Event-Driven Simulations of Particles in a Box

Oda Lauten[a]

[a]*Department of Physics, Norwegian University of Science and Technology, N-7491 Trondheim, Norway.*

Abstract

In this computer experiment, the motion of particles is simulated according to the laws of elastic collision. The method used is event-driven simulation. The system under consideration is a square box filled with a gas made up of hard disks. Five different regimes were explored: i) The scattering angle as a function of impact parameter when a particle collide with another particle which is larger and heavier; ii) the speed distribution of a gas consisting of one type of particle; iii) the speed distribution of two gases; iv) the development of kinetic energy of a mixture of two gases; and v) the crater formation when a large projectile collides with a wall of smaller and lighter particles.

1. Introduction

The hard disk model is an idealised model of the motion of two-dimensional particles in a container. The system under consideration is a two-dimensional square box with lengths $L_x, Ly = 1$. The box is filled with hard disks, from now on called particles. The particles interact via elastic collisions with each other and with the reflecting boundary of the container, from now on called a wall. Between collisions, the particles move in straight lines with constant velocities. It is assumed that all collisions are instantaneous, and the possibility of simultaneous collisions between more than two objects is ignored.

The formulas for at what time two particles will collide and for at what time a particle will collide with a wall, can be found in the assignment text [2]. Also the formulas for the new velocites after collision can be found there.

2. Implementation

2.1. Numerical method

Since the collisions do not occur at regular time intervals, it is appropriate to use an event-driven simulation. An event-driven simulation uses adaptable time steps equal to the time until the next event. This can be done since the particles follow an undisturbed translational motion until an event occurs. The generic algorithm is as follows:

1. Initialise all particles.
2. Initialise all collisions that might take place if the particles just follows a straight line with constant velocity, and store the collision times in a priority queue.
3. Loop through all collisions chronologically starting with the first occurring:
   (a) Move all particles forward in time until the earliest collision.
   (b) For particle(s) involved in the collision, update the collision counter, calculate new velocities and new collision times, and store the collision times in the priority queue.
   (c) Identify the new earliest collision which is valid[1].

2.2. Data structures

The particles have a known position $\boldsymbol{x}_i = (x_i, y_i)$, velocity $\boldsymbol{v}_i = (v_{xi}, v_{yi})$, radius r_i, mass m_i, a counter which tells how many collisions the particle have been in and a variable which stores the last time they were in a collision. All of these variables are stored in objects of the class `Particle`. The class also have a method to increment the collision counter, a method to update the position of the

[1]By valid collision it is meant a collision for which the collision count of the particle(s) involved is still the same as when the collision was stored in the priority queue. If the collision count is no longer the same at the time of collision, the involved particle(s) have collided with something else in the meantime.

particle to the new time step and a method to set the new velocity of a particle after a collision.

For each iteration of the event-driven simulation, all particles have to update their position, but only those involved in the collision have to update their velocity vector and collision counter. Therefore a class `Collision` is introduced containing the time of the collision, the id of the particle(s) involved and the collision count of the particle(s) involved at the time the collision was added to the priority queue.

The new position and new velocity for the involved particle(s) in the collision could also be stored in the `Collision`↩ -object. This would be a waste of computer time, since before the time of the collision, the involved particle(s) could be involved in one or more collisions with other particles, and hence, the collision is not valid any longer and the whole object is discarded. Instead the new velocities after a collision is calculated only after the collision is taken out of the priority queue.

2.3. Priority queue

The priority queue needs to be ordered by increasing time. It has to be initialised with the possible collisions from the start state. Then it is iterated over, and along the way new collisions are calculated and inserted at the appropriate place in the queue. Therefore, the priority queue needs to have a method which takes an element and inserts it into the queue and a method which returns the smallest element in the queue. An appropriate data structure for this is the min heap. The execution times of a min heap is $\mathcal{O}(\log n)$, where n is the size of the queue. In this simulation there was not implemented a priority queue from scratch, but rather used the implementation of the heap queue from the python module `heapq` with the methods `heappush` and `heappop`.

The heap can take in elements which are tuples and then sort by the first element in the tuple. If there exists tuples that have equal first elements it will sort by the second element of the tuple and so forth. A challenge with the first implementation of the code, before the class `Collision` was introduced, was that the second element in the tuple was an array and did not have a comparison method, so when two or more collisions occured at the same time, the code crashed. This was solved by implementing a class named `Collison` with a self-defined comparison operation `__comp__(self, other)` which returns a negative integer if the collision time of `self` is smaller than the collision time of `other` and positive if `self.time>=other.time`.

2.4. Stopping condition

It is useful to have a stopping condition for the loop, so that the particle state can be evaluated. Since problems 1 to 4 in the assignment is interested in the state with a certain average collision count, the loop is implemented in such a way that it runs until the average collision count of all the particles reach a certain value. It is also implemented that the loop ends when the energy goes below 10% of the initial energy since this is beneficial for problem 5 in the assignment.

2.5. Uniformly distributed particles

To initialise N uniformly distributed particles in an area $(x_a, x_b) \times (y_c, y_d)$, the x_i of the particle is drawn from a uniform distribution of numbers from the half-open interval $[x_a - r_i, x_b - r_i)$, and similarly, the y_i of the particle is drawn from the half-open interval $[y_c - r_i, y_d - r_i)$. The random number generator used to do this was the `numpy`↩ `.random.uniform`. The reason that r_i is subtracted, is to make sure that none of the particles overlap with the walls. Then, the function `is_overlap()` checks if the newly drawn particle position overlaps with any of the existing particles. The function checks this by calculating the distance between the new particle and every other particle. If the distance is smaller than the two particles radii combined, then the particles overlap and the newly drawn particle position must be discarded. If the distance is greater than the combined radii for all the other particles, then the particle does not overlap with any other and can be placed there.

When the packing fraction of particles gets around $\frac{1}{2}$, as it does for the final task of this exam, the placing of a new particle takes much computation time. Therefore, it was implemented in such a way that the positions of a given number of particles was stored to file the first time the code was run. Then, for the later runs, the positions of the particles was loaded from the file.

2.6. Plot of particle state

To plot the particles in a figure the `matplotlib.pyplot.Circle()` was used. This uses the position array (x_i, y_i) and the radius r_i to plot a circle in the right position.

2.7. Multiprocessing

The function that calculates if and when a particle will collide with another particle is likely the most time-consuming part of the code. Parallelisation of this part would have been an efficient way to save computing time, but there was not enough time to implement that. A simpler approach was to use multiprocessing of some parts of the code with Python's package `multiprocessing`.

2.8. TC model

In the event-driven simulation, the time during which two particles are in contact is zero. However, the simulation run into difficulties when the time between events, Δt_n, becomes too small and the situation known as inelastic collapse occurs. This happens typically for systems with inelastic collision, restitution coefficient $\xi < 1$, where energy is lost at each collision. If the number of collisions per time goes to infinity, this would make the simulation stop at a fixed time, since all collisions taking place at time t must be resolved before the simulation can reach time $t + \delta t$.

The TC model takes into account a finite contact of duration. This is a physical relevant parameter since there cannot happen an infinite number of collisions in a finite time interval, every contact takes some finite time. So far, collisions has been assumed to be instantaneous, but by introducing the duration of contact t_c any collision that happen within a time interval t_c involving the same particle(s) are assumed to be perfectly elastic.

This is implemented by storing the last collision time for each particle. Then, when resolving a collision, it is checked if the involved particle(s) last collision happened less than t_c ago. If they did, ξ is set to 1 for that collision. By adding this physical relevant parameter some computing time is saved. This is only relevant for the two final tasks of the exam, since the three first are with $\xi = 1$.

3. Results and discussion

3.1. Scattering angle as a function of impact parameter

First, the elastic scattering of two hard particles was studied. One small and light particle with velocity in the positive x-direction hits one large and heavy particle which is stationary. The large and heavy particle had $r_1 = 0.1$ and $m_1 = 10^6$, while the small and light particle had $r_2 = 0.001$, $m_2 = 1$ and $\boldsymbol{v} = [1, 0]$.

The perpendicular distance between the path of the small particle's centre to the centre of the large particle is called the impact parameter b. By running multiple simulations of the small particle hitting the larger one with different values of b, a plot showing the scattering angle θ as a function of impact parameter was obtained, see figure 1. The figure shows that when the small particle hits straight on the centre of the larger particle, $b = 0$, then the small particle is deflected in the opposite direction. As the impact parameter gets bigger, the particle gets less deflected. When the impact parameter is greater than the combined radii of the two particles, the small particle is not deflected at all. The numerical results is in good agreement with the exact relationship one gets by geometrical considerations, $b = R_{12} \cos \frac{\theta}{2}$, where $R_{12} = r_1 + r_2$.

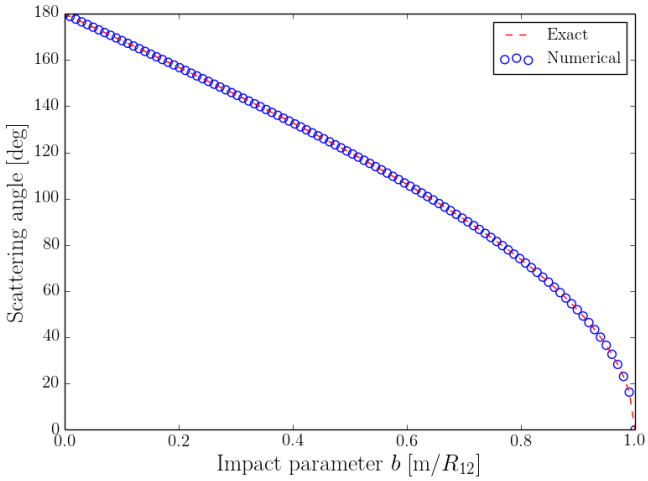


Figure 1: The scattering angle as a function of impact parameter for when one small and light particle with velocity in the positive x-direction hits one stationary, large and heavy particle. The collision is considered elastic with restitution coefficient $\xi = 1$. The dotted red line represent the exact solution obtained from geometrical considerations.

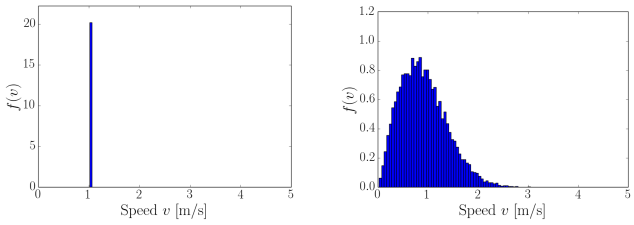
3.2. Speed distribution of a gas

Second, the initial and final speed distribution of a two-dimensional box with 1000 equal particles was studied. The particles had $m_i = 1$ and $r_i = 0.001$. The particles started out at uniformly distributed random positions and random velocities given by $\boldsymbol{v} = [v_0 \cos\theta, v_0 \sin\theta]$ where $v_0 = 1$ was the same for all the particles and $\theta \in [0, 2\pi]$ was a uniformly distributed angle. The restitution angle was set to $\xi = 1$. The initial speed distribution is a delta function centred around v_0, see figure 2(a). Then the simulation ran until it reached equilibrium, namely when the average number of collisions per particle was $\gg 1$. To get a smooth distribution the system ran 15 times. The final speed distribution resembles the Maxwell-Boltzmann probability distribution, see figure 2(b).

The speed distribution in an ideal gas of particles should follow the Maxwell-Boltzmann distribution [3]

$$f(v) = 4\pi \left(\frac{m}{2\pi kT}\right)^{3/2} v^2 \exp\left(-\frac{mv^2}{2kT}\right), \qquad (1)$$

where k is the Boltzmann constant, T is the thermodynamic temperature and m is the mass of the particle.



(a) Initial speed distribution. (b) Final speed distribution.

Figure 2: The initial and final speed distribution of 1000 particles in a two-dimensional box. The particles starts out at uniformly distributed random positions and random velocities given by $\boldsymbol{v} = [v_0 \cos\theta, v_0 \sin\theta]$ where v_0 is the same for all particles and $\theta \in [0, 2\pi]$ is a uniformly distributed angle. The restitution coefficient is $\xi = 1$, the mass $m = 1$ and the radius $r = 0.001$.

3.3. Speed distribution of a mixture of two gases

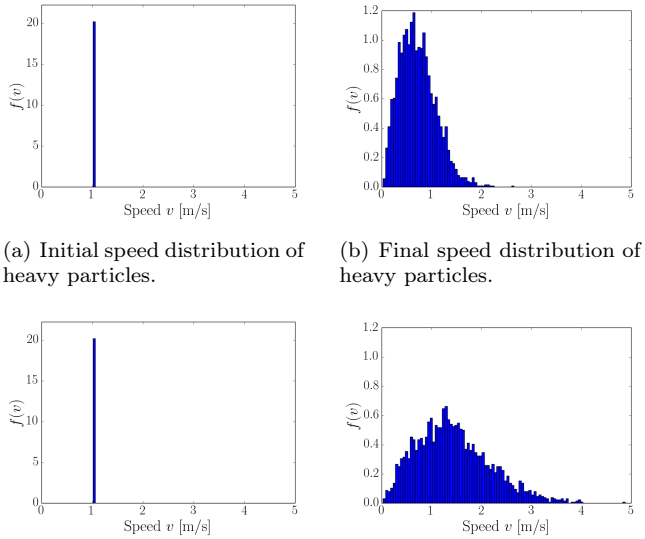
Third, the initial and final speed distribution of a two-dimensional box with 1000 particles of two different masses was studied. Half of the particles had mass $m = m_0$ and the other half had mass $m = 4m_0$, where $m_0 = 1$. All the particles had radius $r = 0.001$. The particles initial setup was equal to that for the homogen case presented earlier. Then the simulation ran until it reached equilibrium, namely when the average number of collisions per particle was $\gg 1$. To get a smoother distribution the system ran 5 times.

The initial speed distribution for the light particles, see figure 3(a), evolved into a final speed distribution resembling the Maxwell-Boltzmann probability distribution from Eq. (1), see figure 3(b). It increases parabolically from zero for low speeds, then it is curved downwards around the maximum, and decreases exponentially for high speeds. The average speed of the light particles was

$$\langle v \rangle_{\text{light}} = 1.404,$$

and the average kinetic energy for the light particles was

$$\langle E_k \rangle_{\text{light}} = 1.275.$$



(a) Initial speed distribution of heavy particles. (b) Final speed distribution of heavy particles.

(c) Initial speed distribution of light particles. (d) Final speed distribution of light particles.

Figure 3: The initial and final speed distributions of 1000 particles in a two-dimensional box. Half of the particles have mass $m = m_0$ and the other half have mass $m = 4m_0$. The particles starts out at uniformly distributed random positions and random velocities given by $\boldsymbol{v} = [v_0 \cos\theta, v_0 \sin\theta]$ where v_0 is the same for all particles and $\theta \in [0, 2\pi]$ is a uniformly distributed angle. The restitution coefficient is $\xi = 1$, the mass $m_0 = 1$ and the radius $r = 0.001$.

Also the speed distribution of the heavy particles evolved from a delta function centred around v_0, see figure 3(c), to the final speed distribution resembling the Maxwell-

Boltzmann probability distribution, see figure 3(d). The average speed of the heavy particles was

$$\langle v \rangle_{\text{heavy}} = 0.697,$$

which is about half the velocity of the lighter particles. The average kinetic energy for the heavy particles was

$$\langle E_k \rangle_{\text{heavy}} = 1.225,$$

which is almost equal to the kinetic energy of the light particles. This is as it should be in thermodynamic equilibrium.

When m in Eq. (1) decreases, the distribution gets wider and the amplitude decreases. This trend can also be seen for the two different particles under consideration; as the mass of the particles decreases, the distribution flattens out, see figure 3.

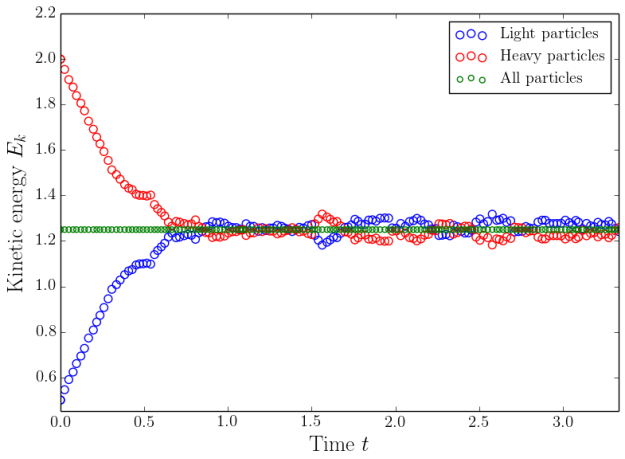
3.4. Development of kinetic energy for a mixture of two gases

Figure 4(a) shows the development of the average kinetic energy of the light particles, the heavy particles and both together. With $\xi = 1$, the system reaches equilibrium, and the average kinetic energy of the two gases is the same. According to kinetic theory [1], the average kinetic energy of a gas of particles is directly proportional to the temperature and all gases at a given temperature have the same kinetic temperature. Therefore, the two different particle gases must have the same temperature, and hence, they are in thermodynamic equilibrium.

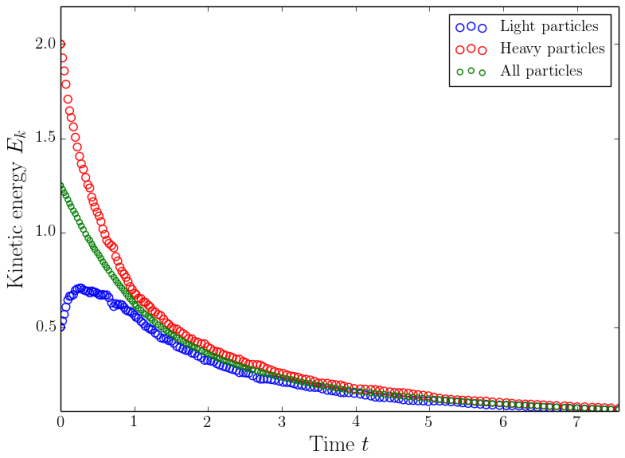
On the other hand, when $\xi < 0$, the system does not reach equilibrium, see figure 4(b) and figure 4(c). The kinetic energy is constantly decreasing. Hence, neither the kinetic energy nor the temperature is the same for the two gases. The lower the restitution coefficient is, the fewer collisions are needed to decay the average kinetic energy.

3.5. Crater formation

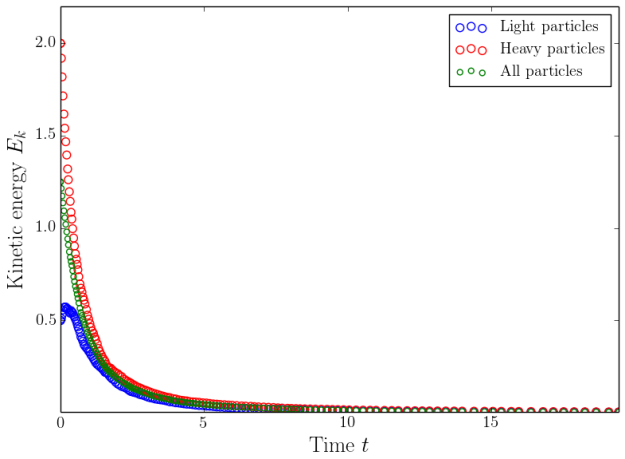
Finally, the formation of a crater following a projectile impact was studied. A system consisting of 5001 particles was initialised. One particle had larger mass and radius than the others, and would operate as the projectile. It had



(a) Restitution coefficient $\xi = 1$.



(b) Restitution coefficient $\xi = 0.9$.



(c) Restitution coefficient $\xi = 0.8$.

Figure 4: The development of the average kinetic energy of 1000 particles in a two-dimensional box for different values of the restitution coefficient. Half of the particles have mass $m = m_0$ and the other half have mass $m = 4m_0$. The particles starts out at uniformly distributed random positions and random velocities given by $\boldsymbol{v} = [v_0 \cos\theta, v_0 \sin\theta]$ where v_0 is the same for all particles and $\theta \in [0, 2\pi]$ is a uniformly distributed angle. The mass $m_0 = 1$ and the radius $r = 0.001$.

an initial position of $\boldsymbol{x}_0 = [0.5, 0.75]$ and a velocity in the negative y-direction of $\boldsymbol{v}_0 = [0, -v_0]$. The remaining 1000 particles started out stationary at uniformly distributed random positions within the area bounded by $x = 0$, $x = 1$, $y = 0$ and $y = 0.5$, see figure 5(a). These had all the same mass of $m = 1$. To get a packing fraction of $\frac{1}{2}$ in the lower half of the box, the radius was set to be

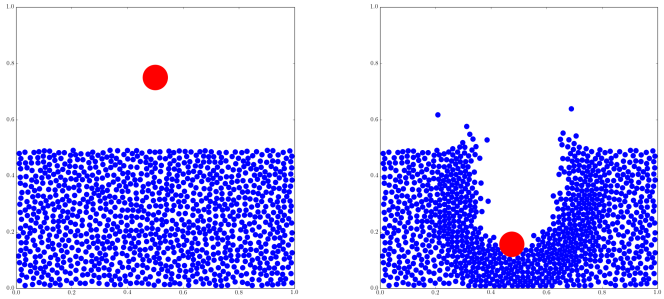
$$r = (4\pi N)^{-\frac{1}{2}},$$

where N is the number of particles. For $N = 1000$, the radius was about $8.92e - 3$. For the first simulation, the mass of the projectile was set to 25 times the mass of the smaller particles, the radius of the projectile was set to five times the radius of the smaller particles and the projectile was given a speed of $v_0 = 5$. The restitution coefficient was set to $\xi = 0.5$. The simulation ran until only 10 % of the initial energy remained, see figure 5(b).

The size of the crater was measured by counting how many particles that was affected by the projectile and then divided by the total number of particles,

$$\text{crater size} = \frac{\text{number of affected particles}}{\text{number of particles}}.$$

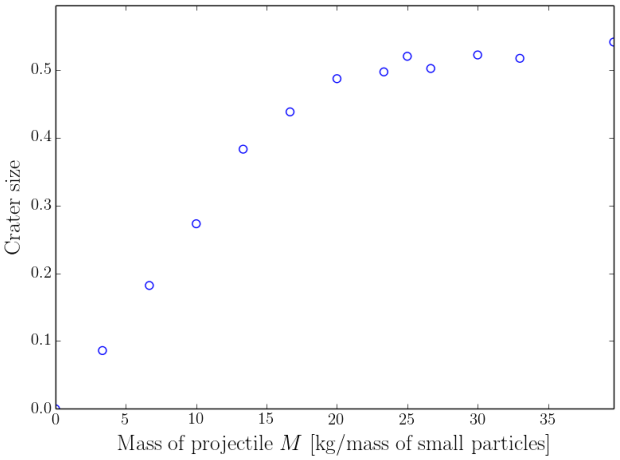
The particle was considered affected if it had moved more than $0.1r$ from where it initially was positioned.



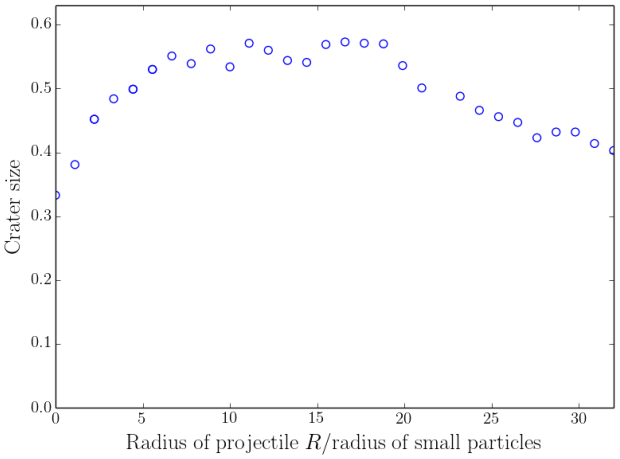
(a) Initial. (b) Final.

Figure 5: Example of the crater that forms when a projectile, with mass $M = 25m$, radius $R = 5r$ and vertical, downward speed v_0, hits a wall of 1000 stationary particles, with mass m and radius r, uniformly distributed in the lower half of the square box with a packing fraction of $\frac{1}{2}$.

Intuitively one would expect that the projectile would have increasing impact with increasing mass since it then would deliver more momentum to the particles it hits. As seen in figure 6(a) it is a linear increasing relationship between



(a) Parameter scan across the mass M of the projectile.



(b) Parameter scan across the radius r of the projectile.

Figure 6: Size of crater as a function of different parameters. One large and heavy particle with velocity in the downward y-direction hits a wall of particles, with packing fraction $\frac{1}{2}$ in the lower half of the square box. The size of the crater is measured by the fraction of particles affected by the projectile impact.

crater size and mass of projectile until the mass gets around 25 times bigger than the small particles mass. The flattening out of the graph could be because the particle forms a crater all the way down to the horizontal wall and then the small particles cannot move any longer in the negative y-direction. If there wasn't any walls, the linearity would probably have continued for heavier particles as well.

The effect of an increasing radius is harder to imagine. As seen in figure 6(b) the crater size increases until the projectile has a radius 7 times larger than the small particles, then the increasing radius does not make a significant difference until the projectile has a radius 20 times larger than the small particles. Then the crater size start to increase with radius. This could be because the greater the radius of the projectile, the faster the projectile would be stopped since it would collide with more particles on its path in the negative y-direction.

4. Conclusion

The event-driven simulation avoids unphysical hard-disk overlaps in any case due to the adaptable time step. There is no numerical integration error, because the trajectories of the particles are evaluated to the full precision of the computer. But due to the assumptions that all collisions are instantaneous, difficulties with inelastic collapse may occur for $\xi < 1$, where the number of collisions per time goes to infinity.

5. Acknowledgements

I have not collaborated with anyone during this final exam in TFY4235 Computational Physics.

6. References

[1] W. Kauzmann. *Kinetic Theory of Gases.* Dover Publications, Inc., 1966.

[2] T. Nordam. *Computational Physics (TFY4235/FY8904): Final exam.* Norwegian University of Science and Technology, 2016.

[3] P. A. Tipler, G. Mosca, and Freeman. *Physics for Scientists and Engineers - with Modern Physics.* NTNU, 6 edition, 2008.