Efficiency of an analytical propagator with collision detection for Keplerian systems

Dylan Aliberti

To obtain the degree of Bachelor of Science at the Delft University of Technology.
To be defended on Friday 2 september 2022.

Bachelor Thesis
Technical University of Delft
Student number: 5170036

Delft, 29 August 2022
Supervisors: Dr. P.M. Visser
Prof. dr. J.M. Thijssen

This thesis is also available digitally at `http://repository.tudelft.nl/`.

Abstract

The aim of this thesis was to test the efficiency in practice of an analytical propagator with collision detection for N-body Keplerian systems. This can be used to simulate the evolution of a protoplanetary disk, which gives insight into how planetary systems form. The analytic propagator calculates collisions one by one, while a numerical propagator would compute each time step. The idea of using the analytic propagator is that collisions are rare in astronomical scales, such that jumping from collision to collision and calculating it, is more efficient than calculating all the time steps that are between collisions. Simplifying the orbits of the planetesimals into perfect Keplerian orbits, analytical solutions exist which are used by the analytic propagator.

In this thesis, the runtimes of simulations were measured as well as other properties directly related to the runtime. The overall efficiency of the algorithm with respect to N seemed to be $O(N^3)$, which is one power less than previously predicted. The prediction was that the runtime of the full simulation is $O(N^2\epsilon + \frac{N^4 s^3}{Ia^3})$. Here ϵ is the maximum eccentricity, s/a is the ratio of a planetesimal's radius to the semi-major axis of its orbit, and I is the maximum inclination. This was calculated by estimating the total number of collisions to be $O(N^2 s^2/Ia^2)$ and the runtime for each collision to be $O(N^2 s/a)$. But the number of collisions turns from quadratic to linear in N, implying that above a certain N almost all planetesimals collide, which reduces the power of N by one. For comparison, the octree code has an algorithmic efficiency of $O(N \log N)$ per time step, and the number of steps for a fixed integration time grows as $O(N^{4/3} \log N)$.

Table of contents

List of all symbols and units

See table 1 for units/constants, and table 2 for symbols. In this report a set of astronomical units is used (table 1) rather than SI units. This way the accuracy of floating point numbers can be improved. Floating point numbers lose accuracy in some cases when working with very large or very small numbers, especially with both of them at the same time.

Notation conventions

Vectors will be typed in bold letters.
For a vector $\boldsymbol{v}$, $\hat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{|\boldsymbol{v}|}$ denotes the normalized vector.
$\boldsymbol{v} \cdot \boldsymbol{w}$ denotes the dot product.
$\boldsymbol{v} \times \boldsymbol{w}$ denotes the cross product.

symbol	meaning	value in SI units	value in astronomical unit system
AU	astronomical unit	$1.49598 \cdot 10^{11}$ m	1 AU
$M_\odot$	mass of the Sun	$1.9884 \cdot 10^{30}$ kg	1 $M_\odot$
y	year	31536000 s	1 y
G	gravitational constant	$6.67384 \cdot 10^{-11}$ $m^3 kg^{-1} s^{-2}$	39.42 $AU^3 M_\odot^{-1} y^{-2}$
$S_\odot$	radius of the Sun	$6.963 \cdot 10^8$ m	0.00465 AU

Table 1: Used units and constants. All SI values and conversion factors except for year are taken from Binas [2] and manually converted.

symbol	quantity	unit
a	semi-major axis	AU
b	semi-minor axis	AU
$c = a\epsilon$	semi-focal separation	AU
l	semi-latus rectum	AU
$\boldsymbol{L}$	angular momentum	$\mathrm{M}_\odot\mathrm{AU}^2\mathrm{y}^{-1}$
$\boldsymbol{\epsilon}$	eccentricity vector	-
$\hat{\boldsymbol{\mu}}$	$\hat{\boldsymbol{L}} \times \hat{\boldsymbol{\epsilon}}$	-
Ω	ascending node	rad
I	inclination	rad
ϖ	argument of periapsis	rad
ν	true anomaly	rad
E	eccentric anomaly	rad
$\mathcal{R}$	rotation matrix	-
t	time	y
	runtime	s
T_{sim}	maximum simulation time	y
V	volume	AU^3
$\boldsymbol{r}$	position vector	AU (three numbers)
r	radial distance	AU
$\langle r \rangle$	time-average radial distance	AU
S	radius of central body	AU
M	mass of central body	$\mathrm{M}_\odot$
N	number of particles	-
m	mass of planetesimal	$\mathrm{M}_\odot$
s	radius of planetesimal	AU

Table 2: All symbols used. The units are included in the astronomically defined system

1 Introduction

Thanks to modern technology, we know a lot about the Solar system, and a lot of objects are tracked continuously. To give an idea, over a million objects are known orbiting the Sun [4]. However, the way the Solar system was formed is still a big topic of debate. To gain insight in formation of solar systems, simulating the formation may help. Such a simulation would start with a protoplanetary disk orbiting the central body, with a lot of planetesimals colliding and merging into bigger objects.

Including collision detection in a simulation increases the time needed to compute it. Even just simulating different objects attracting each other already gets computationally expensive quite fast, because for N bodies attracting each other, the number of pairs for which the forces need to be calculated scale with N^2. Some clever algorithms exist that group objects together in bigger chunks and calculate gravity between chunks, for example the Octree code [5]. This greatly reduces the number of pairs that need to be calculated, while still maintaining a reasonable accuracy. The algorithmic efficiency per time step of this code is $O(N \log N)$. Collision detection makes the computation harder, since all objects need to be checked with the nearest neighbours for any collisions. For a fixed integration time, the number of steps grows as $O(N^{4/3} \log N)$ when including collision detection [9].

In this thesis, an analytic propagator is used instead of a numerical one. In this thesis, Visser's algorithm for collision detection in N-body Kepler systems [9] is tested. The analytic propagator calculates all collisions and jumps through these collisions one by one. In order to do this, the algorithm first goes through all pairs of planetesimals and makes a list of all possible collisions between them. For a protoplanetary disk orbiting a central body, this initialization process is estimated to be of order $O(N^2\epsilon)$ with ϵ being the maximum eccentricity. In the second phase, the algorithm jumps from collision to collision, applying them and updating the rest of the list with any changes caused by the collision. Because planetesimals generally have a very small radius compared to the semi-major axis of their orbits (we call this ratio s/a), collisions are very rare. On the contrary, the efficiency of the octree code is independent from s/a, since checking nearest neighbours happens independently from the radii of the planetesimals. The algorithmic efficiency of the analytical propagator is $O(N^2\epsilon + \frac{N^4 s^3}{Ia^3})$, where I is the maximum inclination in the protoplanetary disk. The order of N is higher than that of the octree code, but by s/a being small, the analytic propagator is still more efficient when the population of planetesimals is relatively sparse.

In the chapter 2, a summary of the analytical algorithm is given, as well as some properties of Kepler orbits necessary to understand it. In this paper this algorithm is tested in the context of a protoplanetary disk. The way this disk is randomly generated for the simulation is described in chapter 3. The runtime of the simulations is recorded, as well as other properties directly related to it. In the first part of chapter 4 part of the initialization is tested. In the second part, full simulations are experimented on. For the interested reader, the appendix includes methods for visualizing what is happening inside a simulation, which proved very useful for debugging and for presentation purposes.

1.1. Numerical efficiency

We are interested in the efficiency of Visser's algorithm. The main goal is therefore validating the order of efficiency of the whole algorithm. In order to get the big picture, we test different parts of the algorithm.

See table 1.1 for an overview of order estimations, copied from Visser's paper, including the notes.

Table 1.1: Algorithmic efficiency

algorithmic step	runtime $\propto O(\cdot)$	memory $\propto O(\cdot)$
create particle list	N	N
sort particle list	$N \log N$	1
create collision list	$N^2\epsilon$	$\frac{N^2 s}{a}$
sort collision list	$\frac{N^2 s}{a} \log \frac{N^2 s}{a}$	1
reduce particle list	$\log N$	0
create κ fragments	κ	κ
sort κ fragments	$\kappa \log \kappa$	1
merge particle lists	$N + \kappa$	1
reduce collision list	$\frac{Ns}{a} \log \frac{N^2 s}{a}$	0
create new collision list	κN	$\frac{\kappa N s}{a}$
merge collision lists	$\frac{N^2 s}{a} + \frac{\kappa N s}{a}$	1
total simulation	$N^2\epsilon + \frac{N^4 s^3}{I a^3}$	$\frac{N^2 s}{a} + \frac{N^4 s^3}{I a^3}$

Order estimations, in big O, of the time and memory requirement in Keplerian collision detection. The top shows the initialisation (N is initial particle count), the middle rows managing one collision, the bottom is for the full simulation (N is final particle count). This total is found because there are an expected $O(N^2 s^2 / I a^2)$ collisions in any fixed simulation time. For comparison the time complexity for tree codes scales as $O(N^{4/3} I^{-1/3} \log N)$. The number of fragments κ produced in a collision may be set to $\kappa = 1$ if collisions only result in mergers and/or scattering. The particle list and the collision list have respective lengths $O(N)$ and $O(N^2 s/a)$. This results in the estimates for the given running times for the reductions of the particle list and the collision list.

2

Theory

2.1. Kepler orbits

In this section some properties of Keplerian orbits are listed that will be used later on in this report.

Kepler coordinates

An elliptical orbit can be described by a set of six Kepler coordinates: The semi-major axis a, eccentricity ϵ, argument of periapsis ϖ, inclination I, ascending node Ω and the true anomaly ν [6]. The semi-major axis and eccentricity define the size and shape of the orbit. For an ellipse, the semi-major axis is half its length, and the eccentricity is a number indicating how stretched the ellipse is, or more specifically, the focal distance divided by the semi-major axis. $\epsilon \geq 1$ results in escape orbits (parabolas and hyperbolas). The meaning of the first three angles is illustrated in figure 2.1. The true anomaly is the angle between the periapsis and where the object currently is.

Eccentricity vector

An alternative way of describing the rotation of an orbit is by a vector instead of three angles. To do this, we can use the so-called Laplace-Runge-Lenz vector $\boldsymbol{A}$, defined by

$$\boldsymbol{A} = m\boldsymbol{v} \times \boldsymbol{L} - GMm\hat{\boldsymbol{r}}. \tag{2.1}$$

This vector is a conserved quantity for any central force following the inverse square law [3], which is the case for gravity. This vector points from the central body in the direction of the periapsis. Instead of the Laplace-Runge-Lenz vector, in this thesis we work with the eccentricity vector. This points in the same direction, but its length is defined to be the eccentricity of the orbit. This is useful for certain calculations. Along with the orbital angular momentum vector $\boldsymbol{L}$, this vector fully defines the orbit [3]. The vectors even have one number more than is strictly necessary to define an orbit, but using these vectors is useful for calculations.

Rotation Matrix for Kepler orbits

A Keplerian orbit can be expressed as a rotation matrix acting on an orbit in the reference plane, with it's periapsis on the positive x-axis. The rotation matrix is expressed by the following three standard rotation matrices [6]

$$\mathcal{R} = \begin{bmatrix} \cos\Omega & -\sin\Omega & 0 \\ \sin\Omega & \cos\Omega & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos I & -\sin I \\ 0 & \sin I & \cos I \end{bmatrix} \begin{bmatrix} \cos\varpi & -\sin\varpi & 0 \\ \sin\varpi & \cos\varpi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.2}$$

Polar formula for conics

Let r and θ be the polar distance and angle respectively, l the semi-latus rectum and ϵ the eccentricity. Then conics (circles, ellipses, parabolas and hyperbolas) can be expressed using the formula [1]

$$r = \frac{l}{1 + \epsilon\cos\theta} \tag{2.3}$$

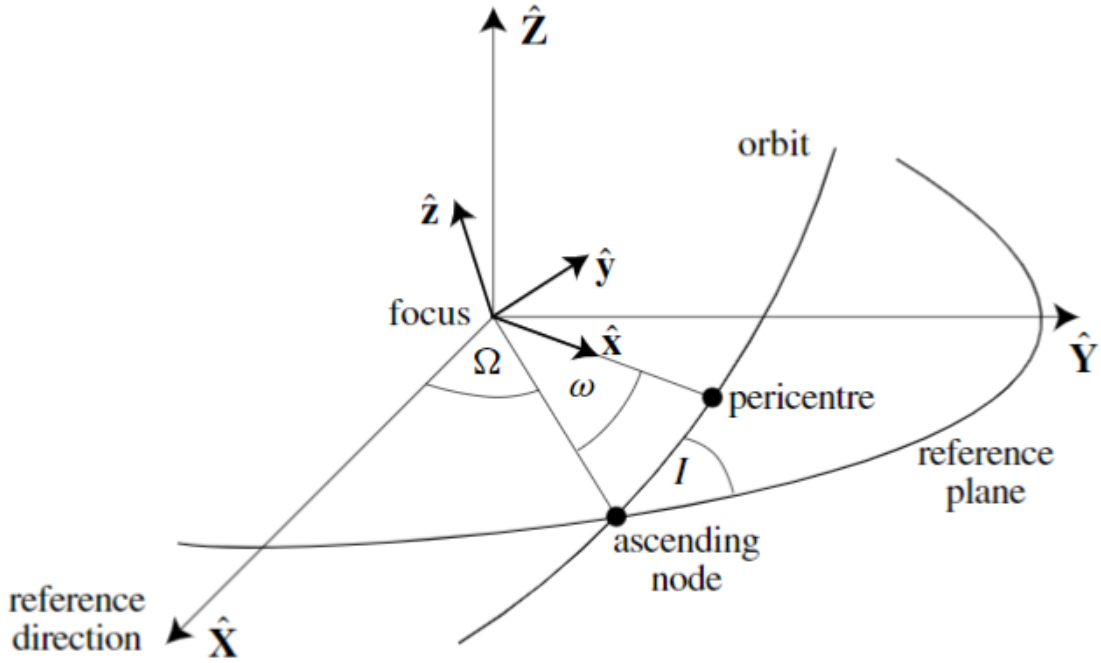

Figure 2.1: Illustration of the meaning of the orbital angles.
Source: Solar System Dynamics [6]

For parabolas this equation has an asymptote at $\theta = \pi$, and hyperbolas have two asymptotes at $\pm \arccos \frac{-1}{\epsilon}$. The branch physically corresponding to actual hyperbolic orbits of particles lies within $-\arccos \frac{-1}{\epsilon} < \theta < \arccos \frac{-1}{\epsilon}$. See figure 2.2 for a visualization.
Keplerian orbits have the shape of such conics, and the semi-latus rectum l for such orbits corresponds physically with orbital angular momentum [1]. Let L^2 be the square magnitude of the orbital angular momentum, G the gravitational constant, M the mass of the central body and m the mass of the particle. Then the semi-latus rectum is calculated by

$$l = \frac{L^2}{GMm^2}. \tag{2.4}$$

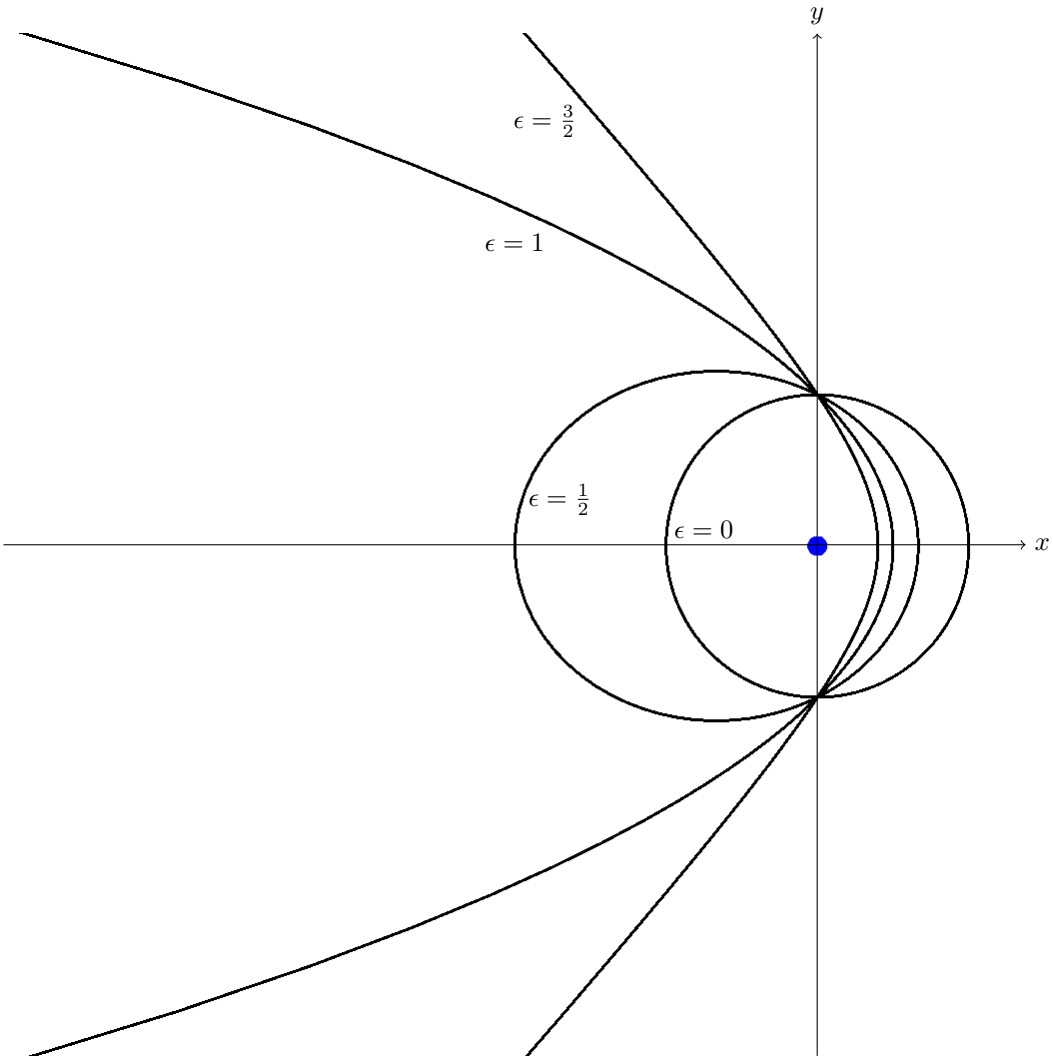

Figure 2.2: Plot showing four different Kelperian orbits, each with the same orbital momentum, and increasing eccentricities from 0 to 1.5 in steps of 0.5. This results in respectively a circle, ellipse, parabola and hyperbola. The blue dot represents the central body. This plot was made using the procedures described in chapter A.1, as a (successful) test. The central body is not to scale.

2.2. Summary of the analytical propagator for a Keplerian system

This report contains experiments done on Visser's algorithm for collision detection for N-body Kepler systems. In this section we summarize the key points of the algorithm. Please see Visser's paper [9] for the full details.

2.2.1. The main idea

Contrary to numerically integrating a set of equations of motion with a fixed time step, this algorithm solves the (unperturbed) motion of the planetesimals analytically. In order to do that, the assumption is made that all planetesimals follow a perfect Kepler orbit around the central body. Sorted lists of all planetesimals and potential collisions are maintained. The main loop of the algorithm applies the earliest collision in the list and updates the planetesimal and collision list accordingly. This way in each step, the algorithm jumps to the next collision. Collisions between planetesimals are very rare since the ratio of their radius to the semi-major axis of their orbit, which we refer to as $\frac{s}{a}$, is very small. This way, this analytic propagator can have larger time steps compared to numerical ones. In the limiting case where the planetesimal radius goes to zero, the algorithm stops after initialization.

2.2.2. Initialization

We have a list of planetesimals indexed $j \in \{1, ..., N\}$, of which we store the following variables:

$$\{t_j^0, \quad a_j, \quad c_j, \quad s_j, \quad m_j, \quad \boldsymbol{r}_j^0, \quad \boldsymbol{L}_j, \quad \boldsymbol{\epsilon}_j, \quad \omega_j\}. \tag{2.5}$$

Here, t^0 is the time of creation, a and c are the orbital radius and focal distance, s is the planetesimal radius, m is the planetesimal mass, $\boldsymbol{r}^0$ its position at t^0, $\boldsymbol{L}$ the orbital angular momentum vector, $\boldsymbol{\epsilon}$ the eccentricity vector, ω the orbital frequency.

The planetesimals are randomly generated within parameters in order to form an astrophysical disk of near-circular orbits. More on that in the chapter 3, experimental method. After that, the planetesimal list is sorted by increasing value of periapsis, so that we can use sweep and prune. That means that, when considering pairs of planetesimals for a possible collision, we only look at pairs whose orbits overlap, in the sense that the apoapsis + radius of the inner planetesimal exceeds the periapsis − radius of the outer one, because else the planetesimals will trivially never collide. Now, using the sweep and prune, we consider all overlapping pairs of planetesimals and compute a list of all possible collisions, which is sorted, so that the earliest collision is always on top. The way possible collisions are computed will be explained in its own subsection 2.2.4.

2.2.3. Main loop

For the main loop I will quote the steps from Visser's method and add comments to explain the steps:

1. If the collision list is empty, end the simulation.
2. Take the pair (i, j) with the soonest collision: the first in the list.
   A pair of particles i and j is meant. The collision list contains pairs that are on collision course and the time at which they would collide.
3. Update the time t to the time $t^1_{(i,j)}$ of the collision.
4. Remove from the pairs list any pair containing i and any pair containing j.
   Since the particles collide, they don't exist anymore afterwards (they are replaced with a larger particle that is the result of them merging). Hence, any future collisions involving those 2 particles will not happen anymore.
5. remove from the particle list the particles i and j.
6. If the orbit of the new particle intersects the central mass or is unbound, go to the next collision on the list.
   In other words, if the new particle does not get a stable orbit, no further calculations are needed since the new particle is not staying, and we move on to the next collision (next iteration of

the main loop). Note that the approximation is made that a particle with an orbit escaping or intersecting the central body would not collide with other particles. Although in theory it can collide, such collisions are extremely rare to the point that the statistics are not influenced by this approximation. If such precision is needed, the algorithm can be extended with analytic solutions for hyperbolic orbits intersecting either other hyperbolic orbits or elliptical ones.

7. Create new particle(s) defined by $\{t^1_{(i,j)}, a, c, s, m, \boldsymbol{r}^0, \boldsymbol{L}, \boldsymbol{\epsilon}, \omega\}$.
   Note that the particles possibly being multiple ones is only for models that consider the particles breaking up in fragments or when considering gravitational scattering. For this report we simply merge the particles, always resulting in 1 particle. The values of the new particle are calculated by applying conservation principles.

8. For any new particle, consider the other particles and decide if the pair is on collision course. If this is the case, calculate the time of the earliest collision. This step is further broken down in its own subsection 2.2.4.

9. Make a sorted list of the new collision possibilities with a record of the collision time and the pair, soonest collision first.
   With new particles come new possible collisions which need to be calculated, and are included into the collision list in the next step.

10. Merge this sorted list with existing sorted list of collision possibilities into a full list of pairs, sorted by time of the collision event, soonest collision first.
    Contrary to sorting an arbitrary list, merging 2 sorted lists is a linear operation with respect to number of items, and since the new collision list is a small list, sorting it is also a small operation.

2.2.4. Calculating the collisions

Before proceeding, recall that this calculation only has to be done on the pairs that come through the sweep-and-prune, which makes a lot of difference for small eccentricities.

1. The Minimum Orbit Intersection Distance (MOID) is calculated. In order to do this, the two nodes are computed where the orbital planes intersect each other, see figure 2.3. The movement at the planetesimals is then linearized at those nodes, allowing us to compute the closest distance. Since this uses an linearization, it is an approximation of the MOID. This approximation fails when the difference in inclination of the orbits is very small. More specifically, this inclination needs to be larger than $(s_1 + s_2)/(a_1 + a_2)$ in order for the approximation to work. Since s/a is typically very small for planetesimals in an astrophysical disk (e.g. 10^{-5} for objects like Earth), we are safe. If this MOID is smaller than the sum of radii of the two planetesimals, they come close enough to be able to collide, so we need to perform the next step.

2. The collision time is calculated. In order to do this, first Calculate the earliest passage times. Both planetesimals will be exactly at the potential collision position at a certain time, modulo the orbital period. From both planetesimals the earliest such time is calculated. This still has to be done at both nodes. Now, we need to find the smallest integers k and l such that if planetesimal 1 completes k orbits and planetesimal 2 l orbits, that they are at the collision place simultaneously. This can be done efficiently making use of continued fractions. It is a bit complicated, but I encourage you to read the thesis of Aron Schouten [8]. It contains a good explanation of it, backed up by test results.

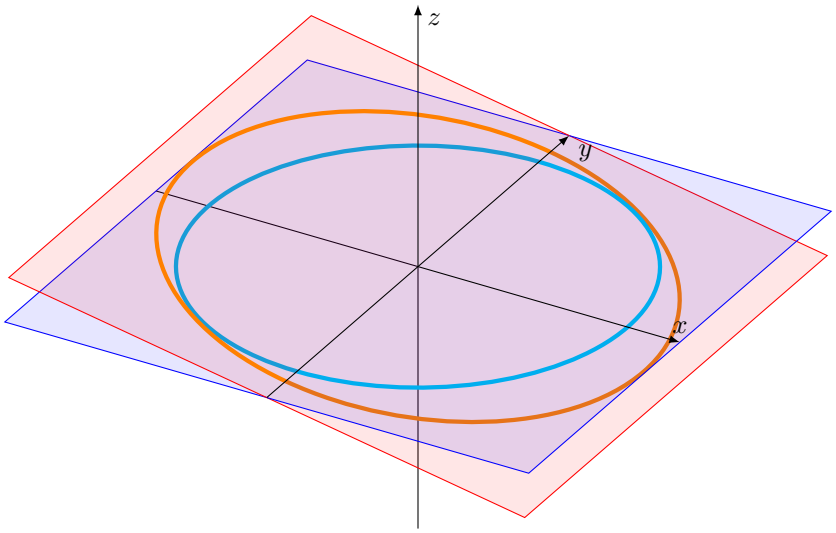

Figure 2.3: Two circular orbits with inclinations of 0 and 10 degrees. The nodal line is the line where the orbital planes intersect each other, which is in this case the y-axis. For nonzero difference in inclination, the nodal line is well-defined. Since planetesimals are small compared to their orbit, collisions must happen close to the nodal line, because the orbital planes are too far separated far from the nodal line.

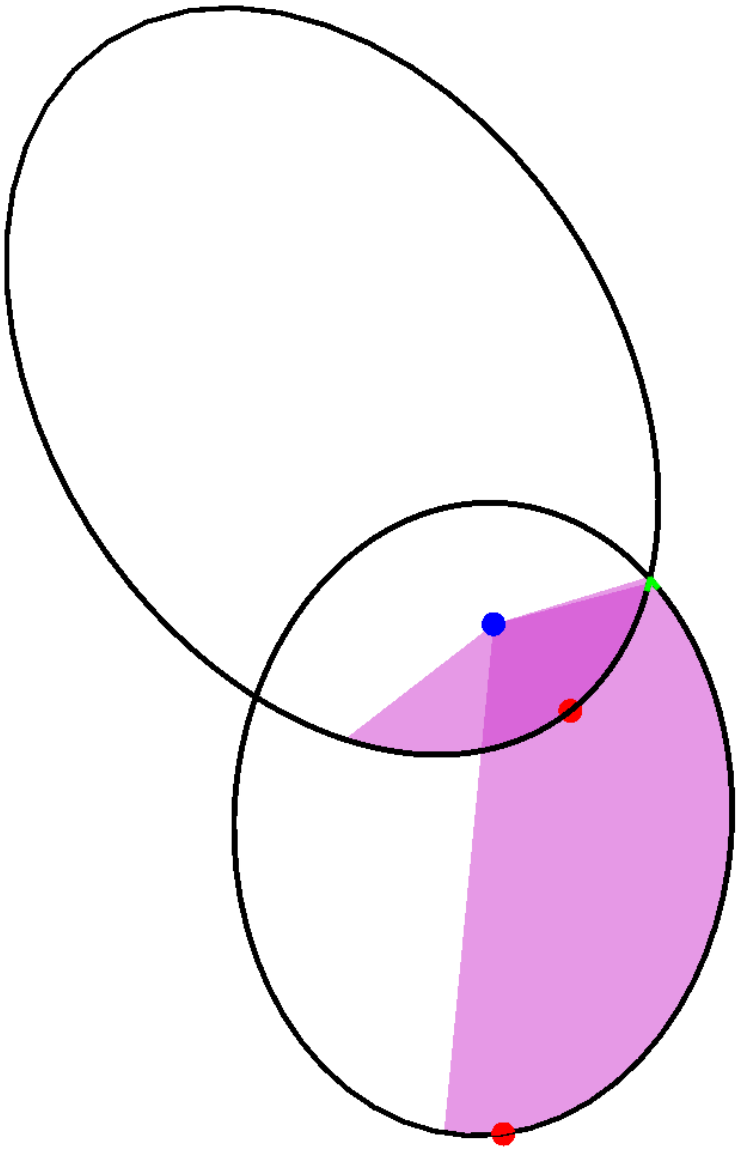

Figure 2.4: Image obtained using the visualization methods from the appendix A.1. This is a test with two very eccentric orbits, that have a MOID close enough for collision on the right. Note that the simulations in the experiments done in this thesis do not have such high eccentricities. This is solely for demonstration. The purple sections represent orbital sections going from the planetesimals initial position to the place of possible collision. Small green arrows are visible near the place of collision. Figure 2.5 zooms in on the place of collision.

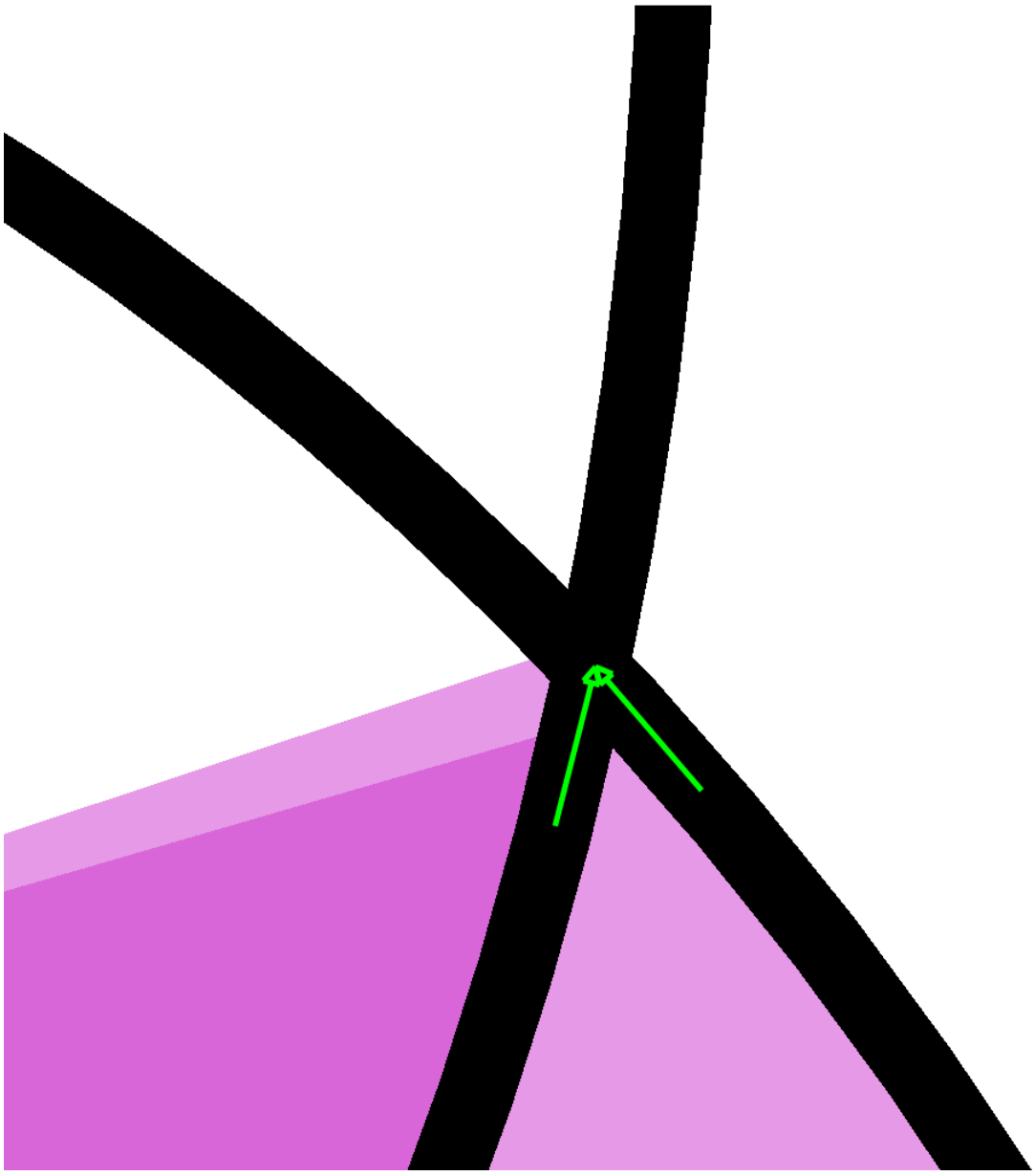

Figure 2.5: The place of collision between two very eccentric orbits in a test simulation. This is a zoomed in version of figure 2.4. Here, the green arrows can be clearly seen. They point from the orbital nodes to the place of collision. It can be seen that in this case the nodal linearization works well.

3

Experimental Method

3.1. Creating a random disk of particles

In order to test the algorithm, we want to create a random disk of particles orbiting the central body, resembling a proto-planetary disk. Before moving on, we note that the computer can generate random numbers uniformly distributed between 0 and 1. Let's call this random variable ξ. ξ can be generated by built-in pseudo-random number generators. We generate a seed for the pseudo-random number generator based on the current system time, so that each time a seed is generated the results are different from last time. We also include this seed when saving data from the simulation, so that the simulation can be reproduced if further inspection is needed.

3.1.1. Random rotations for the orbit

We choose the argument of periapsis ϖ and the ascending node ☊ uniformly between 0 and 2π. That is, their values are calculated with $2\pi\xi$ so that $\varpi \in [0, 2\pi)$. If the cosine of the inclination I is chosen uniformly between 0 and 1, the distribution of all angles together form a uniform sphere [7]. However, in this case we are making a disk, so let $I_{\max}$ be the maximum inclination in the disk. Then the minimum value for $\cos I$ is $\cos I_{\max}$, and maximum value is 1. Hence, $\cos I = \cos I_{\max} + (1 - \cos I_{\max})\xi$. By taking the arccosine we finally get an explicit formula for randomly generating I:

$$I = \arccos\Big(\cos I_{\max} + (1 - \cos I_{\max})\xi\Big). \tag{3.1}$$

3.1.2. Radial distribution

We want to make a uniform disk with a constant particle number density. Earlier (subsection 3.1.1) we have determined a way to choose random angles within a disk. Now let r be the distance between the particle and the central body. For circular orbits r is the radius of the orbit, but I will link r to elliptical orbits in subsections 3.1.3 and 3.1.4. Since we are applying the algorithm in 3D space, the volume of a sphere scales with a power of 3 with respect to its radius. Hence, we expect to find a cumulative distribution function for r that follows a power of 3.

Let's calculate the volume of a uniform disk, with as boundaries $I_{\max}$ and certain $r_{\min}$ and $r_{\min}$. This is a kind of torus with a circularly bent trapezium as its base shape, shown in figure 3.1. This is a crude model for real planetary disks, but the way we defined it allows us to compute the volume using spherical coordinates. Let $\hat{r} \in [0, \infty)$ be the distance to the origin, $\theta \in [0, 2\pi)$ be the azimuthal angle and $\phi \in [0, \pi]$ the altitude angle. For an inclination of zero, the vector pointing to the position is horizontal, meaning that $\phi = \frac{1}{2}\pi$. The lower and upper bounds of ϕ are then $\frac{1}{2}\pi \pm I_{\max}$. We compute the volume V of the disk in the same way we would for a sphere, but restricting ϕ:

$$V = \int_{\frac{1}{2}\pi - I_{\max}}^{\frac{1}{2}\pi + I_{\max}} \int_0^{2\pi} \int_{r_{\min}}^{r_{\max}} \hat{r}^2 \sin\phi \, d\hat{r} d\theta d\phi. \tag{3.2}$$

Computing the inner two integrals we get

$$V = \frac{2\pi}{3}(r_{\max}^3 - r_{\min}^3) \int_{\frac{1}{2}\pi - I_{\max}}^{\frac{1}{2}\pi + I_{\max}} \sin\phi d\phi, \tag{3.3}$$

where we can already see the cubical dependency of V on r.
Also evaluating the ϕ part, we finally get

$$V = \frac{4\pi}{3}(r_{\max}^3 - r_{\min}^3) \sin I_{\max}. \tag{3.4}$$

Note that, as a quick validation, substituting $I_{\max} = \frac{1}{2}\pi$ and $r_{\min} = 0$, returns the volume of a sphere.

We will now compute the cumulative distribution of r such that the distribution is uniform along the whole disk. In the case of uniformity, the chance of finding the random particle closer to the origin than r equals the ratio of the volumes:

$$P(r < R) = \frac{\frac{4\pi}{3}(R^3 - r_{\min}^3) \sin I_{\max}}{\frac{4\pi}{3}(r_{\max}^3 - r_{\min}^3) \sin I_{\max}} = \frac{R^3 - r_{\min}^3}{r_{\max}^3 - r_{\min}^3} \quad , \quad R \in [r_{\min}, r_{\max}]. \tag{3.5}$$

In order to randomly pick r, we write

$$\frac{R^3 - r_{\min}^3}{r_{\max}^3 - r_{\min}^3} = \xi, \tag{3.6}$$

which, by inverting, results in the following formula

$$r = \left((r_{\max}^3 - r_{\min}^3)\xi + r_{\min}^3\right)^{\frac{1}{3}}. \tag{3.7}$$

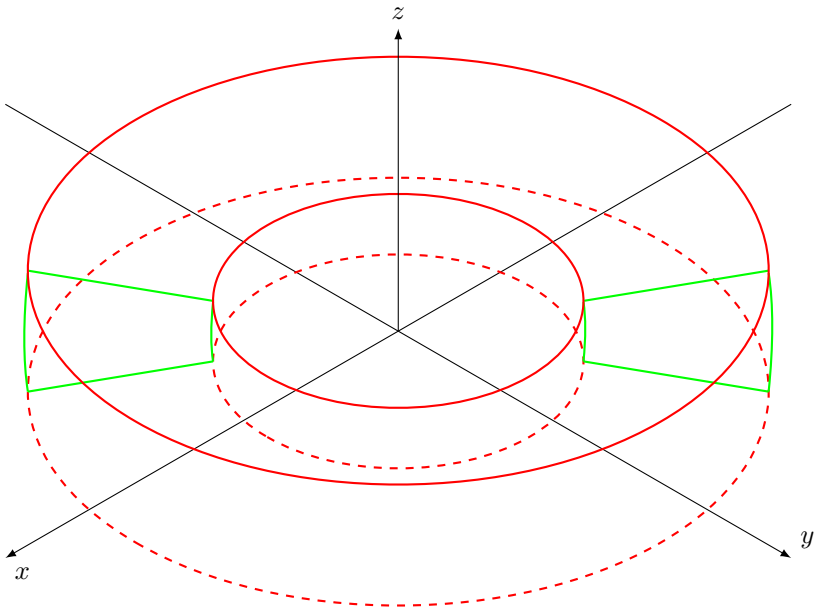

Figure 3.1: The disk. It is a kind of torus, with a circularly bent trapezium as its base shape. An intersection through the middle of the disk is shown in green, where we can see the bent trapezium. The inner part is thinner than the outer part, due to the boundary being an angle, namely the maximum inclination.

3.1.3. Approximation for small eccentricities

For small eccentricities, the orbits are nearly circular. Hence, the semi-major axis can be treated as if it is a radius of a circular orbit. We proceed by picking a random radius using equation (3.7), which we set as the semi-major axis. We also pick a random eccentricity $\epsilon \in [0, \epsilon_{\max}]$ uniformly. Because the orbit is nearly circular, the true anomaly is approximately equal to the mean anomaly. Thus we can pick a true anomaly uniformly between $[0, 2\pi]$, resulting in the particle having approximately uniform chance of appearing anywhere on the orbit.

3.1.4. Generalization for large eccentricities

For high eccentricities, the circular approximation does not work anymore. Instead, we make an extra requirement that the disk should be uniform regardless of time. We define the time-averaged radius $\langle r \rangle$ as

$$\langle r \rangle = \frac{1}{T} \int_0^T r(t) dt, \tag{3.8}$$

where T denotes the orbital period and t the time. Now we use Kepler's equation [6]

$$t = \frac{E - \epsilon \sin E}{\omega} \tag{3.9}$$

for making a variable substitution of t by E, resulting in

$$\langle r \rangle = \frac{1}{2\pi} \int_0^{2\pi} r(E) \frac{1 - \epsilon \cos E}{\omega} dE. \tag{3.10}$$

For the next step we use the eccentric formula for r [6]

$$r(E) = a - a\epsilon \cos E, \tag{3.11}$$

where a is the semi-major axis. Substituting this into the integral yields

$$\langle r \rangle = \frac{a}{2\pi} \int_0^{2\pi} (1 - \epsilon \cos E)^2 dE, \tag{3.12}$$

which evaluates to

$$\langle r \rangle = a\Big(1 + \frac{1}{2}\epsilon^2\Big). \tag{3.13}$$

Having obtained a formula for the average radius $\langle r \rangle$, we can now pick our random orbital elements. We choose $\langle r \rangle$ randomly according to equation (3.7). We then pick $\epsilon \in [0, \epsilon_{\max}]$ uniformly. We then calculate a using a rewritten form of equation (3.13)

$$a = \frac{\langle r \rangle}{1 + \frac{1}{2}\epsilon^2}. \tag{3.14}$$

Furthermore, instead of randomly picking the true anomaly uniformly, we pick the mean anomaly uniformly between 0 and 2π, since we want the initial distribution to not have any time-dependency. In fact, if the true anomaly is chosen uniformly for elliptical orbits, particles are biased to be more in the parts of their orbits closer to the central body. The result is that, in the first few years, a lot of particles come from their periapsis outwards, creating a visible explosion-like effect if visualized. That is why we now uniformly pick the mean anomaly instead of the true anomaly. Using the fixed point iteration given by equations (A.12) and (A.13), we can then calculate the eccentric anomaly, through which a particle can be positioned. The fixed point iteration is a numerical approximation, but because it is used to randomly place particles, small errors do not make noticeable differences.

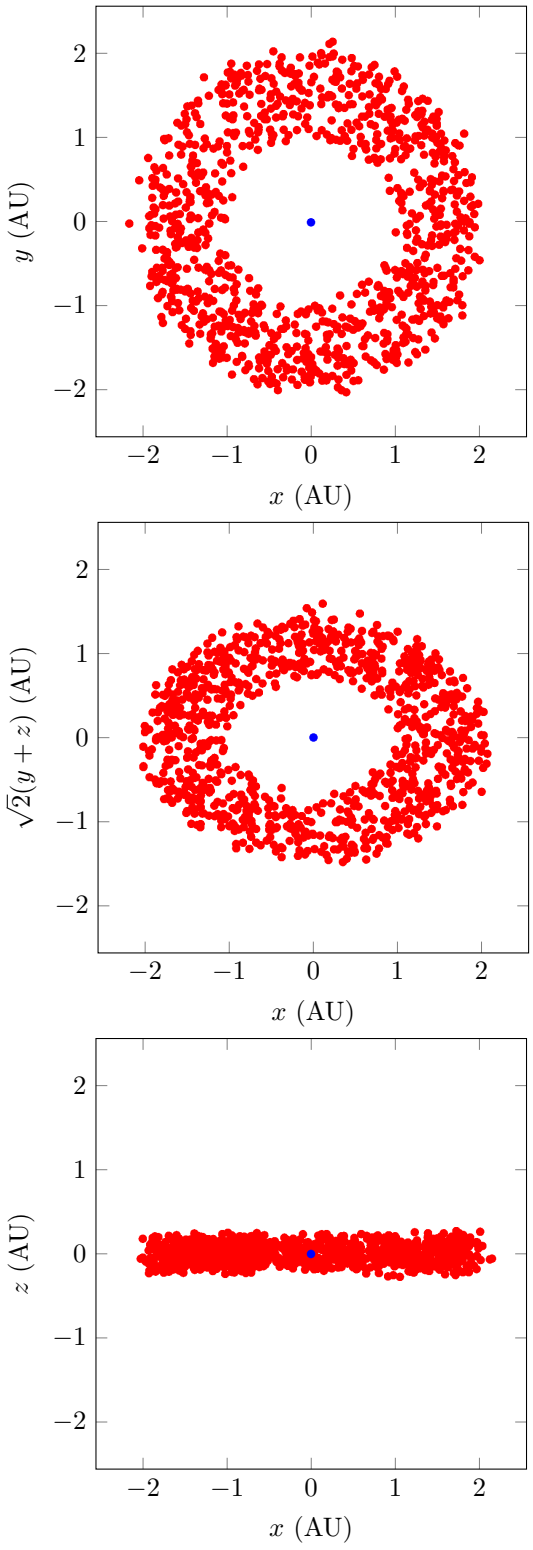

Figure 3.2: Orthogonal projections of a planetary disk randomly generated using the procedures described in section 3.1. From top to bottom: top side, diagonal view, and side view. The blue dot is the central body, and the red dots are the 1000 planetesimals orbiting the central body. The size of the planetesimals is not to scale (they would be invisibly small otherwise).

3.2. Simulation parameters

3.2.1. Pair test

One thing we are interested in is how many pairs of particles the algorithm has to check. In an N-body simulation, the number of all possible pairs is $\binom{N}{2} = \frac{N^2-N}{2}$. Each pair of particles has 2 intersection nodes, so the maximum possible number of nodes that the algorithm has to consider is $N^2 - N$. For testing the number of pairs, we omit the calculation of what time the calculations actually happen. This is because we are only testing the number of pairs that the algorithm has to consider, which makes calculating the time unnecessary. This way more data can be collected in the same time. The run time is still measured, but instead of the full initialization, it is the time it takes to calculate the pair nodes without the time at which collisions take place. We will refer to this procedure as the pair test. We are interested in the impact of N, max I, max ϵ and s/a, since they can directly impact the run time. We generate 4 sequences of 100 simulations (so 400 in total), one sequence for each variable. That variable is then randomized each of the 100 simulations. See table 3.1 for all parameters.

Parameter	Value	Value when randomized
S	0.00465 AU	
M	1 $M_\odot$	
N	10^4	$10^{6\xi}$ (pair test) $10^{5\xi}$ (full run test) $10^{4\xi}$ (full run increased s)
max I	10^{-3} rad	$\frac{\pi}{2}10^{-7\xi}$ rad
min a	1 AU	
max a	2 AU	
max ϵ	10^{-3}	$10^{-5\xi}$
m	10^{-6} $M_\odot$	
s	$2 \cdot 10^{-5}$ AU	$10^{-1-5\xi}$ AU
	$2 \cdot 10^{-3}$ AU (full run increased s)	
T_{sim}	10^6 years	

Table 3.1: Simulation parameters for all experiments. One variable is changed at a time, which gets a random value as shown in the right column, while all other variables are set to the value in the middle column.

3.2.2. Full run test

We want to know how long it takes for a full simulation to finish. The simulation first has an initialization phase where the initial collision list is calculated. After that the main loop starts running, where in each step a collision is applied and collisions with the new planetesimal are calculated. Hence we are interested in the run time for the initialization and for each step. The runtimes of the initialization and the steps in the second phase are recorded. We take 100 simulations where we take a random N each time, and run it fully. This will be referred to as the standard simulations. Besides the standard simulations we also compute another set of 100 simulations, this time with increased $s = 10^{-3} \cdot a_{\max}$. This idea came after looking at the standard simulations, because the results implied there is more to see on another scale. See chapter 4 for explanation.

For both sets of simulations, the following graphs are tracked:

- Runtime (initialization and phase 2) vs N
- Runtime per frame vs N
- Length of collision list throughout the simulation vs N
- Initial length of collision list vs N
- Number of collisions vs N
- Remaining number of planetesimals vs N

4

Results and discussion

4.1. Pair test

A total of four variables have been varied, which are the number of particles N, the maximum eccentricity ϵ_{max}, the maximum inclination I_{max} and the planetesimal radius s. For each of those, the number of checks, pair nodes and runtime have been measured, shown in figures 4.1 to 4.8. The label "checks" indicates the number of planetesimal pairs that got through the sweep-and-prune as a possible collision pair, hence the number of pairs that need to be checked by the algorithm. "pair nodes" indicates the number of nodes where the orbits of a pair of planetesimals gets close enough to collide, which are a maximum of 2 for each pair. Not all such nodes result in a collision, because the planetesimals might collide with other planetesimals sooner, or their orbital frequencies might have a ratio such that the collision time is outside the simulation time. Note that many of the figures contain dashed lines, which correspond to integer powers of the variable on the x-axis, in order to get a better picture of the shape of the graph.

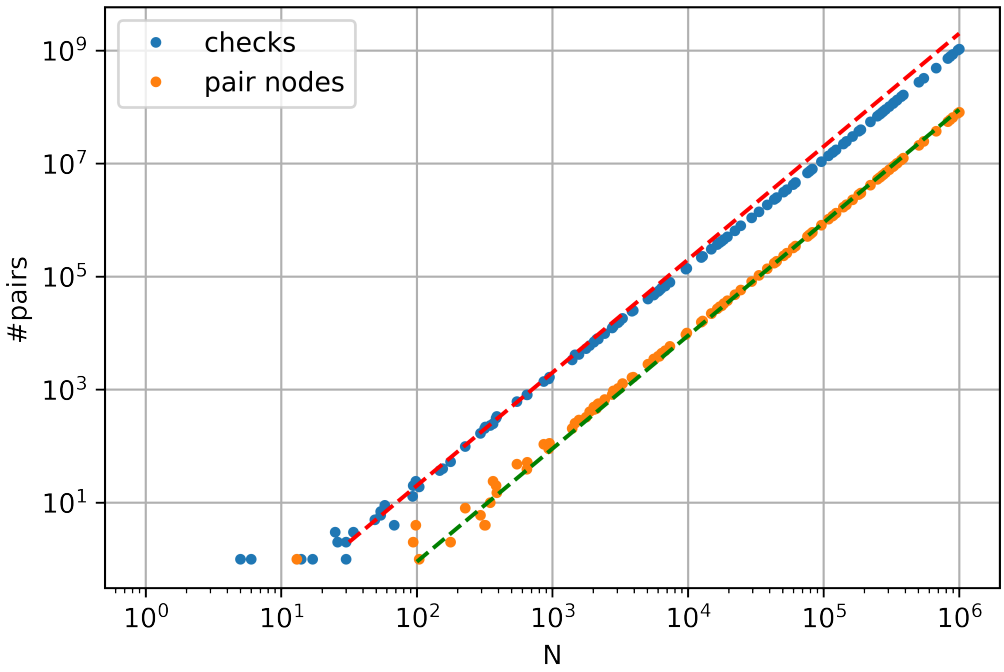

Figure 4.1: Checks and pairs vs number of particles N on a double log plot. The dashed lines have a slope of 2, matching the measurements. This indicates a quadratic dependency on N. This is the expected result, since the maximum possible number of pairs is also quadratic in N. Note that the sweep-and-prune does a great job. The maximum number of pairs is, in the double log plot, almost the line N^2, but the number of checks (the blue line) is a lot lower. For example at $N = 10^6$, the number of checks is 10^9, while the maximum would be almost 10^{12}. We also see that the number of pairs is less than the number of checks, even though its maximum value is double the number of checks (two nodes for each pair of planetesimals). This is expected, because the fact that two orbits overlap in the radial part does not mean that the orbits actually come close enough for collision.

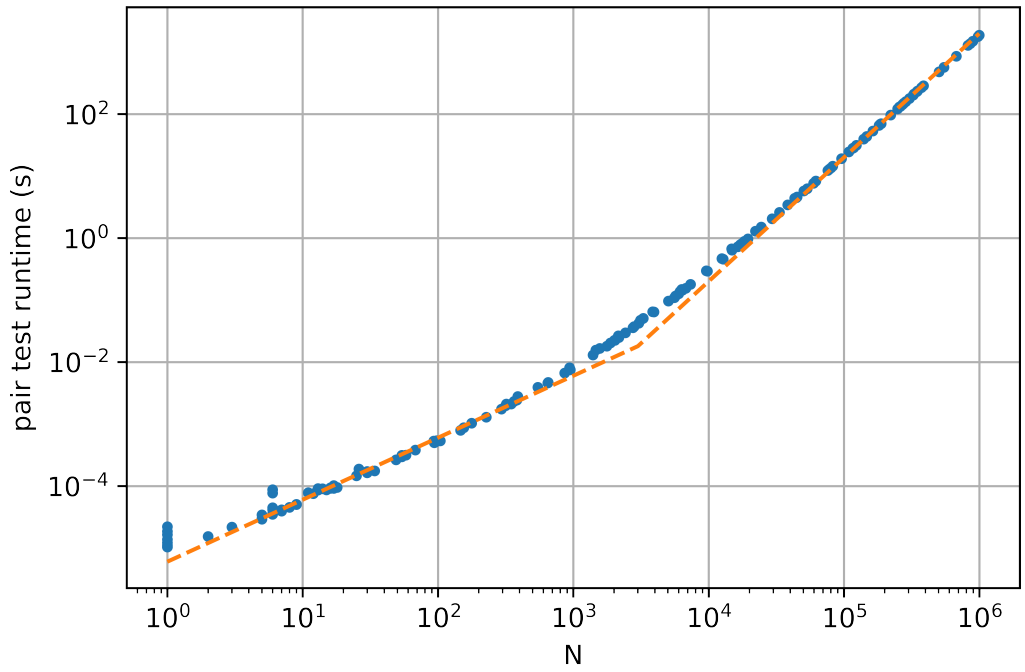

Figure 4.2: Pair test runtime vs number of particles N on a double log plot. The runtime shown here is the time it took for the computer to perform the pair test (initialization without calculating collision time). For low N the relationship is linear, but as N gets higher the graph becomes quadratic. This is illustrated by the dashed lines. The result is expected. That is because creating the particle list is $O(N)$ and sorting the particle list is $O(N \log N)$ which is almost linear in the scope of this graph. The number of pairs scales with $N^2(\frac{s}{a} + \epsilon)$, which starts becoming dominant in the region $N > \frac{1}{\frac{s}{a}+\epsilon}$. In this experiment $\epsilon = 10^{-3}$ and $\frac{s}{a} \ll \epsilon$, so we expect the quadratic dominance to start from order of magnitude 10^3. And in fact, this nicely agrees with the graph.

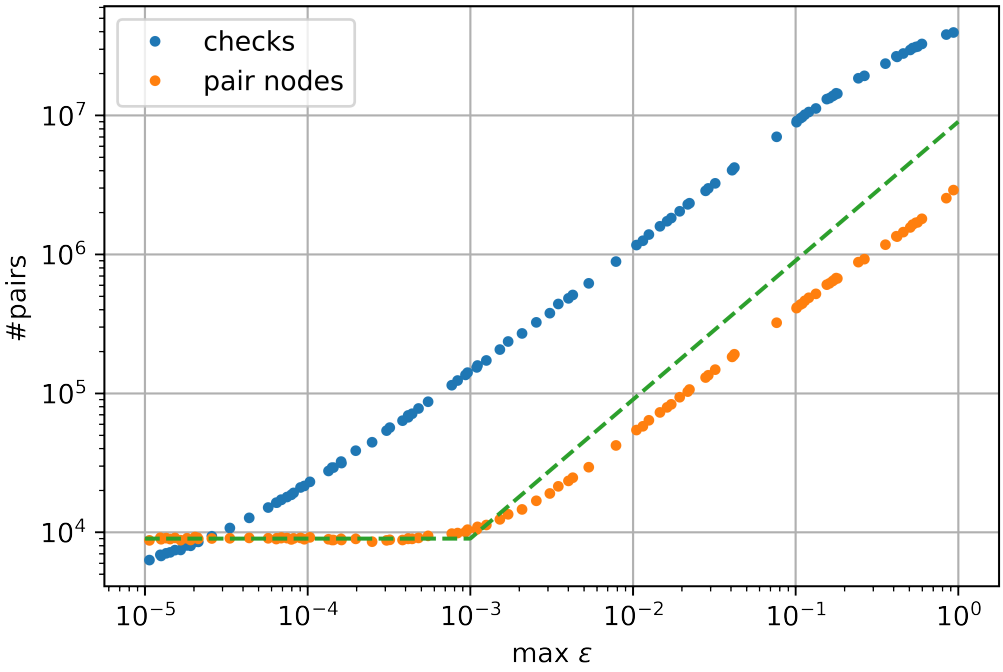

Figure 4.3: Checks and pairs vs maximum eccentricity on a double log plot. The number of checks scales linearly with ϵ, which is expected form the theory for $\frac{s}{a} \ll \epsilon$, which is the case here. It does show a tendency to flatten out at the sides of the graph. On the right side, the graph stops at $\epsilon = 1$. For eccentricities equal or greater than 1, the orbits are not closed anymore. Simulation of escape orbits were omitted because they would make negligible difference in the statistics. And in fact the maximum number of checks is $N^2 = 10^8$, which is nearly reached for ϵ close to 1, confirming that continuing with eccentricities > 1 cannot make significant difference. Indeed, with such high eccentricities actually all orbits overlap with each other and the sweep-and-prune becomes useless at decreasing the number of checks. For the lower end of the graph, we recall that the number of checks scales with $N^2(\frac{s}{a} + \epsilon)$, and as ϵ goes below 10^{-5} actually the $\frac{s}{a}$ becomes dominant, which was constant when varying ϵ, hence the curve flattens out below $\epsilon = 10^{-5}$.
For the number of pair nodes, a clear turning point is visible at $\epsilon = 10^{-3}$. The graph starts constant, and switches to an almost linear relation. The dashed line shows a constant and linear relation, and as we can see, the slope of the number of pair nodes is slightly less than 1. Note that at the left of the graph, the number of checks is slightly lower than the number of pair nodes, which is because each check results in a pair that can have up to two pair nodes. For ϵ to become dominant, we need $N^2\epsilon > N^2\frac{s}{a} + N$, thus $\epsilon > \frac{s}{a} + \frac{1}{N}$, which is at approximately 10^{-4}. This is one order of magnitude lower, which might be due to a different constant factor, although the turning point seems too high.

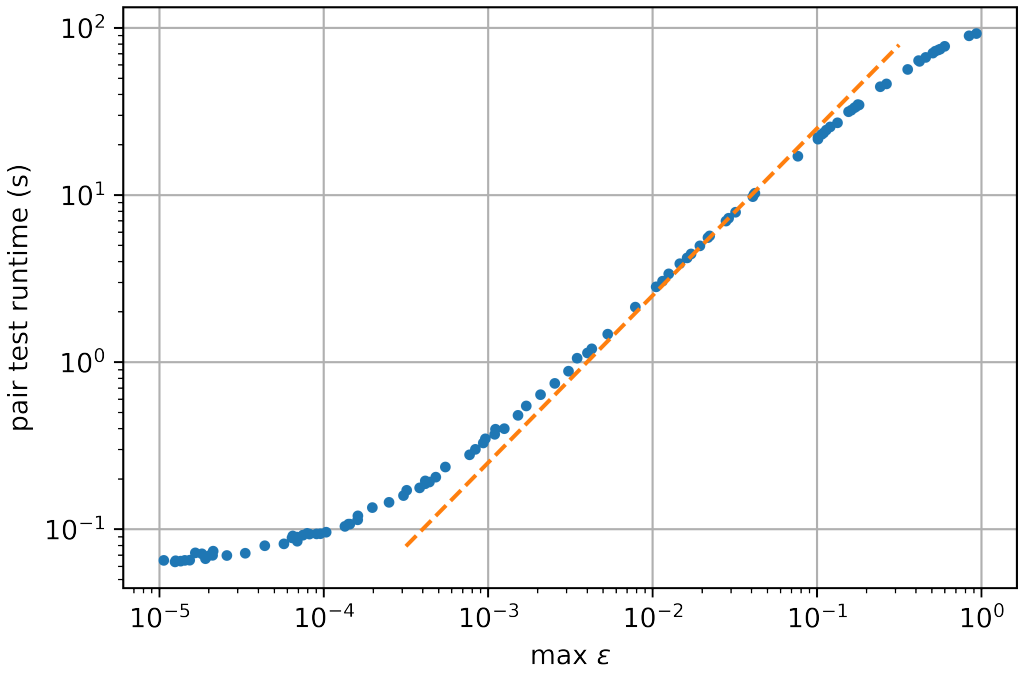

Figure 4.4: Pair test runtime vs maximum eccentricity $\epsilon_{\max}$ on a double log plot. The graph is roughly linear, illustrated by the dashed line, but flattens at the edges. This corresponds with figure 4.3, which showed a similar curve for the number of checks. For high eccentricities, the algorithm needs to do a lot of checks, even though only a small part ends up us a pair. Since most of the time is spent doing the checks, the graph of the runtime has approximately the same shape. Note however, that this curve seems more smooth than the checks curve in figure 4.3, which must be due to the part of the algorithm outside the loop that is taking up a small part of the runtime.

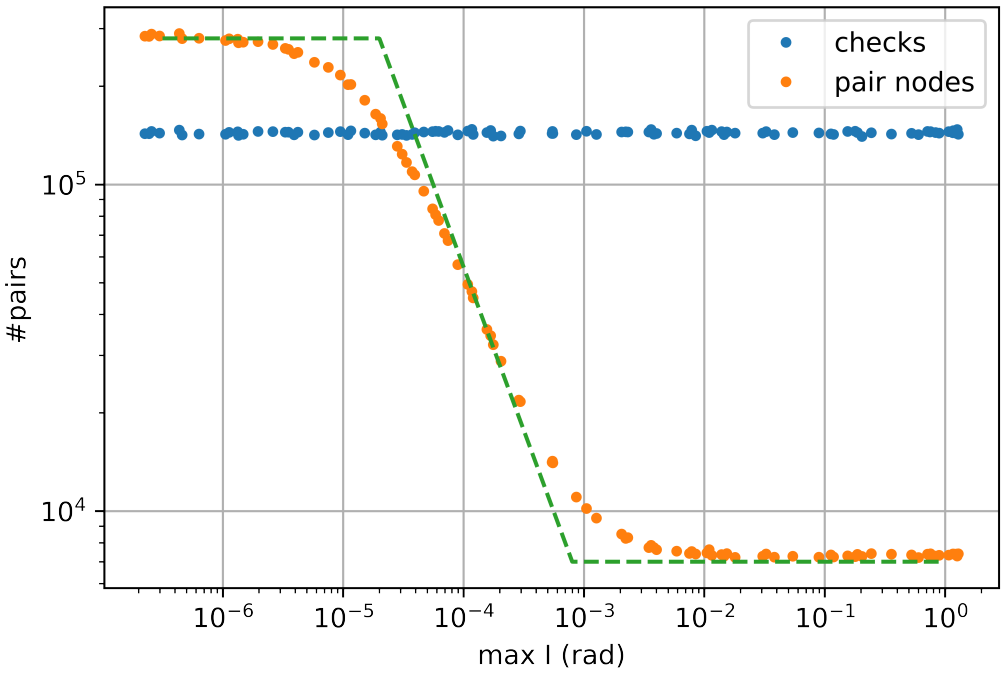

Figure 4.5: Checks and pairs vs maximum inclination on a double log plot. The number of checks is independent of inclination. The sweep-and-prune checks if 2 orbits overlap radially by comparing periapses, apoapses and radii of the planetesimals. The inclination is independent of those variables, as shown by the graph. For the number of pair nodes, 2 turning points are visible in the graph: 2 constant parts with a linearly decreasing part in the middle. An inclination below $s/a = 10^{-5}$ implies that the orbital planes are aligned so close, that the planetesimals may collide anywhere on the orbit. It follows that making the inclination any smaller would not add extra pairs anymore. Again, on the left of the graph the number of pair nodes is higher than the number of checks, because each checks results in up to two pair nodes. In the middle part of the graph, as I gets bigger than s/a, the number of pair nodes become less and less. The collisions can only happen closer and closer to the orbital nodes, reducing the probability of two planetesimals forming a pair. This goes on until the right turning point is reached around 10^3. The radius around the node at which the collisions can happen has a nonzero minimum, namely $s_1 + s_2$. Having such a minimum implies that the graph needs to flatten above a certain I, which is the case here, although it is not clear why that turning point is at 10^3.

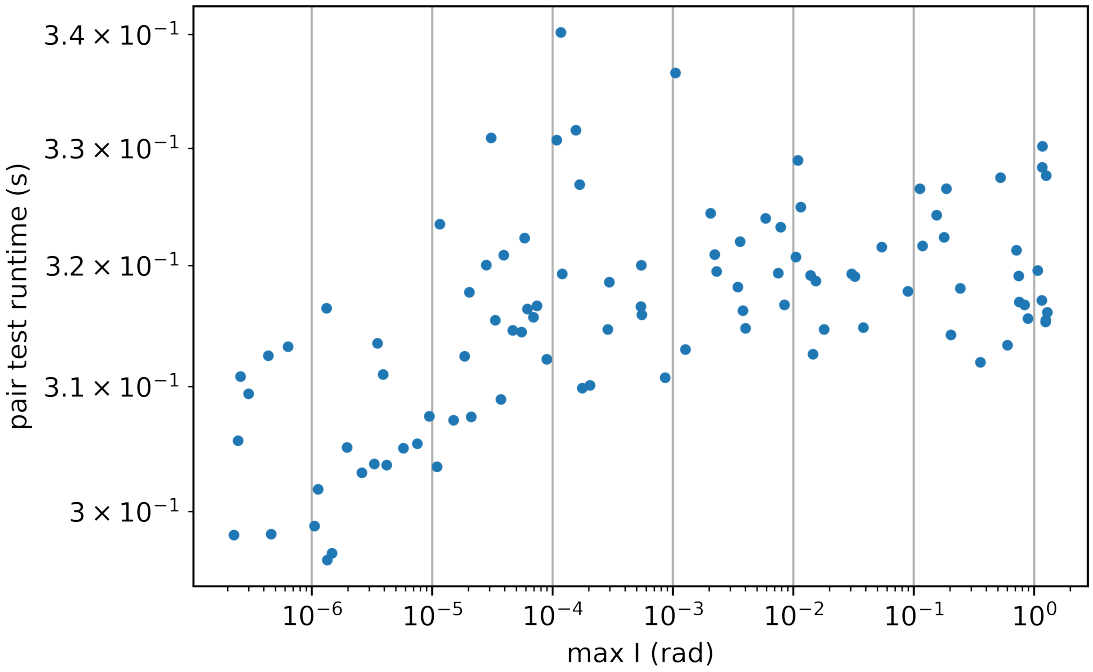

Figure 4.6: Runtime vs maximum inclination on a double log plot. The inclination does not seem to have any significant influence on the runtime. Since the number of checks is constant with respect to inclination, the only difference it makes is in the number of pairs. That explains why the impact of inclination on runtime is a lot smaller than the impact of the other tested variables since they also affect the number of checks.

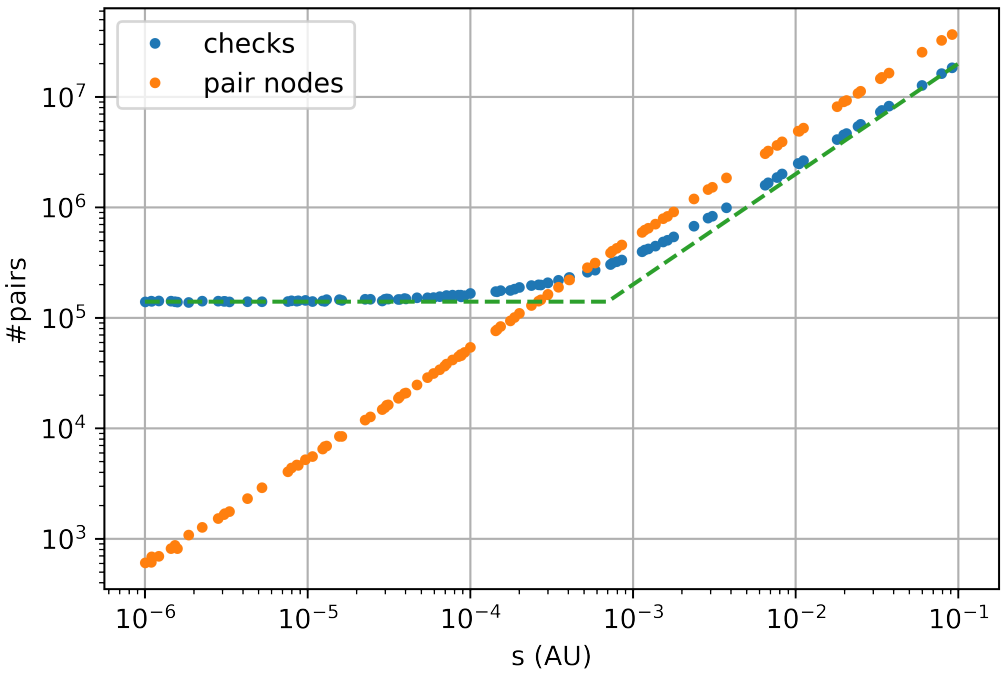

Figure 4.7: Checks and pairs vs radius of planetesimals on a double log plot. Note that $a_{\max}$ stays constant at 2 AU, while s is changed. The number of checks starts significantly and linearly depending on s from about $2 \cdot 10^{-3}$ AU, which is indicated by the dashed line. For smaller s the radii of the planetesimals do not dominate the sweep-and-prune comparisons, but when s gets bigger than ϵa, the radii of the planetesimals start making a difference in the sweep-and-prune. The actual pairs however still depend linear on s, since small planetesimals in general lead to fewer collisions. Once again, the number of pair nodes is slightly more than the number of checks, because each check results in up to two pair nodes.

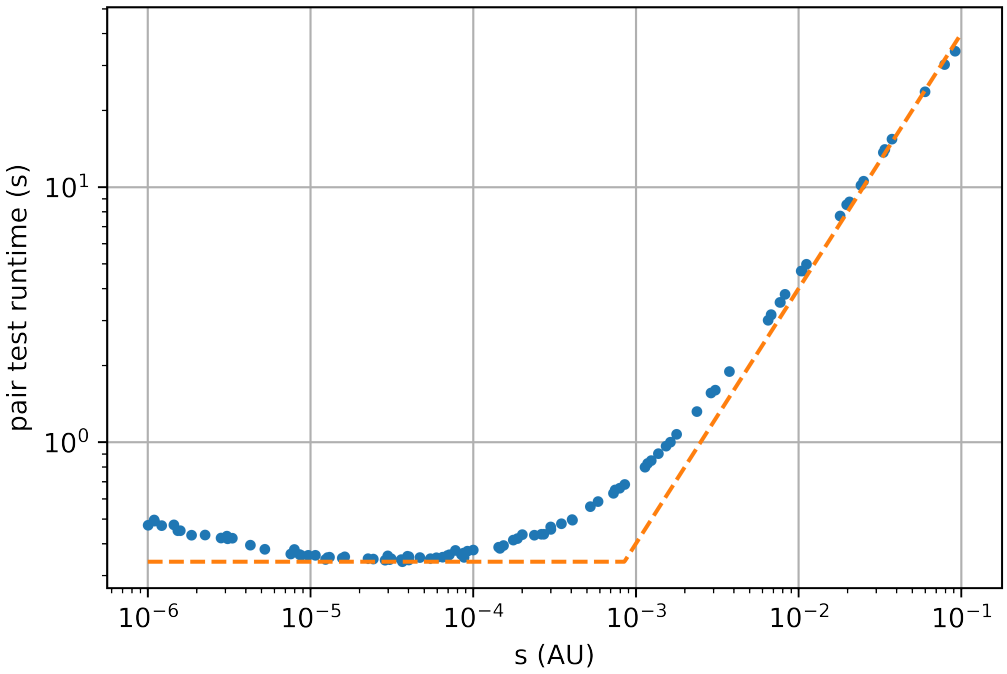

Figure 4.8: Runtime vs radius of planetesimals on a double log plot. For $s > 10^{-5}$ AU the graph resembles the number of checks from figure 4.7, which makes sense since much time of the algorithm initialization is spent in doing the checks. For lower s the runtime does not get lower anymore, indicating that some minimum time has been reached, which depends on other variables. The graph has a negative slope at low s. However, the expectation was that the slope would be non-negative everywhere. In a full initialization this could be due to the calculation of the collision time taking longer because of the low radii, but that cannot be it, since in the pair test the initialization is not fully done. The dashed line illustrates the linear part and a constant line for comparison.

4.2. Full run

The experiments in this section are about simulations running fully. The simulation has an initialization phase, and a second part which we will call "phase 2" here. Dashed lines of integer power relationships are included in some graphs for comparison. Simulations with two sets of parameters were tested, with the only difference being that s was increased to 10^{-3} in order to observe what happens in the region $N > a/s$. We will call the first simulations the standard ones, and refer to the other simulations as the ones with increased s.

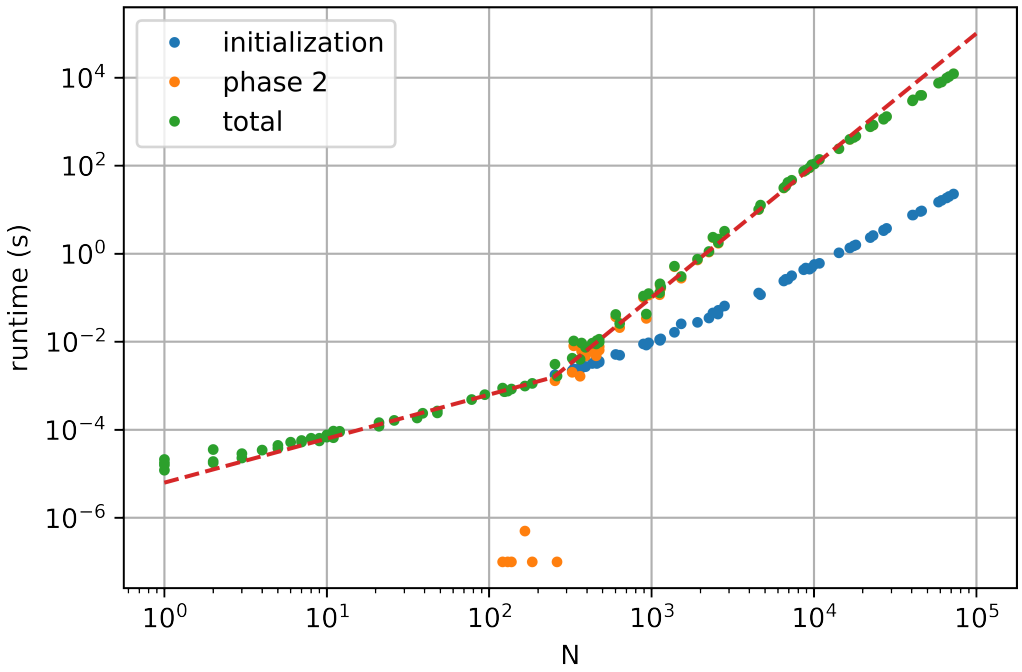

Figure 4.9: Runtime vs N for $s = 10^{-5} \cdot a_{\max}$ on a double log plot. Up to about $N = 3 \cdot 10^2$ the initialization dominates the total time, and then phase 2 takes over before the initialization gets to the quadratic region. As shown by the dashed line, this first part is linear in N. Once phase 2 becomes dominant, the total runtime depends on N with a power of 3. In figures 4.13 and 4.11 we see that the number of collisions grows quadratically in this region, and the runtime per frame is linear, so together it becomes a power of 3. The graph does seem to be slightly less steep at the end, suggesting that there is more to see. Increasing N was not an option due to very high runtimes, so this was the motivation to run the simulation with increased s.

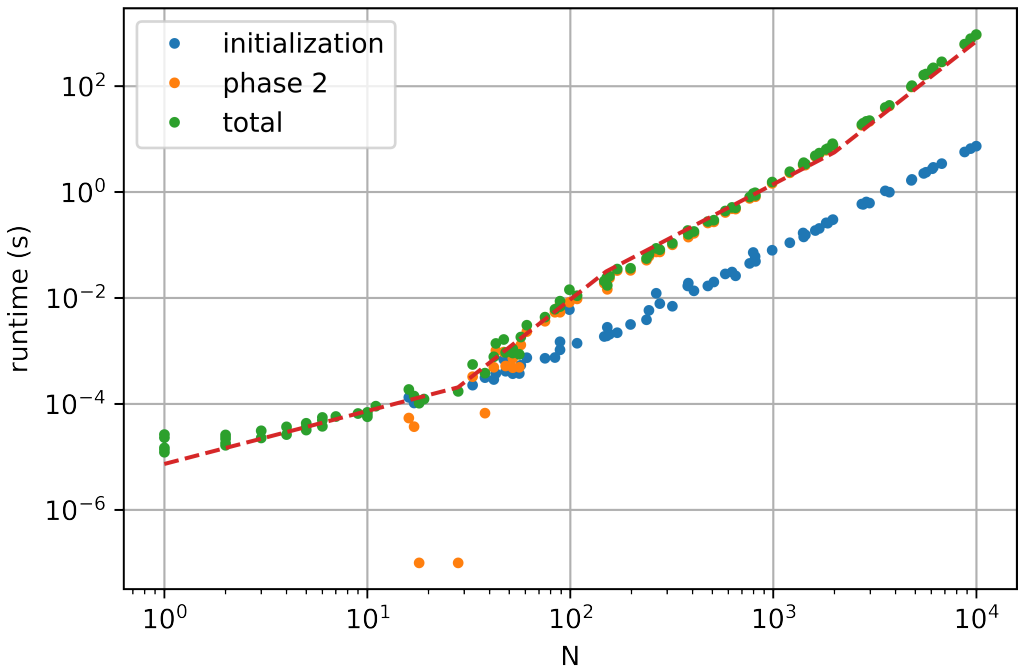

Figure 4.10: Runtime vs N for $s = 10^{-3} \cdot a_{\max}$ on a double log plot. The graph starts with the same shape as in figure 4.9, but scaled differently due to the larger s. Again, the initialization runtime dominates the first part, in this case until halfway between $N = 10$ and $N = 100$. From that point, phase 2 is dominant. The graph increases with a power of 3, until the next turning point above $N = 100$. From that point it temporarily increases quadratically, until halfway between $N = 10^3$ and $N = 10^4$, from which the runtime again increases with a power of 3. The dashed line illustrates this rational relationship.
Contrary to the standard simulations, the runtime of the frames in this set of simulations has an almost quadratic relationship with N (see figure 4.12.
The turning point after 10^2 can be explained by figure 4.14, where we see that the number of collisions, and hence the number of frames, turns linear instead of quadratic, hence the total runtime loses one power of N.

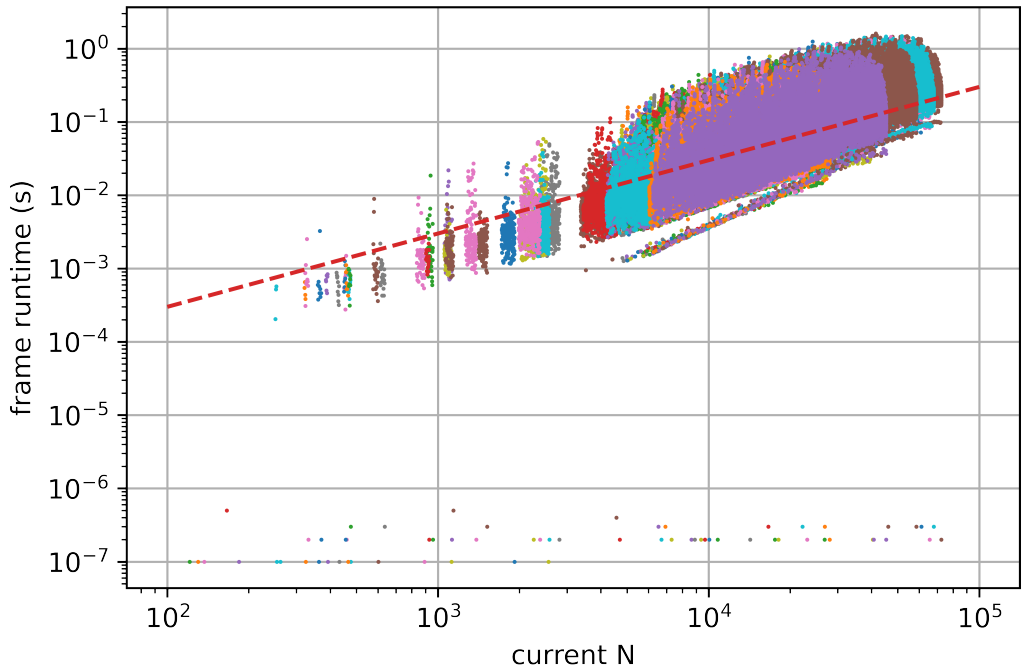

Figure 4.11: The runtime of all frames vs N for $s = 10^{-5} \cdot a_{\max}$ on a double log plot. A frame represents one loop in the algorithm, thus one collision being applied. Each color represents another simulation, although the dots in the graph overlap a lot. The dashed line shows a linear function for comparison. The runtime seems to be linear at first, although it appears to get slightly steeper at the far right of the graph. This is the expected result. From table 1.1, the most dominant parts of the frame runtime are $N^2 s/a$ and N. Hence, the runtime should be linear at first, and turn to quadratic for a sufficiently large N, which can be expected at $N > a/s$. Note that some points lie between 10^{-7} and 10^{-6} seconds. These must be the cases where the new merged planetesimal collided with the central body. In that case nothing has to be calculated anymore and the algorithm jumps to the next collision. While testing, 10^{-7} seemed to be around the resolution of the clock used, so it is unclear if those cases take up the full 10^{-7} seconds or if they get computed even faster.

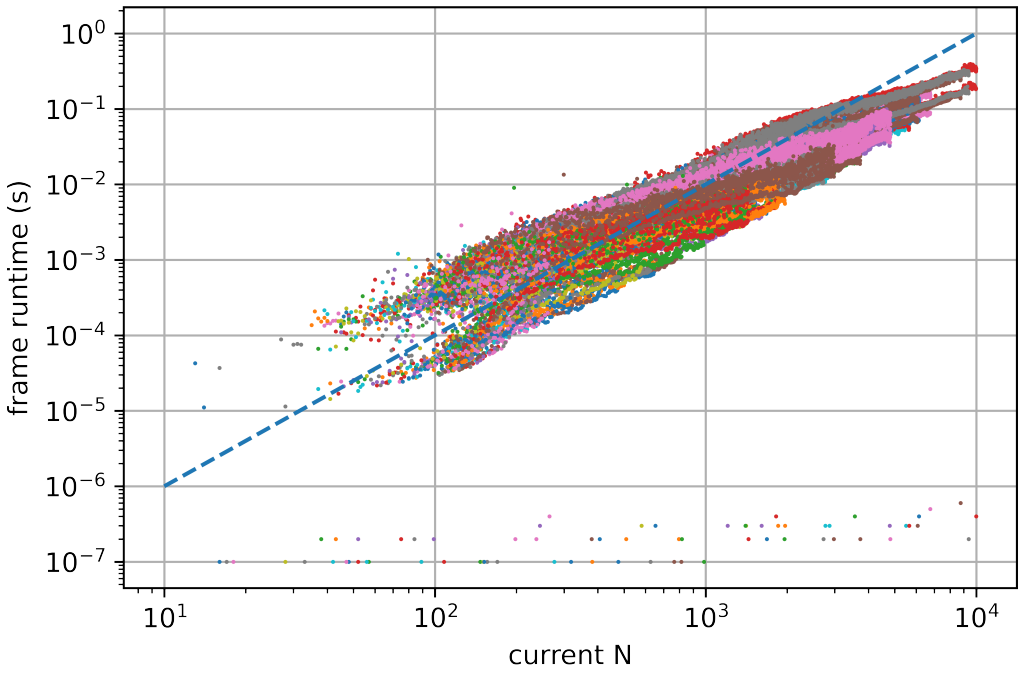

Figure 4.12: The runtime of all frames vs N for $s = 10^{-3} \cdot a_{\max}$ on a double log plot. A frame represents one loop in the algorithm, thus one collision being applied. Each color represents another simulation, although the dots overlap a lot. Contrary to figure 4.11, here the dashed line shows a quadratic function for comparison. The actual relationship is almost quadratic, but definitely more than linear. Although the dominant parts of the frame runtime are decided by integer powers of N, there are still a lot of other nonlinear parts that can cause the graph to not follow the shape of an integer-power function. Since s/a is increased to 10^{-3} in these simulations, all the terms in table 1.1 that contain s/a get more dominant, so in this case the N from "merge particles lists" is not dominant anymore, and hence the graph looks more quadratic than linear.

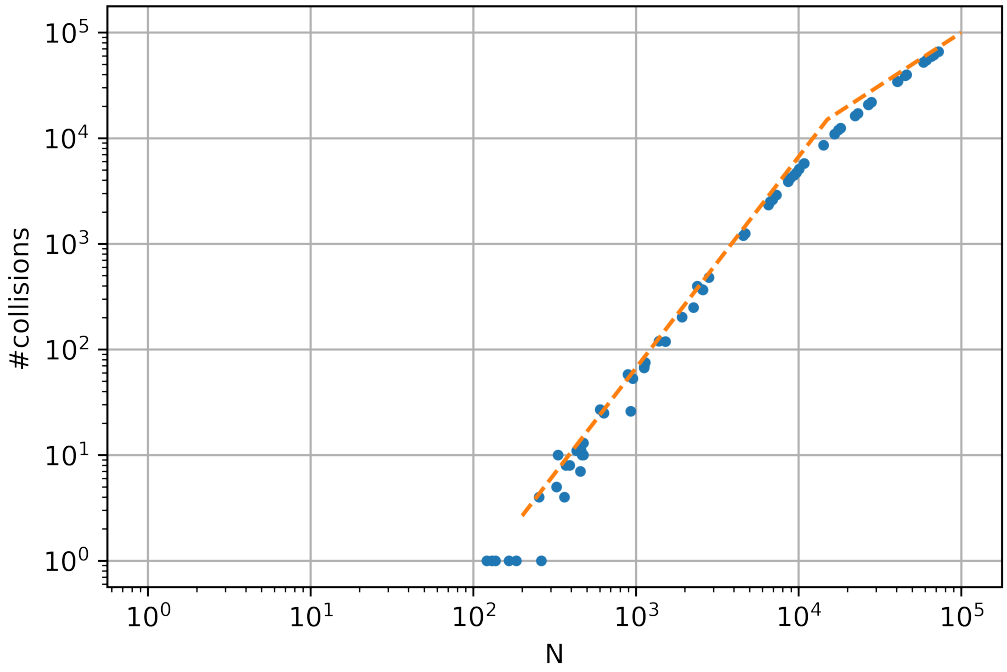

Figure 4.13: Total number of collisions vs N for $s = 10^{-5} \cdot a_{\max}$ (each dot represents one simulation) on a double log plot. As is mentioned in the caption of table 1.1, the number of collisions is expected to increase quadratically. At the right side of the graph, however, the slope seems to get lower. The dashed line shows quadratic and linear slopes. Figure 4.14 looks at a broader range of this graph, giving more insight.

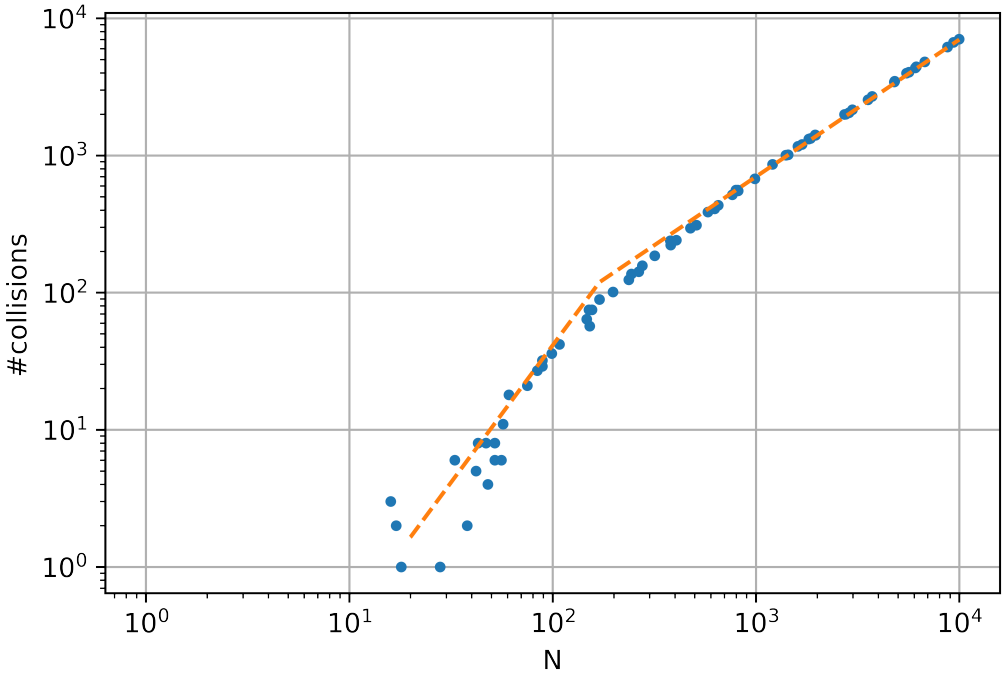

Figure 4.14: Total number of collisions vs N for $s = 10^{-3} \cdot a_{\max}$ on a double log plot. Just as in figure 4.13, the graph starts quadratically. As N gets larger, the graph turns linear. The dashed line shows a quadratic and linear slope for comparison. This makes sense, because the number of collisions cannot grow quadratically forever, since that would mean there are more collisions than planetesimals, while in each collision at least one planetesimal disappears. In other words, for sufficiently large N, almost all planetesimals collide eventually. Note that the dashed line is slightly lower than exactly N, but comes very close to being exactly N for high N.

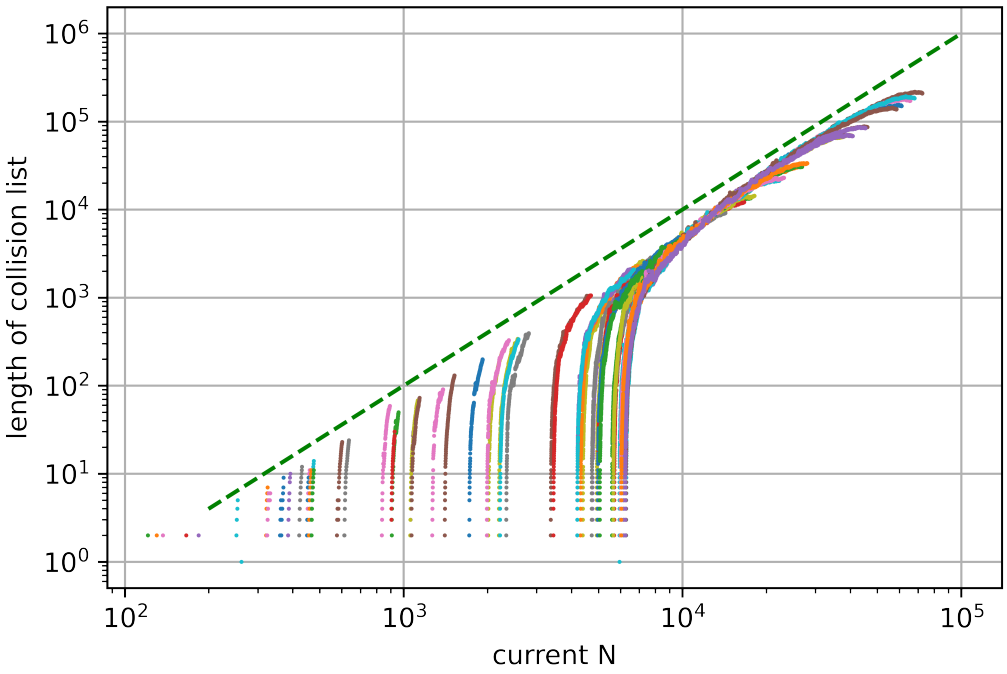

Figure 4.15: Lengths of the collision lists vs N throughout the simulations with $s = 10^{-5} \cdot a_{\max}$ on a double log plot. Each color represents a different simulation. We can see that, for simulations starting with low number of planetesimals, not many collisions happen while the length of the collision list drastically decreases to zero, ending the simulation. For the larger simulations however, we see that the length of the collision list roughly follows the same quadratic relationship as with the initial length. Remarkably, in the large simulations, the slope at which the length of the collision list decreases is a lot smaller, almost flat. The dashed line shows a quadratic function for comparison. Specifically the initial lengths of the collision lists are illustrated in figure 4.17.

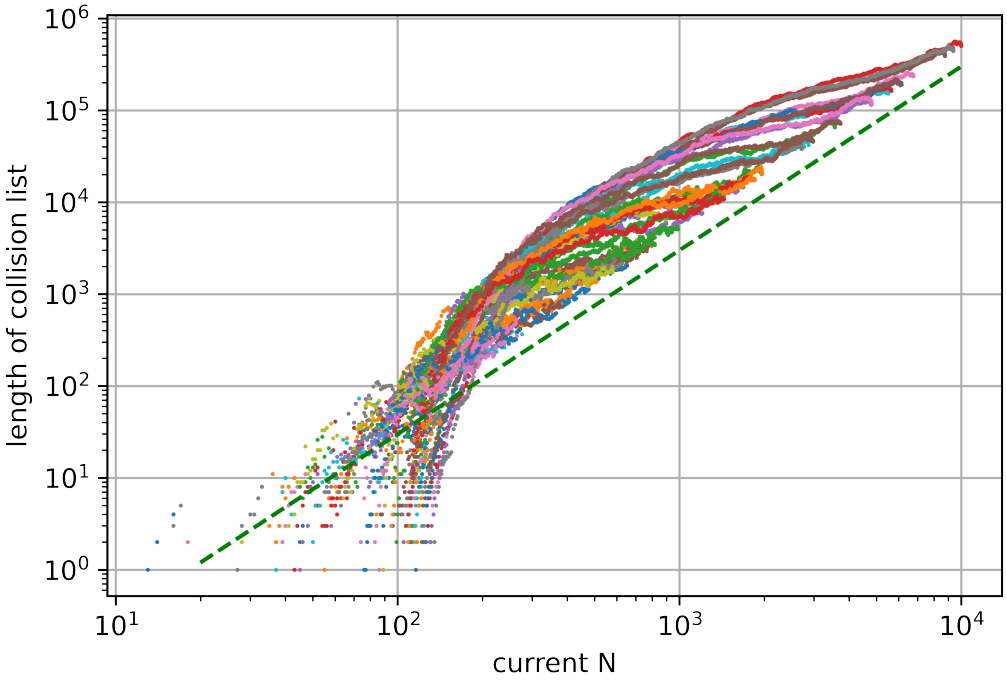

Figure 4.16: Lengths of the collision lists vs N throughout the simulations with $s = 10^{-3} \cdot a_{\max}$ on a double log plot. Each color represents a simulation. Just as in figure 4.15, the same quadratic line is visible for all the initial lengths. The dashed line is a quadratic line for comparison. In figure 4.15 the collision list moves roughly along the quadratic line as it shrinks down during the simulation, except for the two ends of the line, but in this figure the graphs deviate a lot more from the quadratic line. The quadratic relationship with the starting number of planetesimals still holds (more on that in figure 4.18), but for large N the list length seems to decrease linearly with each frame instead of quadratically.

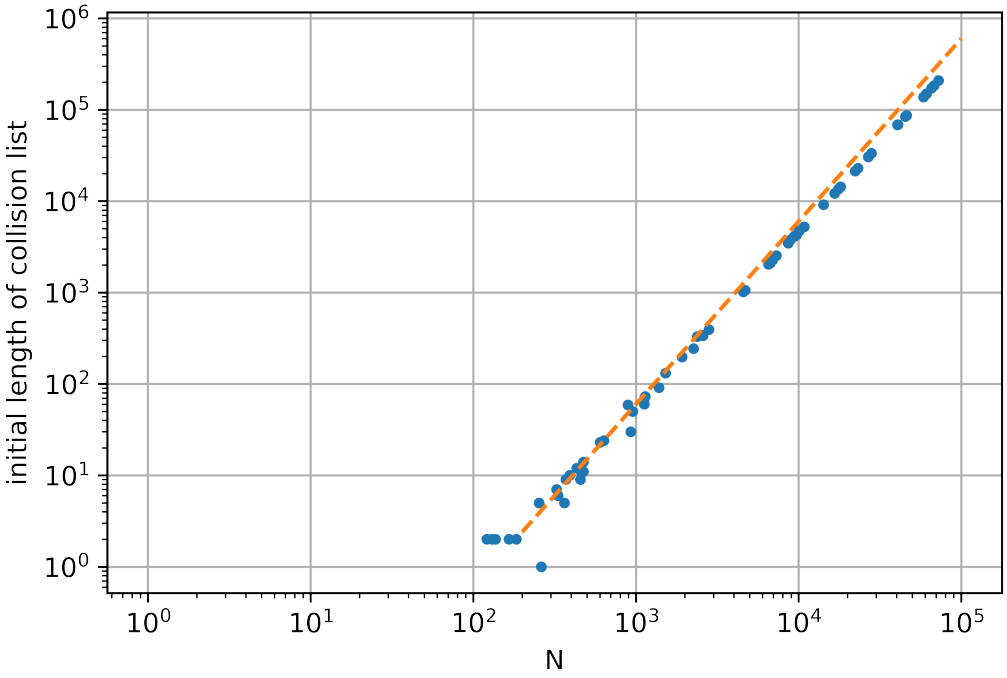

Figure 4.17: Initial lengths of the collision lists vs N for $s = 10^{-5} \cdot a_{\max}$ on a double log plot. The dashed line shows a quadratic line, illustrating that the initial length of the collision list has a quadratic relationship with n. Note that this graph is almost the same as figure 4.1. The only difference is that that figure was from the pair test, thus calculating the collision times and filtering out collisions outside the simulation time was omitted. This figure does does include the time calculations. This implies that, in this case, most collisions from the initialization are within the simulation time.

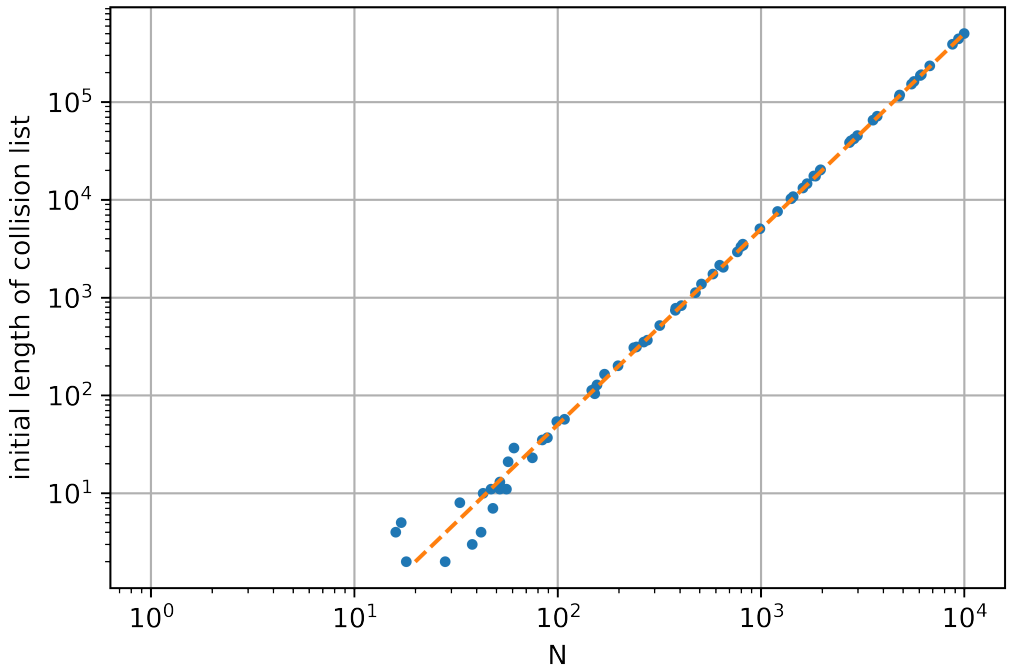

Figure 4.18: Initial lengths of the collision lists vs N for $s = 10^{-3} \cdot a_{\max}$ on a double log plot. This graph is the same as the graph in figure 4.17, but two orders of magnitude higher. By table 1.1, the number of pairs linearly on s/a, and s/a was increased by two orders of magnitude for this experiment, explaining the shift.

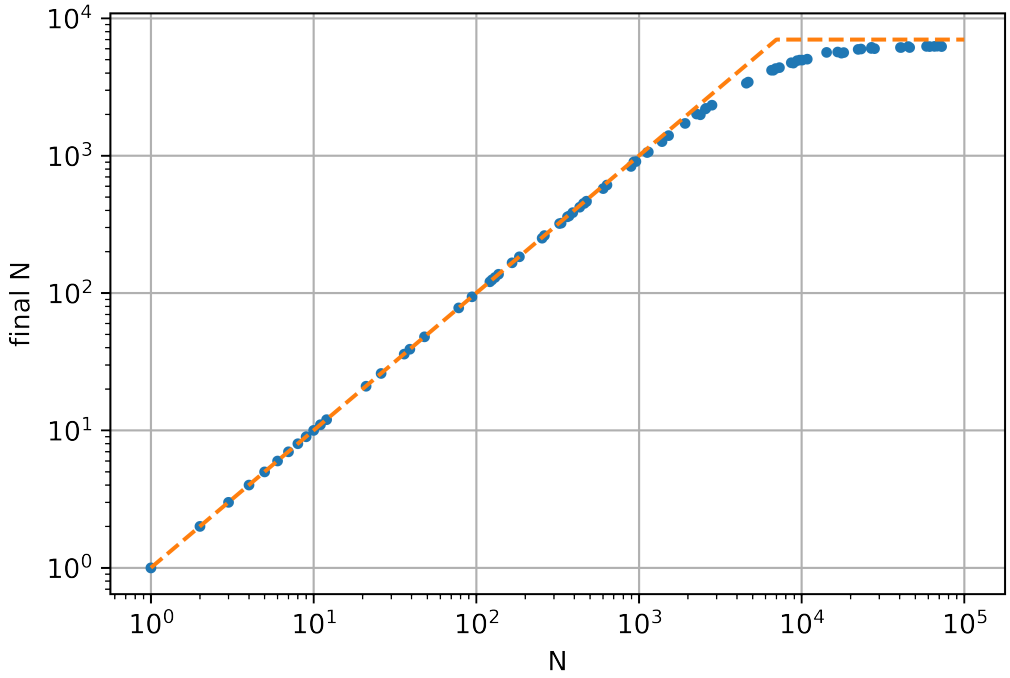

Figure 4.19: Remaining number of planetesimals vs N for $s = 10^{-5} \cdot a_{\max}$ on a double log plot. For low N, almost no collisions happen, hence the final N is close to the initial N. For sufficiently large N we see that the graph hits a plateau. This makes sense, because if more planetesimals would remain, they would be likely to still have collisions until they are back on the line. The dashed line shows a linear and constant function.

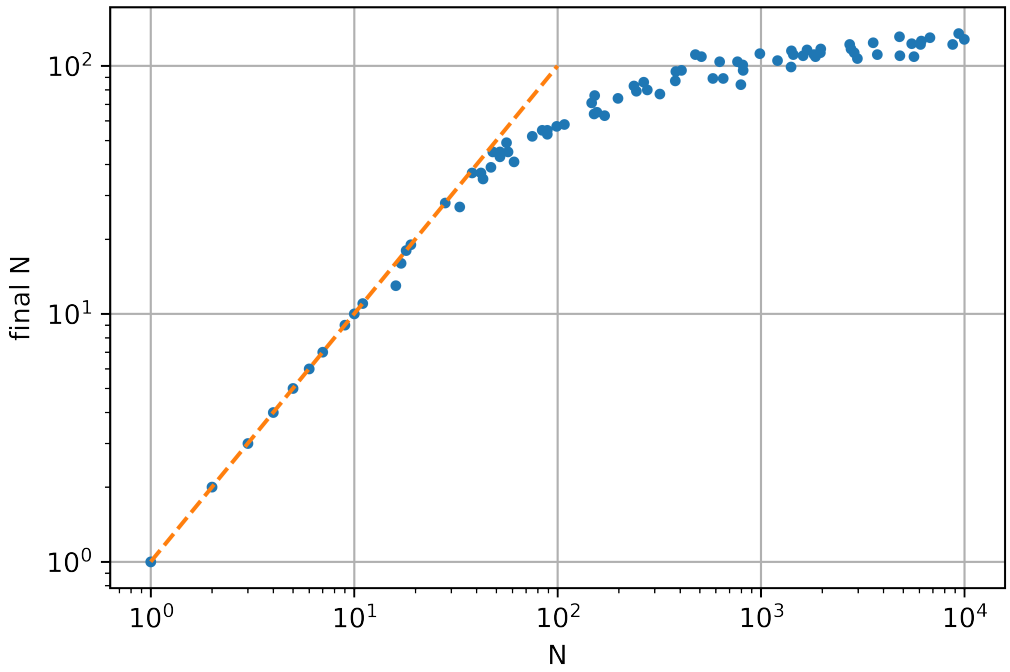

Figure 4.20: Remaining number of planetesimals vs N for $s = 10^{-3} \cdot a_{\max}$ on a double log plot. Just as in figure 4.19, the graph starts linear and flattens out for large N. Contrary to the other figure, the part for large N does not get fully constant, but still a lot less steep than a linear function.

4.3. Additional comments from development of the code

During development of the code, some interesting things happened that deserve to be mentioned.

- **Order of planetesimals in the continued fraction module**
  The continued fraction algorithm, which is part of Visser's algorithm and improved by Aron Schouten, accepts two planetesimals as input, which are ordered by orbital period. It was written so that one planetesimal has to close the gap such that they end up simultaneously at the right place. In some cases, the planetesimal with the smallest orbital period was not the planetesimal behind, and instead it was the other planetesimal that needed to close the gap. Hence, the roles of the planetesimals needed to be switched. Getting into details of the continued fractions algorithm is not in the scope of this paper, but for the interested reader, the roles of the variables k and l from Schouten's thesis [8] were switched. This corresponds with the line going through the lattice starting above zero instead of below. This problem was found when checking the results visually and seeing that it worked just half of the time before (after applying the solution it worked consistently everytime).

- **Rare illegal escape orbits**
  Because we are using the simple merger (inelastic collisions), and all planetesimals start with closed orbits, no new planetesimals should be able to get in an escape orbit. However, in some rare cases planetesimals still got escape orbits. This is due to the linearization around the orbital nodes. That approximation only works for sufficiently large mutual I between the planetesimals. In some rare cases, two colliding planetesimals coincidentally have almost the same inclination, which results in the approximation over-estimating the distance of the collision point to the central body. Since the new planetesimal still gets the same momentum, but starts farther from the central body, it might get an escape velocity. Then the semi-major axis gets negative, because of the way the orbital mechanics were implemented. However the program was not designed to handle escape orbits, and this resulted in numbers getting the value NaN (Not a Number) during calculations. Ultimately this crashes the continued fractions module into a practically infinite loop (10^{19} iterations. This is the order of the numerical limit of the 64-bit integer). This has been solved by deleting any such planetesimals, because by the algorithm escape orbits should be deleted as well as planetesimals colliding with the central body.

- **Iterative scheme at the orbital nodes**
  The analytical propagator uses a linearization approximation around the orbital nodes in order to calculate where a collision would happen. In some rare cases where the orbits have a very low mutual inclination, this approximation becomes visibly far from the real position. During a meeting, an idea came up to make an iterative scheme that uses this linearization multiple times in order to come to the right answer in multiple steps. Essentially it is the same linearization, then projecting the calculated position onto the orbit, and repeat. When implemented, the results looked very good visually. However, somewhere deeper it seemed to not always converge. It only worked when a maximum number of iterations was given, which then consistently good results. No further time has been spent in developing this iterative scheme, but it does have potential. This iterative scheme has been left out of the final program, in order not to cause unforeseen consequences.

- **Low memory use**
  Researching the memory used by Visser's algorithm was not the goal of the experiments. However, memory usage was still visible in the program editor. The algorithm is very light in terms of memory use, to the point that memory is no point of concern at all. During the experiments the memory usage was in the order of hundreds of megabytes. Most simulations used around 100 MB.

5

Conclusion

The goal of this thesis was to get more insight in the efficiency of Visser's algorithm for collision detection in N-body Kepler systems. The runtime of simulations was tested as well as some properties related to runtime. First, part of the initialization was tested, where the following four variables of the simulated disk were varied: number of particles N, maximum eccentricity $\epsilon_{\max}$, maximum inclination $I_{\max}$ and planetesimal radius s. After that, two sets of full simulations were done, one with $s = 10^{-5} \cdot a_{\max}$ and one with $s = 10^{-3} \cdot a_{\max}$.

Checks and pairs
The number of checks and pairs in the initialization depends on the tested variables mostly in the expected way. The number of checks and pairs depends quadratically on N, and so does the runtime of the whole initialization. The number of checks and pairs is also affected by the other variables. The impact of other variables is mostly linear, but only within the range where the effect of the variable is dominant relative to the other variables. The maximum inclination does not affect the number of checks, and does not have significant influence on the runtime of the initialization.

Initialization runtime
The runtime of the full initialization follows the same pattern as the partial initialization (without collision time), because it still scales with the number of checks and pairs. The runtime of the initialization is roughly a combination of the number of checks and the number of pairs, because for both checks and pairs, computational work needs to be done. For larger N ($N > 10^3$ for $s = 10^{-5} \cdot a_{\max}$ and $N > 10^2$ for $s = 10^{-3} \cdot a_{\max}$), the initialization runtime is quadratic in N. For smaller N the initialization runtime is linear, because creating a random particle list and sorting it takes the most time in that case. The value of N at which this turning point lies is confirmed to depend on s, by the two sets of full simulations giving different results, but it should also depend on any other variables that affect the number of checks and/or pairs nodes, since the initialization runtime becomes quadratic in N once the number of pair nodes is large enough.

Full runtime
The total runtime consists mostly of the initialization in the range of N for which the initialization is linear. The runtime of the second phase (the part after initialization) seems to become dominant just before the initialization turns quadratic in N. The behaviour of the total runtime is complicated, but for the largest N tested ($N = 10^5$), the algorithmic efficiency of the algorithm was $O(N^3)$. The total runtime of the simulation was predicted to be of order $O(N^2\epsilon + \frac{N^4 s^3}{Ia^3})$. In measured order of N is thus one power less. This can be explained due to the number of collisions becoming linear in N for sufficiently large N, while the prediction of the fourth-power region was based on a quadratic number of collisions. This is good news, because the algorithm is thus faster than anticipated. The runtime of each frame is somewhere between linear and quadratic in N depending on the parameters.

Length of collision list and remaining particles
Because the possible collisions need to be memorized, we also wanted to know how long this list can

become. We thus also examined the length of the collision list during the simulations. This length does not follow the initial quadratic line in N throughout the simulation. The order of the rate per collision at which the collision list shrinks is at the start of the simulation much lower than at the end. For large enough N, the length of the collision list is found to decrease quadratically in the middle part of the simulation. The shrinking rate being the highest at the end of the simulation makes sense, because the impact of one planetesimal disappearing is relatively bigger when there are less other planetesimals around. The number of remaining planetesimals after a simulation seems to converge to a constant. This is because the probability of more planetesimals not having collisions is unlikely.

Advantages and limitations of the method

The analytic propagator is efficient for sparsely distributed planetesimals. It takes advantage of the planetesimal size being very small compared to the size of the orbits, and thus collisions being rare. Since the algorithm jumps from collision to collision, it can make very big jumps through time if collisions are rare. On the other side, if the protoplanetary disk is densely distributed (high s and/or N), collisions happen more often, and thus the analytic propagator has to perform more computational work per simulated time-unit, thus becoming less effective. However, the octree code remains unaffected by this change. It has to perform nearest neighbour searches for collision detection, which is independent of the size of the planetesimals.

Furthermore, some approximations are made that make the analytical propagator less accurate. The orbits are treated as perfect Kepler orbits, but realistically, these orbits could be perturbed in the long term, by secular effects such as orbital precession. Usually this takes in the order of millions of years, so simulations within shorter time intervals are safe. The values of N and s need to be high enough to justify neglecting precession within a fixed amount of time.

Future recommendations

The behaviour of the total runtime is complicated, and for part of it, the causes are not clear. Therefore, measuring the runtime of each individual step in the main loop of the algorithm could provide more insight in how different parts of the main loop become dominant in different regions. In this thesis only the runtime of the whole loop was measured for each cycle, as well as the runtime of the initialization.

Furthermore, s has a big impact on the runtime of a simulation. In the limiting case where s goes to zero, the runtime becomes just the initialization time, because no collisions happen and the algorithm stops in the first iteration of the main loop. Hence, it would be useful to systematically measure the impact of s on the total runtime by varying s while holding N (and all other variables) fixed. In this thesis only two different values of s were considered. This had the purpose of seeing how the runtime behaves with respect to N for $N < a/s$ and $N > a/s$.

If the algorithm is to be used in a context where accuracy of individual collisions matter, it could be expanded to be more accurate. That would involve including escape orbits for possible collisions, and improving the calculation of the position of the collision, which is currently done by a linear approximation around the orbital nodes. This linear approximation works well, except in rare cases were two orbits coincidentally have an extremely low mutual inclination. Although these cases are rare, it does result in some collisions not being accurate. An idea of an iterative scheme repeatedly using linearization has been tested. The results looked promising visually, but on a smaller scale it seemed to not always converge. Perhaps with more careful ending conditions that could be fixed.

References

[1] Robert A. Adams and Christopher Essex. *Calculus*. Pearson, ninth edition, 2018. ISBN 978-0-13-415436-7.

[2] R.E.A. Bouwens, P.A.M. de Groot, W. Kranendonk, J.P. van Lune, C.M. Prop van den Berg, J.A.M.H. van Riswick, and J.J. Westra. *Binas*. Noordhoff Uitgevers, sixth edition, 2013. ISBN 978-90-01-81749-7.

[3] H. Goldstein. *Classical Mechanics*. New York: Dover, 1964.

[4] Wm. Robert Johnston. Known populations of solar system objects: 25 july 2022. URL `http://www.johnstonsarchive.net/astro/sslistnew.html`.

[5] Donald Meagher. octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. 1980.

[6] Carl D. Murray and Stanley F. Dermott. *Solar System Dynamics*. New York: Cambridge University Press, 2000. ISBN 9781139174817.

[7] Dmitry Savransky, Eric Cady, and N. Jeremy Kasdin. Parameter distributions of keplerian orbits. *The Astrophysical Journal*, 2011. doi: 10.1088/0004-637X/728/1/66.

[8] Aron Schouten. Collision detection using continued fractions, 2022. URL `https://repository.tudelft.nl/islandora/object/uuid:77150c61-c226-4ecb-983b-86a9a41645f4`.

[9] P.M. Visser. Collisional kepler system. Submitted to *Astronomy & Astrophysics*, 2022.

A

Appendix

A.1. Visualization of the planetary disk

For debugging and presentation purposes, it is useful to visualize the simulations that we make with Visser's algorithm. In order to visualize the state of the simulation onto the screen at various times, we will need some procedures capable of doing certain coordinate transformations and other calculations that are not part of the algorithm itself. The angular momentum $\boldsymbol{L}$ and eccentricity vector $\boldsymbol{\epsilon}$ completely define an orbit [3], and all procedures in this chapter make use of that vectorial formulation of orbits.

In this chapter we will derive everything we need in order to draw planetesimals and orbits on the screen. To do that, everything has to be expressed in Cartesian coordinates. Note that these procedures can be done on the graphics card itself, in so-called shader programs, which does the calculations in parallel, so that even millions of particles can be rendered at a smooth frame-rate.

A.1.1. Orthonormal basis of the orbital plane

In some cases we need a set of two orthonormal vectors that span the orbital plane of a particle inside 3D space. When we have found such a set, we can use it to transform 2D representations inside the orbital plane, such as the polar formula (2.3), into 3D space.

Let $\hat{\boldsymbol{L}}$ be the normalized vector in the direction of the angular momentum, and let $\hat{\boldsymbol{\epsilon}}$ be the normalized vector in the direction of the eccentricity. Note that $\hat{\boldsymbol{\epsilon}}$ and $\hat{\boldsymbol{L}}$ are orthogonal for all valid orbits, since the eccentricity vector points parallel to the orbital plane, while the angular momentum is orthogonal to the orbital plane. In order to span the orbital plane, we want to find another vector $\hat{\boldsymbol{\mu}}$, orthogonal to $\hat{\boldsymbol{\epsilon}}$ and $\hat{\boldsymbol{L}}$, which can be found using the cross product:

$$\hat{\boldsymbol{\mu}} = \hat{\boldsymbol{L}} \times \hat{\boldsymbol{\epsilon}}. \tag{A.1}$$

This results in the desired orthonormal basis of the orbital plane, depicted in figure A.1. However, we are not able to normalize $\boldsymbol{\epsilon}$ if the orbit has no eccentricity, which is the case for circular orbits. That makes sense, because without an eccentricity, the perihelion and thus the true anomaly are not defined, so the direction of $\hat{\boldsymbol{\epsilon}}$ is in fact arbitrary. In order to make the procedure "water proof", we may assign a direction for $\hat{\boldsymbol{\epsilon}}$ ourself. Note that, when using Kepler coordinates, the direction $\hat{\boldsymbol{\epsilon}}$ is the first column of the rotation matrix $\mathcal{R}$ described in equation (2.2), since $\boldsymbol{\epsilon}$ is parallel to the x-axis before rotating:

$$\hat{\boldsymbol{\epsilon}} = \begin{bmatrix} \cos\varpi\cos\Omega - \sin\varpi\sin\Omega\cos I \\ \cos\varpi\sin\Omega + \sin\varpi\cos\Omega\cos I \\ \sin\varpi\sin I \end{bmatrix}. \tag{A.2}$$

When the eccentricity is zero, the argument of periapsis is not defined, so we choose $\varpi = 0$ as the default angle, resulting in

$$\hat{\boldsymbol{\epsilon}} = \begin{bmatrix} \cos\Omega \\ \sin\Omega \\ 0 \end{bmatrix}. \tag{A.3}$$

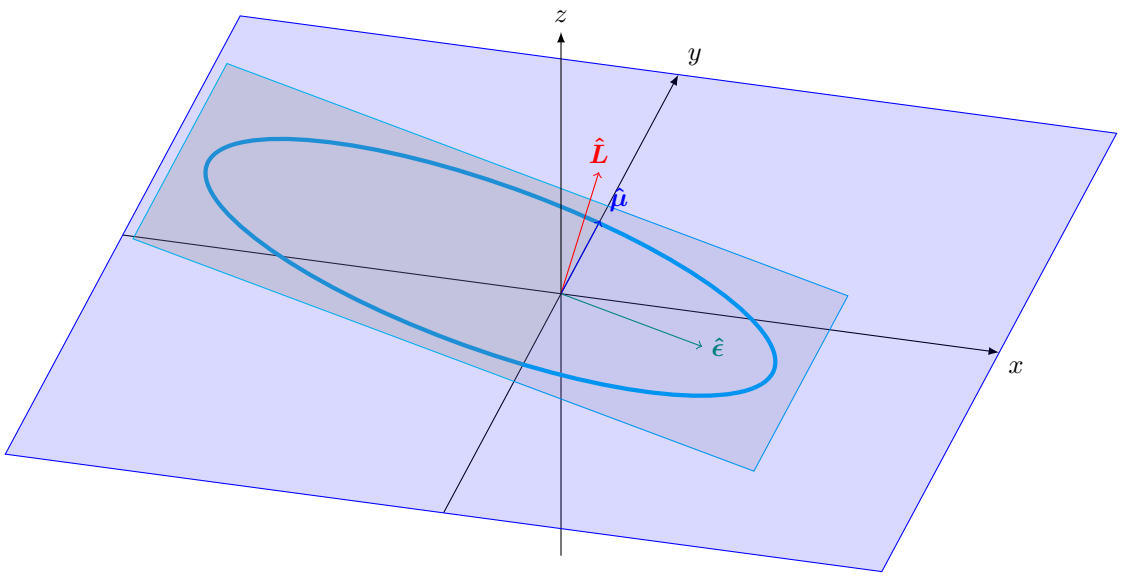

Figure A.1: The orbital plane is represented by the projection of a square spanned by the basis $\{\hat{\boldsymbol{\epsilon}}, \hat{\boldsymbol{\mu}}\}$. This plane is orthogonal to the angular momentum $\boldsymbol{L}$. In this picture the plane has an inclination of 15 degrees and no other rotations. The orbit is an ellipse with an eccentricity of $\frac{1}{2}\sqrt{3} \approx 0.866$, corresponding with the semi-minor axis being half of the semi-major axis.

We now compare $\hat{\boldsymbol{\epsilon}}$ with $\hat{\boldsymbol{L}}$, which equals the third row of $\mathcal{R}$, since $\boldsymbol{L}$ points in the z-direction before applying rotation:

$$\hat{\boldsymbol{L}} = \begin{bmatrix} \sin\Omega \sin I \\ -\cos\Omega \sin I \\ \cos I \end{bmatrix}. \tag{A.4}$$

Now there are two cases:
case 1: $\sin I = 0$.
In this case $\hat{\boldsymbol{L}}$ only has a non-zero component in the z-direction. That means that also the ascending node Ω is not defined, which we also choose to be zero, resulting in $\hat{\boldsymbol{\epsilon}} = \hat{x}$.
case 2: $\sin I \neq 0$. In this case we may divide by $\sin I$. By comparing (A.3) and (A.4), we note that

$$\hat{\boldsymbol{\epsilon}} = \begin{bmatrix} \frac{-\hat{L}_2}{\sin I} \\ \frac{\hat{L}_1}{\sin I} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{-\hat{L}_2}{\sqrt{1-\cos^2 I}} \\ \frac{\hat{L}_1}{\sqrt{1-\cos^2 I}} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{-\hat{L}_2}{\sqrt{1-\hat{L}_3^2}} \\ \frac{\hat{L}_1}{\sqrt{1-\hat{L}_3^2}} \\ 0 \end{bmatrix}. \tag{A.5}$$

Having accounted for all possible cases, we have a systematic way of defining $\hat{\boldsymbol{\epsilon}}$, using the principle of choosing the smallest Kepler angles. Note that if particles actually achieve these edge cases, calculating positions using this method is problematic. This is because in the limit where the eccentricity goes to zero, not all Kepler orbits are defined. Thus assuming the smallest angle, which defines our basis, gives a different result depending on which way the xyz axes are defined. Therefore, using this procedure is not fully safe. Nevertheless, in practice these edge cases will never occur, since the orbits need to be a "perfect" orbit within floating point precision. Also, for drawing the orbits on the screen this does not pose a problem even for edge cases. That is because an orbit with zero eccentricity is a perfect circle, which therefore does not depend on what angle you start drawing from.

A.1.2. Position from true anomaly

We want to calculate the position of a particle, given the true anomaly ν, and the following data of the particle: semi-latus rectum l, eccentricity vector $\boldsymbol{\epsilon}$ and angular momentum vector $\boldsymbol{L}$.
We will use the orthonormal basis $\{\hat{\boldsymbol{\epsilon}}, \hat{\boldsymbol{\mu}}\}$ from section A.1.1. We apply the polar transformation on the basis, so that we can use the polar formula for orbits (2.3)

$$|\boldsymbol{r}| = \frac{l}{1 + |\boldsymbol{\epsilon}| \cos \nu}. \tag{A.6}$$

We also apply the polar transformation inside the orbital plane:

$$\hat{\boldsymbol{r}} = \hat{\boldsymbol{\epsilon}} \cos \nu + \hat{\boldsymbol{\mu}} \sin \nu. \tag{A.7}$$

Finally, by calculating $|\boldsymbol{r}|\hat{\boldsymbol{r}}$ we compute the position in Cartesian coordinates.

A.1.3. True anomaly from position

In section A.1.2 we calculated the position of a particle using the true anomaly of a particle. Now we will show that it also works backwards: Calculating the true anomaly, given the position of a particle. We will use the orthonormal basis $\{\hat{\boldsymbol{\epsilon}}, \hat{\boldsymbol{\mu}}\}$ from section A.1.1. Recall that, even though the true anomaly is ill-defined in some edge cases, we still get a determinate answer by the principle of choosing the smallest angle. In that case it does not make sense to do this calculation, but because it gives a deterministic answer it is suitable for programming, without worrying about edge cases that could crash the program.

Let $\boldsymbol{r}$ be the given position of the particle. Note that any valid given position should be in the orbital plane, which is spanned by $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\epsilon}}$. Hence we can decompose $\boldsymbol{r}$ in projections on $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\epsilon}}$:

$$r_\perp = \boldsymbol{r} \cdot \hat{\boldsymbol{\mu}},$$
$$r_\parallel = \boldsymbol{r} \cdot \hat{\boldsymbol{\epsilon}},$$

where $\cdot$ denotes the dot product.
After that we calculate the arc tangent of $r_\perp / r_\parallel$ to obtain the true anomaly. Most programming languages have a built-in function called atan2(y, x) which also determines in which quadrant the angle is, hence having a periodicity of 2π. Usually this returns the angle inside the domain $(-\pi, \pi]$, but for most calculations in this paper we are calculating in the domain $[0, 2\pi)$. This can be converted by adding 2π whenever the angle returned by atan2 is negative.
Hence, we calculate the true anomaly ν with:

$$\nu = \text{atan2}(r_\perp, r_\parallel) \quad \nu \in [0, 2\pi). \tag{A.8}$$

A.1.4. Position from eccentric anomaly

We want to calculate the position $\boldsymbol{r}$ of a particle, given its eccentric anomaly E, where we will also use the semi-major and minor axes a and b of its orbit. Recall that b can be calculated from $b^2 = (1 - \epsilon)a^2$. We will use the orthonormal basis $\{\hat{\boldsymbol{\epsilon}}, \hat{\boldsymbol{\mu}}\}$ from section A.1.1. Per definition, the Laplace-Runge-Lenz vector $\boldsymbol{\epsilon}$ points from the center of an orbit to the focal point where the central body is located. The distance c from center to focus is $c = \epsilon a$. Hence, the center of the orbit is located at $-\boldsymbol{\epsilon} a$. Seen from the center of the orbit, the ellipse follows the parametrisation of a "squeezed" circle, giving us the formula

$$\boldsymbol{r} = -\boldsymbol{\epsilon} a + a\hat{\boldsymbol{\epsilon}} \cos E + b\hat{\boldsymbol{\mu}} \sin E. \tag{A.9}$$

A.1.5. Eccentric anomaly from position

We will perform the process in A.1.4, but backwards:
Let $\tilde{\boldsymbol{r}} = \boldsymbol{r} + \boldsymbol{\epsilon} a$
Here, $\tilde{\boldsymbol{r}}$ represents the position of the particle relative to the middle of its orbit. Note that the particle lies in its orbital plane, hence its position can be decomposed into a linear combination of two vectors, for which we use the the orthonormal basis $\{\hat{\boldsymbol{\epsilon}}, \hat{\boldsymbol{\mu}}\}$ from section A.1.1:

$$\tilde{\boldsymbol{r}_\parallel} = \tilde{\boldsymbol{r}} \cdot \hat{\boldsymbol{\epsilon}},$$
$$\tilde{\boldsymbol{r}_\perp} = \tilde{\boldsymbol{r}} \cdot \hat{\boldsymbol{\mu}}.$$

By the already known parametrisation we also have

$$\cos E = \tilde{\boldsymbol{r}_{\parallel}}/a,$$
$$\sin E = \tilde{\boldsymbol{r}_{\perp}}/b.$$

Combining the two pairs of formulas has the following results:

$$\begin{aligned}\cos E &= \tilde{\boldsymbol{r}} \cdot \hat{\boldsymbol{\epsilon}}/a,\\ \sin E &= \tilde{\boldsymbol{r}} \cdot \hat{\boldsymbol{\mu}}/b,\\ E &= \operatorname{atan2}(\sin E, \cos E) \quad E \in [0, 2\pi).\end{aligned}$$

A.1.6. Eccentric anomaly from time

In the analytic algorithm described by Visser ([9]), we never have to calculate the eccentric anomaly of a particle at an arbitrary given time. However, for visualization purposes we still need that calculation so that we can view the state of the simulation at different times and check if everything works properly (debugging).

Let t be the time since the particle last passed its periapsis and ω the angular frequency at which the particle orbits the central body. Then for the eccentric anomaly we have the so-called Kepler's equation

$$\omega t = E - \epsilon \sin E. \tag{A.10}$$

We do not have an explicit formula for calculating E, so we will rely on numerical methods. We transform equation (A.10) into the following fixed point problem:

$$E = \omega t + \epsilon \sin E. \tag{A.11}$$

We will solve this fixed point problem in an iterative way, also known as a Picard iteration:

$$E^{(0)} = \omega t, \tag{A.12}$$
$$E^{(n)} = \omega t + \epsilon \sin E^{(n-1)}. \tag{A.13}$$

Note that we have chosen ωt as the initial guess for the solution, since that would be the exact solution in case of a circular orbit. For ellipses we have to do more iterations in order to obtain an acceptable accuracy. In our case we only use this for visualisation. In practice 10 iterations will be enough for visualisation.

Using this we can update the time in order to visualize where the particles are. We first calculate the starting eccentric anomaly of each particle (section A.1.5), where "starting eccentric anomaly" refers to its value at the time t_0 and place r_0 where the particle emerged in the simulation. Using formula (A.10) we then calculate its corresponding value of ωt. We add $\omega(t_{\text{current}} - t_0)$ to "go" to the current time, and finally we use the iteration method to obtain the eccentric anomaly at the current time.

The eccentric anomaly E is ill-defined for (near-) circular orbits. However, since we use this process only for visualisation, we do not have to worry about errors propagating through the algorithm. In the worst case, the specific particle will not be drawn correctly on the screen, but this requires an eccentricity comparable to floating point precision, which practically never happens.

A.2. Technical specifications

CPU: AMD Ryzen 9 5900X 12-Core Processor @3.70 GHz
(The program was not programmed multi-core, so it ran on 1 CPU thread).
RAM: 16,0 GB
System type: 64-bits besturingssysteem, x64-processor
OS: Windows 11 Education, version 21H2. Build 22000.856

A.3. Code

The code for this project was written in C++, using Visual Studio 2022 as the editor. For the graphics, OpenGL was used, along with the GLFW library (GL For Windows). Also the GLM library was used for some mathematical functions. The simulation data was saved into text files and post-processed in a Jupyter Notebook (Python). The ensemble of files containing the code is very big and complicated, so putting it in the appendix is not organized. Instead, the code has been uploaded to GitHub, accesible by the following link: `https://github.com/Dylan-Aliberti/Kepler-Collisions`