Goethe University Frankfurt

Advanced Introduction to C++ and Scientific Computing

Prof. Dr. Claudius Gros

Roche Limit - Project Report

Author:
Hendrik Edelmann, Lars Gröber

January 13, 2017

Contents

1 Introduction

For the module "Advanced Introduction to C++ and Scientific Computing" we created a program for our project [3] to simulate an asteroid passing by a planet within the Roche limit and thus being torn apart.
We programmed in a team of two and it took us about two and a half week to finish the project. The source code can be found in the supplementary material alongside some GIFs showing the progress. The program is written in C++.
To check our simulation we simulated first a toy example which presents the concept quite drastically. Then we turned to a real case of one of Mars' moons, Phobos.
In this document we first explain what the Roche Limit is and how it can be derived, we discuss the physics behind the project and show how we implemented it, lastly we take a look at the aforementioned examples.

— Lars & Hendrik

2 Roche Limit

The Roche Limit describes the maximum radius an object can pass by another object in space without disintegrating itself. It was discovered in 1850 by Édouard Albert Roche.

Two objects in space, only held together by their own gravitational force will also attract onto each other and therefore pull on each others self attracting gravitational force, which is called tidal force. If the tidal force gets equal to the gravitational self attraction we have reached the Roche limit. At this point the smaller object will disintegrate. Objects with a low density have a Roche limit that lies inside the body itself and can therefore not be reached.

2.1 Derivation of the Roche Limit [1]

The gravitational force F_G on a mass u on the surface of the satellite towards the satellite (mass m, radius r) can be expressed by

$$F_G = \frac{Gmu}{r^2}$$

the tidal force F_T acting on the mass u towards the planet (mass M, radius R), with d being the distance between the centers of satellite and planet, is given by

$$F_T = \frac{GMu}{(d-r)^2} - \frac{GMu}{d^2}$$

The Roche Limit is reached once F_G and F_T are equal.

Using the approximation $r \ll R$ and $R < d$ we have for F_T

$$F_T = GMu\frac{2dr - r^2}{d^4 - 2d^3r + r^2d^2} \approx \frac{2GMur}{d^3}$$

Calculating now $F_T = F_G$ gives

$$\frac{Gmu}{r^2} = \frac{2GMur}{d^3}$$

from which we arrive at the Roche Limit

$$d = r\sqrt[3]{2\frac{M}{m}}$$

3 Physics

In order to simulate an satellite being able to be torn apart we needed some way to form the satellite of a large number of individual mass points. In addition to that we also had to simulate gravity and some kind of repulsion between the mass points to keep the satellite in shape.

3.1 Basic movement

The basic movement of any given mass point is rather simple. The underlying differential equation is given by

$$\frac{d^2}{dt^2}(mx(t)) = F(t, x)$$

Where F is the gravitational force acting on every mass point. To evaluate this equation we first calculate the total force acting on that point and use this force to change the velocity of that point according to

$$v_{new} = v_{old} + F/m * dt$$

F/m of course equals the acceleration and dt is the time step between simulation frames. Changing dt will greatly change the precision of any simulation. After calculating the velocity we go on calculating the mass points' position using the same technique

$$r_{new} = r_{old} + v * dt$$

3.2 Cloud of mass points

In order to simulate a satellite being build up by a number of mass points we decided before hand on a number of points N and gave each of them a mass of m_{sat}/N. Forming the cloud we used an algorithm creating an archimedic spiral.

3.3 Collision

To keep the satellite in shape we came up with some ideas. Proposed was a repulsive potential to keep the mass points at a specific distance introducing oscillation into

the system. To account for these we would have needed some kind of damping which would have had to be turned off at some arbitrary distance.

Instead we chose to implement inelastic collisions. They needed to be inelastic to introduce no oscillations, so we made sure that mass points would not "bounce". To calculate the final velocity of two mass points of mass m_1 and m_2 we used

$$m_1 * \vec{v}_1 + m_2 * \vec{v}_2 = (m_1 + m_2) * \vec{v}'$$

$$\vec{v}' = \frac{m_1 * \vec{v}_1 + m_2 * \vec{v}_2}{m_1 + m_2}$$

Using an inertial system in which $\vec{v}_2 = 0$, so we have $\bar{\vec{v}}_1 = \vec{v}_1 - \vec{v}_2$, we can easily calculate the resulting velocity:

$$\bar{\vec{v}}' = \frac{m_1}{m_1 + m_2} \bar{\vec{v}}_1$$

The same is true for $\bar{\vec{v}}_2$, we just have to replace the mass in the numerator by m_2. To check if a collision happened one can just calculate the distance between the centers of both mass points. If this distance is smaller than the sum of the two radii both mass points collided. Resolving this collision is also rather simple, one has only to move both mass points away from each other along the line connecting both centers until they do not collide anymore.

4 Code

In this section we discuss our implementation of the topics discussed in section 3. In order to run the program on Linux you need to execute the run.sh. For a static output without animation execute run.sh without any argument. For an animated output (mp4) execute it with the parameter "mp4" or "gif". For more information, please turn to the Readme file alongside the code.

4.1 Helper classes

We introduced two small helper classes:

- A custom *Vector* struct to handle position and velocity. The only information it effectively holds is a x and a y value. In addition to that it also has overloaded operators to add, subtract and multiply two *Vectors*.
- An *IOManager* to write data into files. It has an overloaded stream operator.

4.2 Main loop

The main loop consists of a time variable t which is incremented by a value *TIME_STEP*, which equals dt in section 3.1, until t reaches *MAX_TIME*, then the loop stops. Inside this loop we implemented both the basic movement as well as the collision detection.

4.2.1 Basic movement

Following the formulas in section 3.1 we loop over all mass points twice. For every one of n^2 combinations we calculate the force acting on one mass point, calculate the resulting velocity and apply it to the position of this mass point.

4.3 BuildSpiral

This function was created to generate a cloud of points that should represent an asteroid. It was important to have a cloud of points so the asteroid could fall apart during the simulation.

We used the formula in polar coordinates: $r = a * \phi$ where $a > 0$ and ϕ is the angle. Then we transformed the polar coordinates to Cartesian coordinates

$$x = r * \cos(\phi)$$
$$y = r * \sin(\phi)$$

4.4 Gravity

This is a simple function to calculate the gravity by using Newton's law of universal gravitation

$$\vec{F}_G = G * \frac{m_1 * m_2}{r^2} * \vec{e}_r$$

4.5 Collision

Here we implemented collision detection and resolve the collisions. Both mass points get pushed away from each other by the same amount. To arrive at the correct velocities, we added/subtracted

4.6 Graphics

4.6.1 Graphics

To produce small videos about moons getting torn apart we use a small OpenGL library called jaogll (maintained as a personal project by Lars Gröber). It uses SDL2, GLM and GLEW which need to be installed before setting up the library.

4.6.2 GIF

Additionally to the non animated output we added a possibility for an animated output. For every timestep there will be a PNG created that will be converted later on to a mp4 or GIF with ffmpeg. (therefore you need to install the package ffmpeg on your computer).

5 Result

5.1 Toy example

In our first example we consider an unrealistic system of a satellite with a very low density and a planet with a very high density. Nevertheless this example shows quite drastically the effects of the tidal forces and how well aligned our simulation is in terms of conservation of energy and momentum.

For the simulation we used the following values (in arbitrary units):

- $M_{Planet} = 100000$
- $R = 10$
- $M_{Satellite} = 10$
- $r \approx 5$

This gives a Roche Limit of

$$d_{toy} = 5 * \sqrt[3]{2\frac{100000}{10}} = 135$$

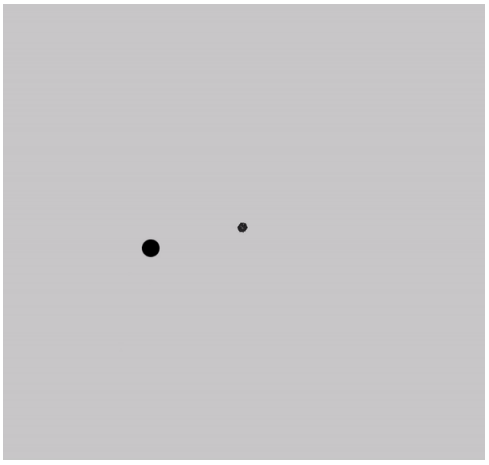

Figure 1: Toy example, animated url: goo.gl/DcQecJ

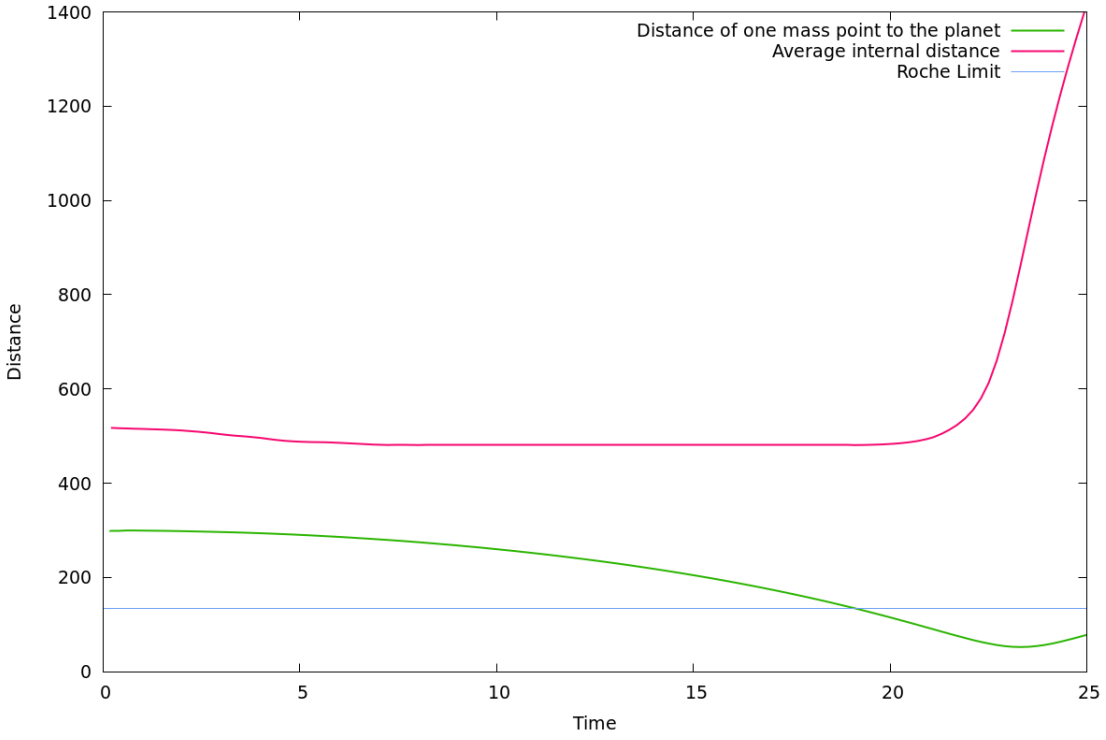

Figure 2: Distances in the toy example

During the simulation we kept track of several different quantities. The two most important can be seen in figure 2, namely the distance of one mass point (in the center of the satellite) to the planet and the average internal distance between all

mass points of the satellite (we averaged the distance between all mass points, not just the closest ones). One can clearly see the satellite being torn apart after crossing the Roche Limit. The average distance had to settle itself in the first interval until a time of 5.[1]

Another quantity of interest is of course the conservation of momentum and energy. Every frame we calculate the total momentum of the system by adding up the momentum of every mass point (including the planet). The energy is measured by adding up the kinetic and potential energy of every mass point. This gives the following graphs (the blue line marks the point at which the moon crosses the Roche Limit).

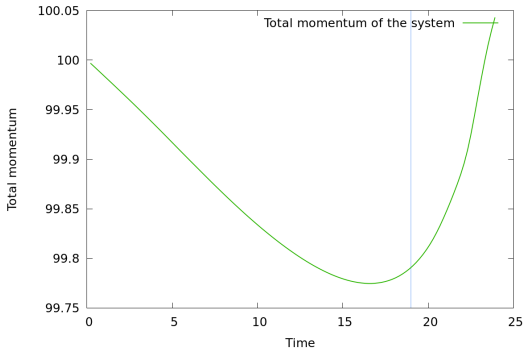

Figure 3: Total momentum of the toy example

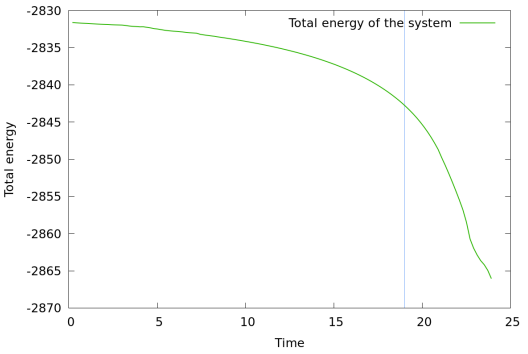

Figure 4: Total energy of the toy example

At first glance the differences in momentum and energy seem to be significant. But if one takes a look at the ranges, one can see, that the changes are minor (less than 1%).

[1] It should be noted here that all units are purely arbitrary, they are only defined by the gravitational constant, which we just set to 1 for this example.

5.2 Phobos - Mars

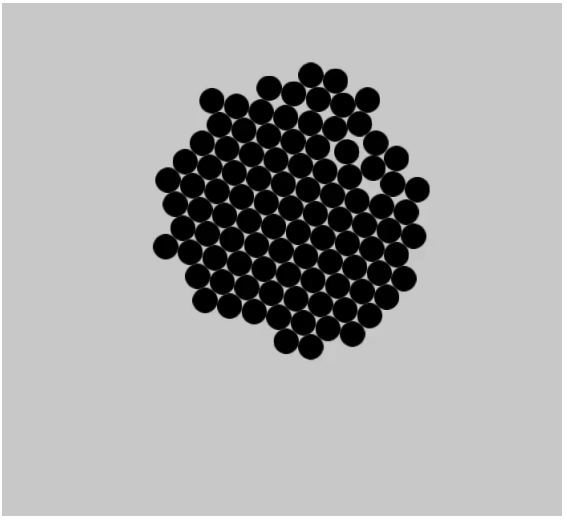

Figure 5: Phobos example, animated url: goo.gl/dedYUN

As a real example we looked at Phobos, a moon of Mars. It's distance to Mars is closer than any other known planetary moon, just about 6000km from the Martian surface. It is believed, that the moon will break up in orbit around Mars within about 30 to 50 million years. [2]

The important quantities are (this time in "correct" units):

- $M_{Planet} = 6.4 * 10^{23} kg$
- $R = 3390 km$
- $M_{Satellite} = 1 * 10^{16} kg$
- $r \approx 11 km$

Therefore the Roche Limit for Phobos is (measured from the center of Mars)

$$d_{Phobos} = 5543 km$$

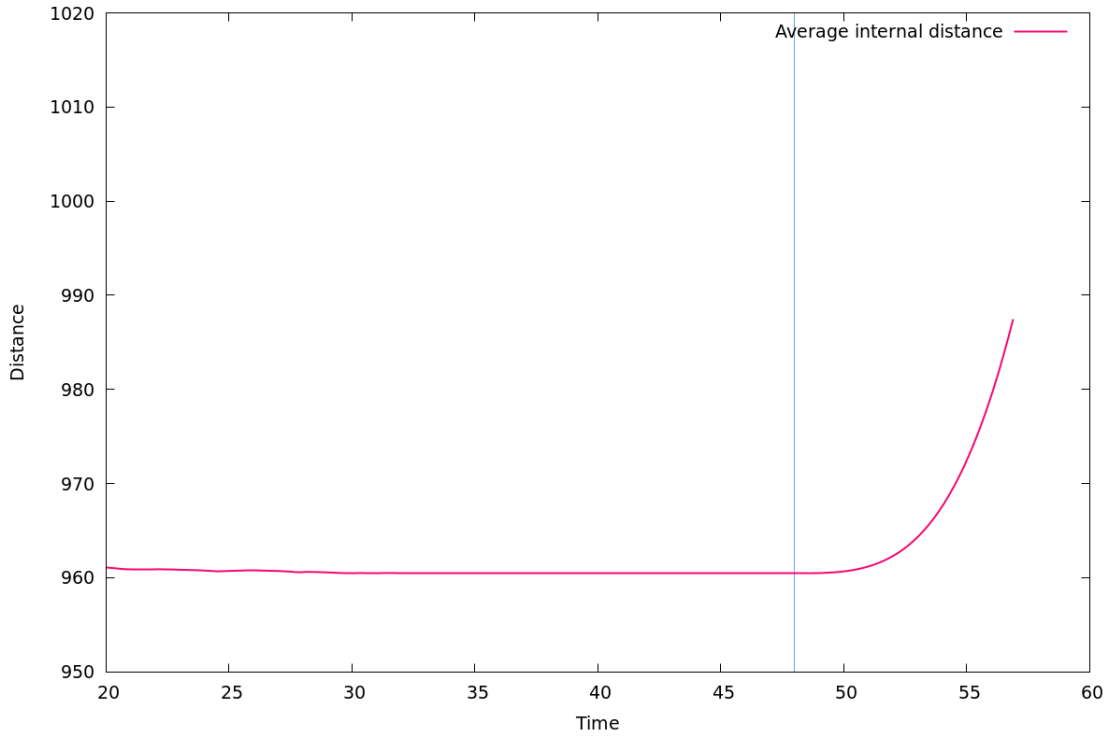

Figure 6: Internal distance of Phobos, the blue line marks the point at which the satellite crossed the Roche Limit

Again our simulation lies very close to the theoretical result. In the next figures we check if momentum and energy is conserved. Similar to the toy example, the differences are minor.

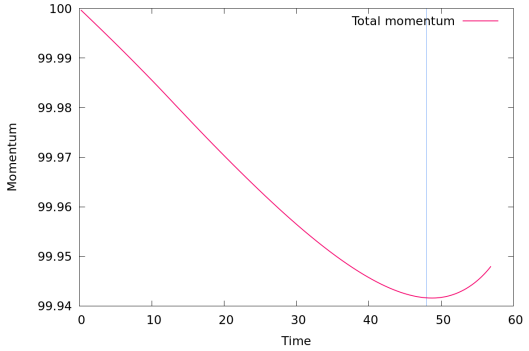

Figure 7: Total momentum of the system Phobos - Mars

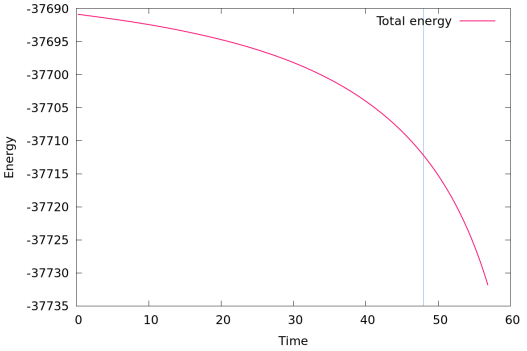

Figure 8: Total energy of the system Phobos - Mars

5.3 Discussion

The asteroid tended to form a non perfectly round body under the influence of gravity, so our results differ slightly from the theoretical Roche Limit as it assumes a circular asteroid with a known radius.
The system is loosing energy and has some momentum fluctuation. These also occur when using only one mass point, so they appear to be a result of the calculation of position and velocity.

References

[1] Derivation of the roche limit. https://en.wikipedia.org/wiki/Roche_limit.

[2] Information about phobos. https://en.wikipedia.org/wiki/Phobos_(moon).

[3] Midterm projects. http://itp.uni-frankfurt.de/~gros/Vorlesungen/CPP/midTermProjects.txt.