N-Body Simulation

You are tasked with constructing an application for an n-body simulation. This is a common astrophysics simulation, modelling orbiting bodies within our solar system and galaxy. Your application will simulate the orbiting patterns of celestial bodies within your system. This may be set data provided to you via command line arguments or randomly generated particles.

On top of modelling this application, you will need to write a report and produce a graphical application which will visualise simulation using the `SDL` library and render the data. You will be provided sample test data involving four bodies.

You should aim to not have any locks in your codebase and your application should clearly benefit from multiple threads processing it.

Modelling The Data

Each `body` is given the following fields. Each x, y, z field relates to a point in space. Each body has a velocity, with each part broken into different components. You will use the current velocity which will update the coordinates every iteration. Each body will have a mass associated which will change the velocity of other bodies.

```
struct body {
    double x;
    double y;
    double z;
    double velocity_x;
    double velocity_y;
    double velocity_z;
    double mass;
};
```

You can use the following constants within your simulation as part of your data.

```
#define PI (3.141592653589793)
#define SOLARMASS (4 * PI * PI) #Use for deriving your own test cases
#define NDAYS (365.25) #Use for deriving your own test cases
#define GCONST (6.67e-11) #Gravitational Constant
```

With the data you should be able to represent each element within the simulation. You are free to modify the `body` struct as part of optimisations and fixes to your application. Prioritise a correct result over execution time.

Simulation

The main function you will need to implement will be responsible for controlling the simulation with a change of time (***dt***) as an argument. You will need to implement the following function within your application. This function will advance the simulation with time difference (***dt***).

```
void step(...)
```

You will need to document your `step` function in your repository's `README.md` file. You will need to calculate the distance and magnitude between different bodies within the simulation. The difference between points is represented through the symbols ***dx, dy, dz***. You can calculate the distance between two bodies using the following formula.

$$distance = \sqrt{dx^2 + dy^2 + dz^2}$$

dt is the difference in time and is input to the `step` function. You will need to calculate the magnitude for the next operation to update the velocity of body. Use the gravitational constant (**6.67e-11**) in your calculation and simulation.

$$magnitude_{ij} = G\frac{mass_i mass_j}{distance^2}$$

You will need to construct a unit vector between ***i*** and ***j***, which you use for the direction.

$$\hat{d} = (\frac{dx}{distance}, \frac{dy}{distance}, \frac{dz}{distance})$$

You will need to calculate acceleration vector using your newly acquired unit vector, magnitude and mass. (Do note, you can apply a to either i or j, it depends on what)

$$a = \frac{\hat{d} * magnitude}{mass}$$

Afterwards you will need to apply changes to the current body's velocity from all other bodies within the system. Based on the current time interval (***t***), to calculate the velocity based on the next time interval.

$$velocity_i(t+1) = velocity_i(t) + \sum_{j!=i}(a_x * dt, a_y * dt, a_z * dt)$$

Do note, while in this context, you may want to apply changes to the ***j***'s velocity. Afterwards, you will need to update the points of all bodies within the system.

$$position_i(t+1) = position_i(t) + velocity_i(t) * dt$$

You will need to implement the following function and extract the energy data from the simulation. The total amount of energy should result in roughly the same value (conserve total energy, however it may fluctuate within the simulate, note the fluctuation in your final report) at the simulation and you can use this function to verify the starting energy value with the final energy value. You may encounter some strange issues with your setup, generation or depending how long it computes, your energy function may return an invalid value. In your report, attempt to explain any cases that exist that impact the simulation.

```
double energy(...)
```

Energy within the system calculated with the following formula. Use as part of your test suite to ensure that your application is correct to the best of your abilities. (Alternatively, you can extract the kinetic energy for all elements and extract potential entity for entities to get the result you want)

$$e = \sum_{i=0}^{N} \left(\frac{mass_i velocity_i^2}{2} - \sum_{j=i+1}^{N} \frac{G mass_i mass_j}{magnitude_{ij}} \right)$$

Your command line application will be invoked with the following command line arguments. The command line program can support `-b` or `-f` flags, `-b` will correspond to the number of randomly generated bodies while `-f` flag corresponds to a file that contains particle data. The second argument requires the change in time value to be used for each step of the simulation.

```
./nbody <iterations> <dt> (-b <bodies> | -f <filename>)
```

Example of executing the program with 50 randomly placed celestial bodies and will iterate 3000 times.

```
./nbody 3000 0.01 -b 50
```

Each iteration will correspond to a change of time of `dt` within the simulation as specified by the command line argument. Below is an example of executing the program with a file containing data and will iterate 3000 times.

```
./nbody 3000 0.01 -f data.csv
```

Your program will need to support the following file format when the `-f` flag is provided. The application will need to handle a filename and load the data in the following format.

```
<x>,<y>,<z>,<vx>,<vy>,<vz>,<mass>
```

GUI

Once you have successfully constructed the simulation and have set up a command line application. You are required to construct a GUI that will visualise the simulation. You have been given an SDL template that will allow you to get your code working. You can access the SDL documentation from the following website https://wiki.libsdl.org/.

You will need to support the following command line arguments that will be in the following form.

```
./nbody-gui <resolution_width> <resolution_height> <iterations> <dt> (-b <bodies>
| -f <filename>) (scale)
```

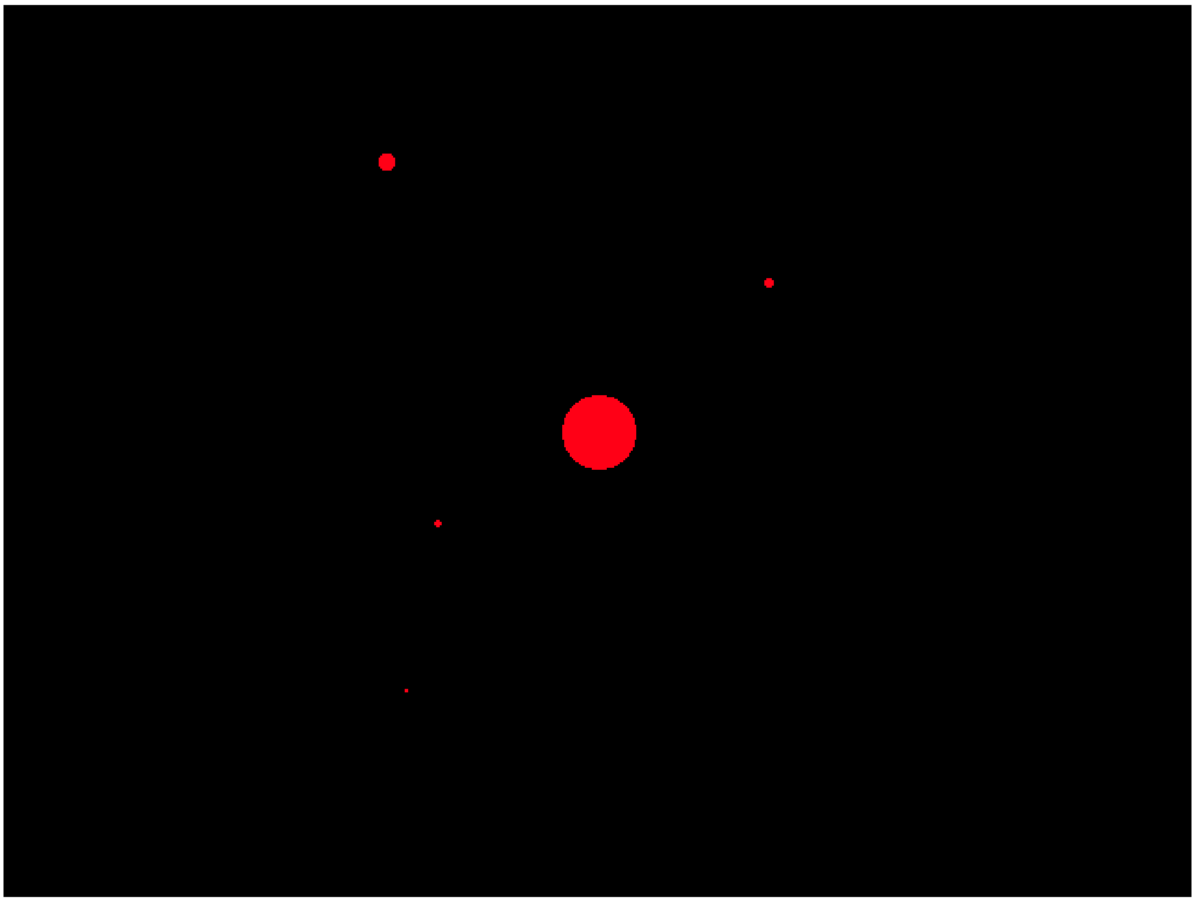
Below is an example of executing this application with 400 bodies randomly scattered.

```
./nbody-gui 800 600 50000 0.1 -b 400 0.00025
```

Like the command line application, your GUI program will be invoked like so.

```
./nbody-gui 800 600 50000 10 -f data.csv 0.00025
```

Each frame corresponds to a time change of `dt` within the simulation. Each body is represented by a simple red circle which the radius will be clamped to 10 pixels or below. Scale the . Below is an example screenshot of the GUI demo. However, you can describe a method for scaling planets within your application. (Picture below violates 10 pixel radius)



You will need to ensure you compile and link to `-lSDL2`. All platforms have access to this library that you will need to install use through out your assessment. Template will be accessible from the resources section of Ed.

Submission (11:59PM 31/10/2020)

You are required to submit source code and a report, documenting your development process, identifying issues with your simulation and how you addressed them. Please state any errors, deviations or issues you encountered during this project in your README.md file.

Your build script should provide the following scripts.

- Command line application
- GUI Application
- Test Suite Execution
- Benchmark

Your application will need to be compiled with the following command.

```
gcc -O0 <source files> -o nbody -Wall -Werror -Wvla -lm -lpthread
```

Your code submission must be submitted to a `github.sydney.edu.au` repository with the name `soft3410a1`. Please invite your tutor to your repository so they can review and assess it.

GUI Application Submission (2%)

Your GUI application submission must link correctly to `SDL` and provide a two-dimensional representation of the simulation.

- Links correctly to SDL
- Constructs and launches a window that will animate system of bodies.
- The application must execute in real-time, try to ensure that your application will run efficiently as a GUI animation.

Command Line Submission (4%)

Your command line submission must be included in your repository and meet the following expectations.

- You will need to construct the simulation and correctly calculate the energy of the system.
- Allow the change of threads and the number of iterations is determined by command line arguments.
- You provide an adequate test suite which tests different number of iterations, number of bodies and number of threads.
- Provides a compilation script with a makefile.

Report (4%)

Your report will need to meet the following expectation.

- Document your findings and development process.
- Identify parts of your application that take the majority of execution time.
- Identify memory accessed and how you have optimised access of data and minimised cache-misses.
- Graph and visualise the execution data of your simulation.
- Compare different approaches within your simulation and how you have increased the performance of your application.
- README.md within your repository describing how your tutor can build your application.

Your report must submitted via Canvas under the **Assignment 1** submission portal. You must upload a *.pdf* file.

Academic Declaration

By submitting this assignment you declare the following

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended).

These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

*I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking