Experiment T

Tidal Disruption of a Comet

1 Introduction

At first glance our solar system seems rather empty with eight well separated planets orbiting a single mature star. On closer inspection, however, it is quickly apparent that our system is dynamically full with most stable orbits occupied by smaller objects like asteroids and comets. Current research suggests that many of these asteroids and comets are primitive and were formed at the same time as the solar system. Thus, studying the properties of these small solar system components may give us a window into how the planets formed and evolved into the solar system we see today.

According to our current theoretical models of planet formation, comets represent the building blocks of planets, planetesimals. The comets that we see in the solar system today are building blocks that never got incorporated into larger bodies and instead were chucked into cold storage by a close dynamical encounter with another large body like a planet.

Numerous studies of passing comets, such as Comet 67P/Churyumov Gerasimenko (Fig. 1) and Shoemaker-Levy 9 (Fig. 2) have found that comets have a very low mean density (lower than the density of water 1 g cm^{-3}), which indicates that there are large amounts of void space within the bodies. In addition, comets seem to have little or no tensile strength; they can easily be torn apart by tidal forces if they happen to have a close encounter with a large planet. The current evidence leads us to believe that a significant proportion of these bodies are *rubble piles* - loosely-bound aggregates of rock and ice, held together primarily by their own gravity with little material strength.

The goal of this lab is to successfully recreate the tidal disruption of comet Shoemaker-Levy 9 as Jupiter's gravity tore it apart in July 1992 and investigate the relationship between the disruption distance and the density and/or size of the comet. The pattern observed (see Fig. 2) is sometimes referred to as a 'string of pearls'. Numerical simulations suggest that it is a signature of the tidal disruption of a loosely bound object. You will be aiming to recreate this image and show that it is indeed a result of tidal forces acting on a loosely-bound aggregate.

This project will be done on a Linux operating system, which might be different from what you may be used to. Almost everything is done on the command line, which can be daunting for a new user. A lot of scientific computing is done on Linux/Unix systems, so this likely won't be the last time you use it! If you are new to Linux, you should complete the Unix tutorial at:
`http://www.star.bris.ac.uk/unixtut/index.html`
before moving onto the project itself. Note that you can run multiple simulations at the same on the Unix system - see the appendices for more details.

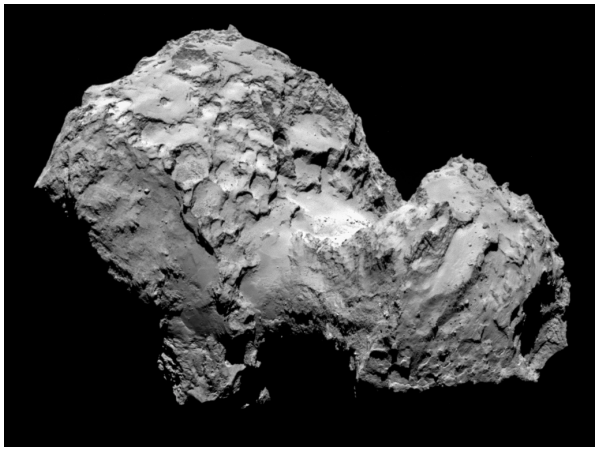

Figure 1: Comet 67P/Churyumov-Gerasimenko (ρ = 0.47 g cm^{-3}) imaged by the Rosetta spacecraft, August 2014.

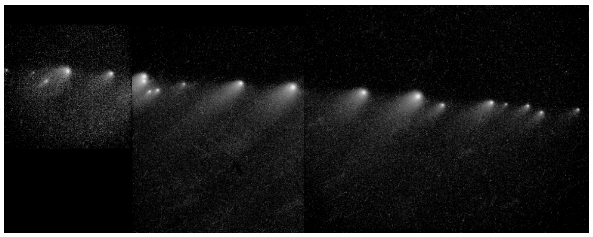

Figure 2: Comet Shoemaker-Levy 9, after tidally disrupting at Jupiter in July 1992. Photo taken by Hubble Space Telescope in March 1994, 4 months before the fragments collided with the planet. Calculated mean density was 0.5 g cm^{-3}.

2 Numerical Methods

In this project you will use a program called `PKDGRAV` to simulate the effects of gravity and physical collisions. `PKDGRAV` is a powerful and efficient *N*-body

integrator, used for studying the dynamics of objects under the influence of gravity across an enormous range of length scales. It can be used to model small asteroid orbits and collisions, planetesimal accretion, stellar cluster dynamics and galaxy simulations, all the way up to the evolution of galactic clusters.

The N-body problem is a computationally demanding one and has been studied for centuries (Newton studied the motions of three gravitationally interacting bodies in his *Principia*). The most basic/direct approach is to evaluate the force on each particle due to every other particle in the simulation and sum them all up to find the net force on that one particle. This is then repeated for every particle in order to advance one time step in the integration. This direct integration method, also known as Cowell's method, scales like $\mathcal{O}(N^2)$ which means that the number of calculations required to advance a single step scales quadratically with the number of particles in the simulation. Thus, the computing resources needed to run simulations with large numbers of particles quickly becomes impractical.

However, using a few careful approximations the scaling and performance can be vastly improved. `PKDGRAV` uses a hierarchical tree algorithm to approximate the gravitational force of distant particles instead of calculating gravitational force of every particle on every other particle. Using a hierarchical tree reduces the scaling of the gravity calculation from quadratic to $\mathcal{O}(N \log N)$, which is a little more than linear in N. Although the gravitational influences of distant mass is smoothed and approximated the hierarchical tree allows larger simulations with more resolution to be completed in a reasonable time frame. You might like to research this more, as it is a commonly used algorithm for N-body integration.

In addition to simplifying the gravity calculation `PKDGRAV` uses a relatively low order (second-order) leapfrog integrator in order to integrate the equations of motion of each particle. Although a higher order integrator would be more accurate the low order approximation of the leapfrog integrator takes less time. Thus, `PKDGRAV` is well suited for calculations involving large numbers of particles where close encounters and physical collisions with other particles will change the trajectories of particles frequently.

2.1 Create a simple rubble pile

Begin by downloading the archive containing the files and programs required for the experiment from blackboard. Create a folder in which you will store all the files required for and generated during this project, and extract the contents of the archive into this folder.

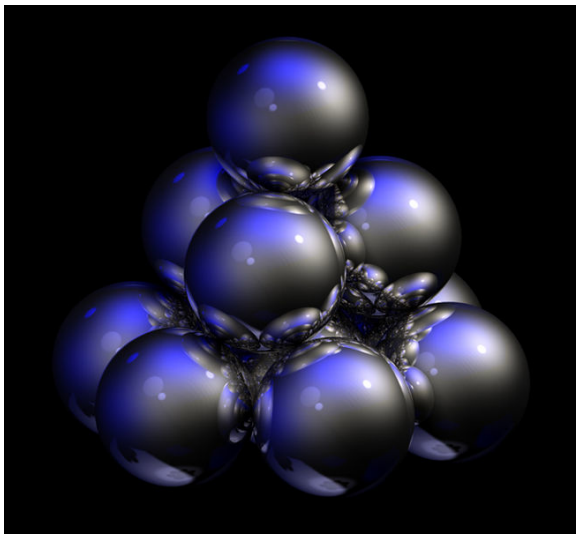

Figure 3: Eleven spheres arranged in a hexagonally close packed (HCP) fashion. HCP is one of the most efficient ways to pack equally-sized spheres. The maximum packing fraction is ~ 0.74.

The first step of your experiment is to create a rubble pile with a given radius, bulk material density, and a specific number of particles. Start by using `rpg` (rubble pile generator), which should be in your project folder.This simple generator only creates rubble piles consisting of identically-sized particles in a hexagonally close packed arrangement (see Fig. 3). The program `rpg` creates HCP rubble piles because it is simple to program and fast to generate.

In order to get a feel for the software this section will show you how to create a pile with `rpg`. Use the program `rpx`, also in your project folder, to create a situation in which two identical rubble piles collide with each other. Run a numerical experiment using `PKDGRAV` to simulate the collision and the aftermath, and create a movie file from the raw data so you can watch the simulation.

In your project folder you should see both `rpg` and a file called `rpg.par`. The latter file is a parameter file and is how you change the settings of the program `rpg`. Open this file to see what you can edit by typing `pico rpg.par`. `Pico` is a simple text editor on Unix systems, and we have given it the argument `rpg.par`. Therefore, this whole command tells the computer to open the file `rpg.par` in the program `Pico`.

The file will be opened on your screen. Here you can input the different parameters of the rubble pile you want to generate. Take some time to read the file and understand what the different parameters do. Enter some reasonable values for an asteroid or comet, and give it an output file name like `hcpcomet.ss`. Set the number of particles to about

100, and the radius to one kilometre. When you are finished, press CTRL+X to exit. It will prompt you to save your changes; type y and hit return. Don't change the filename, just press enter again. Now type ./rpg. Note that ./ is the command to run an executable file in your current directory, so typing ./rpg will run rpg. Then rpg will read your parameter file and output your hcpcomet.ss file. Note: You may get a "Permission denied" error when you try to run rpg or, indeed, any of the programs in the archive. Why is this? How can you fix it? (Hint: look at the commands lx -l and chmod).

You can now look at the rubble pile you have just generated. The files ssdraw and ssdraw.par should also be in your project folder. Open ssdraw.par in the same way you opened rpg.par. You can leave most of the settings as default. Set camera option and view option to 0, this fixes the camera to a specified location, and ensures the view is not centered on some particular object; set camera position and view direction to 0, 0, 1 and 0, 0, -1, respectively. These vectors define the position of the camera and the direction in which it looks. So this puts the camera on the z axis, looking in the -z direction. Set your sky vector to a direction not along the z direction. Set your starting view size to 0, this sets the scale of the image to the minimum required to display the input. Exit and save your settings.

Now type ./ssdraw hcpcomet.ss, or whatever you named your rubble-pile file. This will create a .ras file. Open this file by typing xv hcpcomet.ras, again replacing with whatever your filename was. This should open a picture of your rubble pile.

Now you will crash two rubble piles together. To set up the initial conditions, you will use the program rpx which should be in your folder.

1. Type ./rpx to run it. Take care when entering values a mistake will mean you have to run rpx again!
2. Choose **not** to use simulation units; it's easier to use MKS units for small objects like the two km-sized rubble piles you have generated.
3. You will be prompted to enter file 1. Type hcpcomet.ss.
4. The program will print a list of properties of the rubble pile, and prompt you to change any of them. Type 4 to change the centre-of-mass position.
5. Type y to specify absolute position. Type an offset in **kilometres**. Go for about twice the radius of your rubble pile to begin with, in the +x direction. When entering the vector components do so without commas or brackets, just separate with a space.
6. Now change the centre-of-mass velocity to a hundredth of your offset, in **m/s** (note that the velocity is given in m/s, and position is given in km), in the negative x direction. For example, if you chose a 2 km offset, you would enter -20 m/s.
7. Set the colour of this particle to green, and then enter 0 to continue.
8. You will again be prompted to add a file. Again type hcpcomet.ss, to add a copy of your comet to the situation.
9. This time, set the offset to -2 km, and the velocity as +20 m/s, and make this particle magenta.
10. Type 0 to continue, and then hit return to quit. Don't worry about re-centering the barycentric position for now, as you've already implicitly done this with your choices of position.
11. Some data will be printed about the situation you have set up. Check that they make sense, then type an output file name - make sure you don't leave it as the default, or it will overwrite your original rubble pile comet! Call it cometcrash.ss or something similar. Hit return and your data will be generated.
12. View the situation in the same way as you viewed your single rubble pile, except this time set starting view size to about four times your rubble pile separation.

Now we're going to pass the initial condition file to PKDGRAV in order to evolve the collision in time and crash the rubble piles together. Open ss.par with pico. Set nDigits to 5, nSteps to 500, and dDelta (your timestep) to the equivalent of one second when in units of years/2π (see appendix A for an explanation of these units). Make sure achInFile is your cometcrash.ss file, and set whatever output file prefix you want. Set iOutInterval to 1, and check that nSmooth is 16 or so. Save and exit this, and finally type ./pkdgrav ss.par to start integrating the simulation. This will probably take a while, as we are calculating the motions of two hundred colliding and gravitating particles at every single time step. You can gauge the progress of the integration by having your working directory open in another terminal whilst the integration is running, and simply typing ls every so often to see how many output files have been created.

Once the integration is complete, run the script `draw.sh`, by typing `./draw.sh`. This is a shell script that automates the process of passing in files to `ssdraw` and sticking them together to make a video. Again this will take some time, as image rendering and *N*-body simulations are two of the most computationally demanding things you can do! When complete, your video should automatically play. Finally, make a directory to store all of your raw data in to keep things tidy.

2.2 Model the tidal disruption of Shoemaker-Levy 9

Now generate a HCP rubble pile for use in your Shoemaker-Levy 9 simulation. Firstly you should research tidal forces and the Roche limit, and calculate what the Roche limit of Jupiter is, as well as Earth and the Sun.

Research comet Shoemaker-Levy 9, and make note of its predicted physical characteristics (see appendix B on using the Astrophysics Data System). Next you need to find out the orbital characteristics of the comet in July 1992, when it was disrupted, in order to run the simulation. NASA's Horizons ephemeris tool is a very useful resource for finding the orbital parameters of pretty much any object in the solar system, at any point in the past or future. The Horizons data for comet Shoemaker-Levy 9 is saved as a text file in the archive you have already downloaded. When deciding which time to choose as your comet's initial condition, you should consider when it is entering Jupiter's sphere of influence. (You could then make your life a bit easier by converting the 3D position and velocity vectors to 2D, recalling that 2-body gravitational motion is always planar - this is just helpful when deciding where to place the camera and how to set the viewing geometry in `ssdraw.par`).

Now you need a rubble pile comet. Create your rubble pile comet just like you did in the previous section using `rpg` and `rpx` but give the rubble pile the physical and orbital characteristics of SL9. `rpg` will spit out a `.ss` file, which is a binary solar system file and currently unreadable by you. In order to turn this into something you can read, find the program `ss2bt`. Run this on your `.ss` file, to turn it into a `.bt` file. This can then be opened in a text editor for viewing. The data is arranged in 14 columns, each corresponding to a particular value of that particle. If you generated say 100 particles, then there would be 100 rows of data, each corresponding to one of your particles.

In order from left to right, the values are:

Particle ID
org-idx
Mass
Radius
X position
Y position
Z position
X velocity
Y velocity
Z velocity
X spin
Y spin
Z spin
Colour

The particle mass has units of solar masses, positions and radii are in astronomical units, velocity is in AUs per year/2π, and spin is in units of radians per year/2π (see appendix A for an explanation of these units).

Note that `PKDGRAV` outputs a series of files which are also in the `.ss` format, although they are given a numerical extension indicating their order in the sequence rather than `.ss`. This means that, in addition to using these files with `draw.sh` to generate the animation, you can examine parameters of the simulation at any point by running `ss2bt` on the appropriate file. Furthermore, you can batch convert all the files from `PKDGRAV` by using `ss2bt` and appropriate file name wildcards. How would you do this? Take a look at the code of `draw.sh` if you need a clue.

Now go ahead and give your `.ss` file (containing your hcp comet with the initial conditions dictated by the ephemeris data) as the initial conditions to `PKDGRAV`. You'll need to edit the parameter file `ss.par`, which contains the options for running the simulation. Set `bHeliocentric` to 1 so that `PKDGRAV` will include a central potential in the gravity calculation, and set `dCentMass` to the mass of Jupiter (making sure you use the correct units). You should set the number of steps taken to around 500, and the time step to be about a tenth of the dynamical time, given by

$$\tau_d = \frac{R}{v_{esc}}. \quad (1)$$

where R is the radius of the uncollapsed rubble pile. Substituting in the expression for $v_{esc} = \sqrt{\frac{2GM}{R}}$, where M is the total mass of the rubble pile, we obtain

$$\tau_d = \frac{1}{\sqrt{2G\rho}}, \quad (2)$$

where G is the gravitational constant, and ρ is the bulk density of your assortment of particles. The dynamical time is the characteristic time of the rubble pile over which significant changes occur. Estimate this from the total mass of your particles relative to the region you generated them in. Make sure the parameter `nSmooth` is set to 16 or so; this is the parameter telling `PKDGRAV` how many particle neighbours to check for a collision.

Once all your timesteps have been outputted, check what is happening to the comet. Watch the `.mpeg`. Does it agree with observational data from the actual disruption? If everything is correct, you should see a *string of pearls*, the same formation seen after the comet was torn apart by tidal forces around Jupiter.

Now that you have successfully set up the tidal disruption simulation, think about what data you can collect to analyse and discuss. What are your sources of error for this type of experiment?

What happens quantitatively when you change the bulk density and/or the size of the comet? Why? It might be helpful to plot bulk density of the comet versus disruption distance. Does the disruption distance change with particle number (numerical resolution)? Does the coefficient of restitution have any impact on the disruption outcome? A possible extension to this lab would be to investigate the effects of different parameters in the simulation.

3 Appendices

A `PKDGRAV` system units

The parts of `PKDGRAV` that you will be working with use simulation units. These differ from normal MKS units, because they make the calculations involving astronomical objects easier. The mass unit is one solar mass, so all masses are expressed as a fraction of this. Similarly, the unit of distance is the astronomical unit. The unit of time is, somewhat unusually, the year divided by 2π, or about 5020000 seconds. Thus the unit for velocity is the AU per year/2π, and rotation rate is measured in radians per year/2π. You will need to make sure that when you're giving `PKDGRAV` your generated rubble pile data, the numbers in the `.bt` file are in these units. Otherwise your 10^{10} kg particle will actually be 10^{10} solar masses!

B Running multiple simulations

It is possible to run multiple simulations at the same time on the same system within Unix. Download the parameter files more than once and save them to different directories. Using the command line, run the simulation from within each directory. Make sure you label your files clearly to avoid confusion and record detailed notes on your chosen parameters for each simulation.

C Astrophysics Data System (ADS)

The ADS is a digital library portal for searching the astrophysical literature. Go to `http://adsabs.harvard.edu/abstract_service.html`. Use the `Abstract Words/Keywords` box to search for published journal articles on a particular topic. Use the `AND` button to join more than one string together. Under the `FILTERS` section choose `All refereed articles` in order to avoid unpublished work such as conference proceedings etc.

D Remote access arrangements

The cluster of Linux-based machines in 2.20, arranged to enable social distancing, is available for student use and the expectation is that normally you will work here. However, if this situation changes students may be required to work remotely. This appendix explains how you can continue this experiment under remote-working conditions. If remote working is necessary, you will be given access to a Linux-based server running the simulation software. Advice on how to connect to this is provided here. Connection will be via text-based Secure Shell (SSH) terminal software to issue commands and a Secure File Transfer Protocol (SFTP) client to move files between your area on the server and your local machine.

Computationally intensive tasks such as running the simulations and generating videos will be carried out on the server with the resulting output files transferred to your local machine for viewing and analysis. This is a commonly used approach when accessing high-performance computing facilities such as BlueCrystal. Using a text-based SSH terminal and SFTP is typically more robust over low-bandwidth connections than trying to use a graphical user interface remotely.

Note that all remote access will require you to connect via the University Virtual Private Network (VPN): `https://uob.sharepoint.com/`

sites/itservices/SitePages/vpn.aspx before commencing the steps below.

D.1 SSH Terminal

If you are using Windows, download and install puTTY: https://www.putty.org/. Note that on University managed machines this is probably already installed or available from the Software Centre. Launch the application, in the "Session" category set the connection type to "SSH" and the host name as calgary.phy.bris.ac.uk. The other default settings should be correct. On clicking "Open", a terminal window should open where you can log in with your University user ID and password. You may be prompted with a security warning that the host certificate is not trusted, but accept it anyway. You should then find yourself logged in to your user area on the server.

If you are using Linux or MacOS, an SSH client should be available. Simply open a terminal window and type: ssh *user*@calgary.phy.bris.ac.uk where *user* is your University user name. You will be prompted to supply your password and then logged in to the server. Again, you can accept any supposedly suspect certificates you receive security warnings about.

If you are unable to log in, first check that you are correctly connected to the VPN then contact the lab technicians to ensure you have been given an account on the server.

D.2 SFTP File Transfers

If you are using Windows, MacOS or Linux you can download FileZilla: https://filezilla-project.org/. Note that on University managed machines this is probably already installed or available from the Software Centre. Just below the menu bar, in the "host" field enter sftp://calgary.phy.bris.ac.uk followed by your username and password then hit the "Quickconnect" button. You should see a series of messages as the connection is made and the file systems on the local machine and server should appear in the panels below. You can now simply drag and drop files between the two.

If you are using Linux (and possibly MacOS) you can also use a terminal-based sftp client. Open a terminal window and type: sftp *user*@calgary.phy.bris.ac.uk where *user* is your University user name. You will be prompted for a password and then see an sftp> prompt once logged on. Typing help will get you a list of commands but, briefly, normal Linux shell commands such as ls, cd, pwd etc. navigate the remote filesystem while the same commands prefixed with "l" (lls, lcd, lpwd, etc.) navigate the local file system. The get and put commands are used to transfer files from and to the remote filesystem, respectively.

D.3 Viewing Results

The .ras file format generated by ssdraw is a little unusual. It will open in Irfanview on Windows and GIMP on most platforms. If you wish to convert it to a more common format such as .png or .jpg, this can be done on the server using the ImageMagick convert command, e.g. convert hpcomet.ras test1b.png. This seems to throw an error but still creates a valid file. To view the .mpeg movies, virtually any movie player will work. VLC player is quite good and available on all platforms.