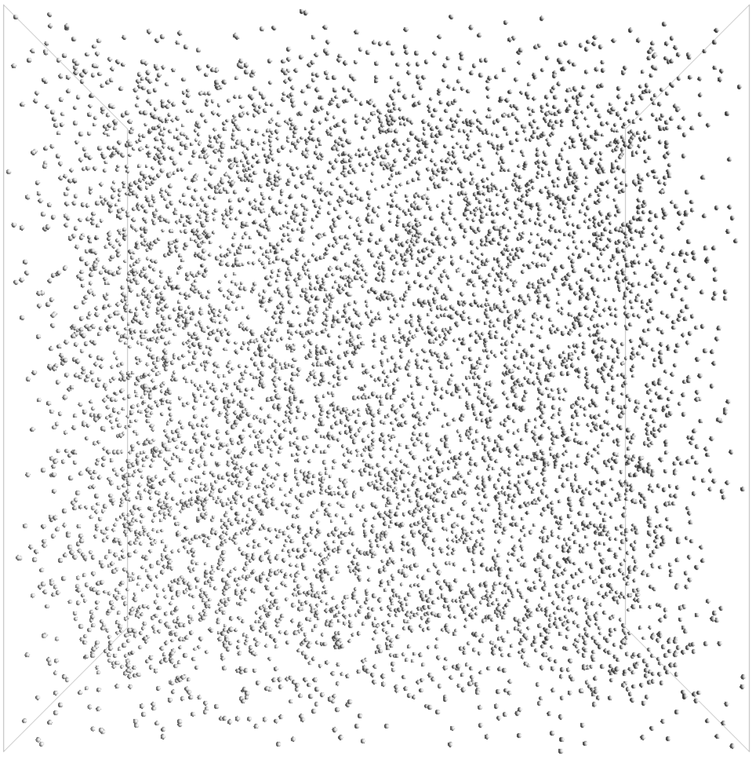

Figure 12: Rendering of the setup used to determine the Maxwell distribution

Note that the density in our example was about 0.1% (the fraction of total particle volume with respect to the world size). This roughly approximates the density of gasses at room temperature. The same result can be found with more dense packings, and with less dense (but this involves a lot more iterations, as collisions will be very rare).

The temperature of the theoretical curve was obtained from the formula above, using that the temperature at the beginning[1] was given by $\langle v^2 \rangle / 3$, in our case 1/3.

The results are clearly in perfect agreement with the predicted result.

4.2 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 for this new particle and each smaller particle had to be checked for interactions with the huge particle separately.

[1]Note that it's not quite appropriate to talk about a temperature in this case. We aren't in thermal equilibrium yet and there is no entropy to calculate $\partial E/\partial S$. Nevertheless, using the equipartition theorem to define the temperature works just fine.

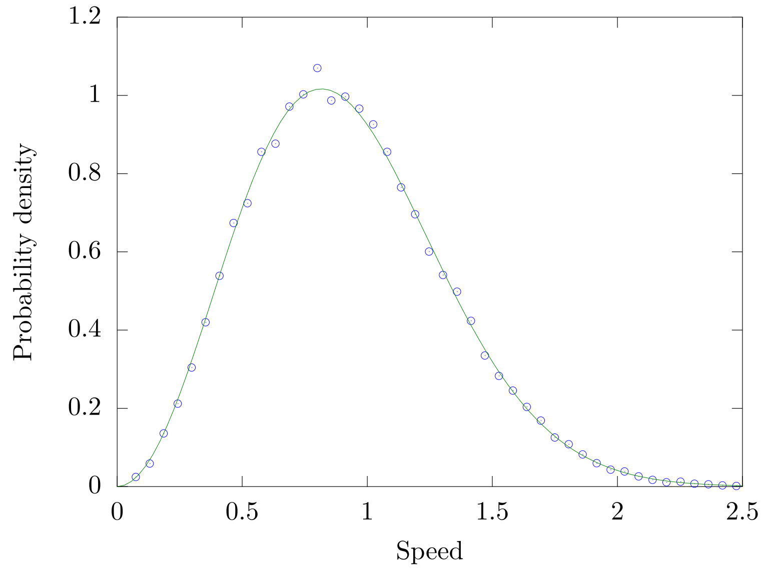

Figure 13: Speed distribution of a simulation and the expected theoretical distribution for a temperature of 1/3

This of course did not change the linear complexity of the algorithm and had a negligible effect on performance.

We placed a particle of radius 10 and mass 1 in the center of a $320 \times 320 \times 320$ world without initial velocity. The rest of the world was filled randomly with 10 000 particles of radius 0.5 and of the same mass. The velocities of these particles had a fixed length of 1 and were pointing in random directions.

The trajectory of the large particle during four runs of the simulation is plotted in figure 14. Brownian motion is discernible in the path.

4.3 Ideal gas law and Van der Waals corrections

If every collision with the wall is noted and the change in momentum is recorded, an expression for the pressure on the wall can be obtained. Using the equipartition theorem to retrieve the temperature, it is possible to simulate the ideal gas law with the additional Van der Waals correction for particles with finite volume. No such simulations were performed.

4.4 Heat conduction and specific heat

Heat conduction is another phenomenon that can be studied. No results are shown in this article but we propose a method how one can implement this using the software.

The first proposal models heat conduction in a gas. We propose to give two opposite walls a different temperature. When a particle collides with a wall, the normal component with respect to the wall gets mixed randomly with the momentum it would have had if it were at the temperature of the wall.

This is enough to model heat transfer from one wall to another. With a sufficiently dense gas, and a sufficient distance between the walls, it is even possible to graph a temperature

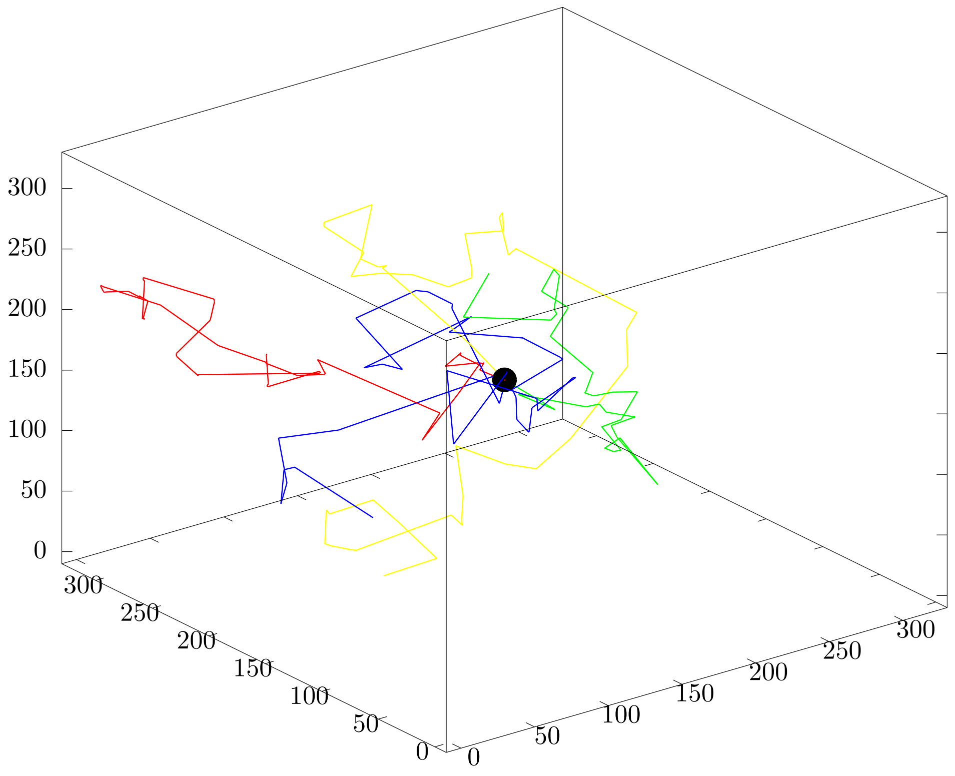

Figure 14: Four trajectories of Brownian motion starting from the center dot. (worldsize $320 \times 320 \times 320$, 10 000 particles of radius 0.5, radius of large particle: 10, mass: 1)

profile by considering the average energy of the particles in function of their distance to the walls.

A second proposal models heat conduction by electrons in a metal. The setting is the same as the previous example, but with an additional regular grid of stationary particles that fills the world. These particles can exchange kinetic energy with the moving particles after a collision. The stationary particle stores its kinetic energy in a vibration mode. This gives an idea of the temperature distribution of the system and doesn't require a very dense gas.

By varying the temperature of the walls, the specific heat can be determined. The vibration modes of the last proposition could even be linked to phonons in the Debye model of a solid to model their contribution to the specific heat.

4.5 Langmuir adsorption model

A simple surface adsorption model can be implemented. We propose that particles making contact with a wall can be adsorbed if their momentum normal to the wall is small enough.

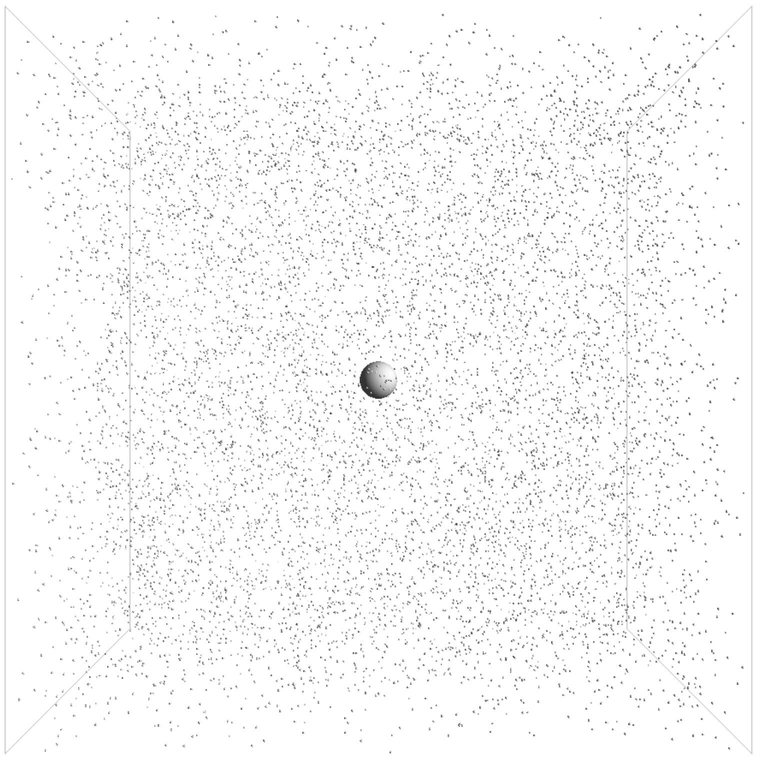

Figure 15: Render of the setup used to generate the Brownian motion

Once adsorbed, their lost kinetic energy is stored in a vibration mode and the particles can move freely in two dimensions on the wall. The particle can be desorped if, through interactions with other particles, it can gain enough kinetic energy. This second particle can be a particle that is itself bound to the wall and moving over its surface, or a particle from the gas that collides with the bound particle.

The proportion of adsorbed particles given by the Langmuir model is given by[1]

$$\frac{N_{\text{ads}}}{N_{\text{gas}}} = \frac{A\lambda_{\text{th}}}{V} e^{\epsilon/kT}$$

where N_{ads} is the number of adsorbed particles, N_{gas} the number of unbound particles, λ_{th} the thermal de Broglie wavelength $\lambda_{\text{th}}^2 = h^2/2\pi mkT$, A the surface area and ϵ the binding energy of the wall.

Note the occurence of the constant of Planck in the expression for the thermal wavelength. It would be interesting to see if our proposed simple (and fully classical) model could be used to estimate this fundamental constant of quantum mechanics.

4.6 Conclusion

The simulations of the Maxwell-Boltzmann distribution and Brownian motion show that our implementation gives the correct results. The proposed models for the ideal gas law with a Van der Waals correction, the heat conduction and heat capacity and the proposed classical model of adsorption are interesting subjcts for further study based on the given performant framework of colliding spheres with space partitioning.

5 Final thoughts

When simulating a many-particle system, considerable performance improvements can be achieved using the technique of space partitioning. We have shown that the naive quadratic complexity can be reduced to linear complexity. This allows one to simulate various physical systems such as a gasses and solids efficiently without losing physical correctness, as was shown for the case of Brownian motion and the Maxwell-Boltzmann speed distribution. Additional phenomena suitable for implementation were suggested for further studies, such as heat conduction and capacity, the Van der Waals correction and a simple classical model of adsorption.

References

[1] Ralph Baierlein. *Thermal Physics.* Cambridge University Press, 1999.

[2] Thomas C. Hales. A proof of the kepler conjecture. *Annals of Mathematics*, 162:1065 – 1185, 2005.

[3] Daniel V. Schroeder. *An introduction to Thermal Physics.* Addison Wesley Longman, 2000.

[4] Chaoming Song, Ping Wang, and Hernan A. Makse. A phase diagram for jammed matter. *Nature*, 453:629 – 632, May 2008.