A Project Report

On

N Body Simulation

BY

A V S DHEERAJ

2015B5AA0600H

Under the supervision of

RAHUL NIGAM

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

PHY F376 Design Project

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(NOVEMBER 2018)

ACKNOWLEDGEMENTS

I would sincerely like to thank a few people without whom this great opportunity for learning would have never been realized. Firstly, my project guide, Rahul Nigam whose relentless pushes were integral to keep me focused and disciplined. There were quite a few times when the outcome was not productive, and dismay set in. Rahul Nigam never quite gave up on me and his sound advice which so close to reality helped me to get the project back on track.

Lastly, my heartfelt thanks to my family and friends who supported me and had faith which helped me.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "N Body Simulation" submitted by A V S DHEERAJ (ID No. 2015B5AA0600H) in partial fulfillment of the requirements of the course PHY F376, Design Oriented Project Course, embodies the work done by him under my supervision and guidance.

Date: 30/11/18 **(RAHUL NIGAM)**

BITS- Pilani, Hyderabad Campus

ABSTRACT

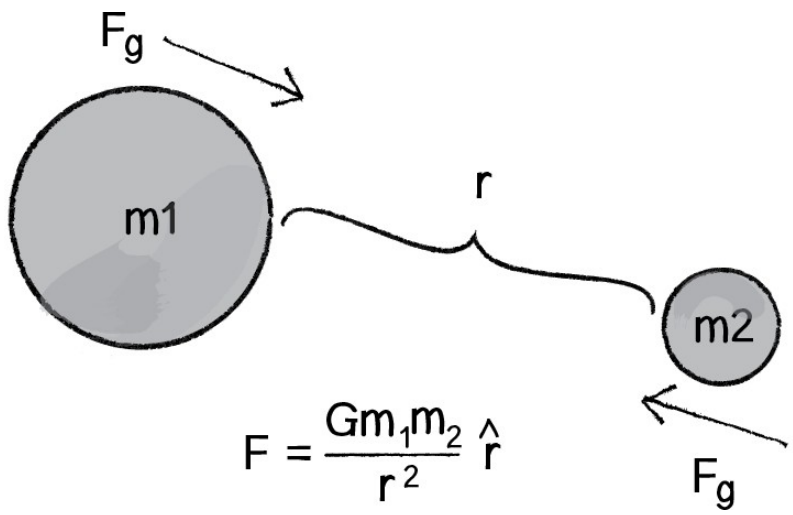
We always wanted to know how our galaxy is formed, how all the stars and planets follow only a specific order that they are meant to. The N-body problem tries to predict the motion of "N" objects under the mutual gravitation. The "N" bodies are simulated in virtual environment and their behaviour is observed. These simulations provides us valuable information like how the system has evolved overtime, the energies of each particles, the formation of clusters, etc.

This paper tries to build a N-body simulator from scratch. We try to apply the best algorithm that suits our purpose. We try to Study the energy of our system as it evolves with time, build a visualizer to view our particles, change the number of particles and change the initial formation of particles from our code.

CONTENTS

Approach to Simulation

Force on each particle is calculated by summing up the forces due to all other particles. Let us consider 4 particles M1,M2,M3,M4. The forces due to all M2,M3 and M4 are calculated and vector added to find the resultant force on M1. Then the acceleration can be computed from dividing the force by mass. This acceleration is used to change the resultant velocity vectors by the formula.V = V + a * delta_t. Where delta_t can be set by the user. Then this velocity is used to update the position vectors by the formula X = X + V * delta_t. These calculations are done for every particle and the final positions are stored to a file for every delta_t. This method is called the Particle-Particle method.The time complexity of this algorithm is $O(n^2)$.There are many other algorithms like tree codes ,Barnes hut algorithm,particle mesh method etc but they lack in accuracy as the P-P method.We try to implement the p-p method for its accuracy and simplicity.



Implementation

Trail1:

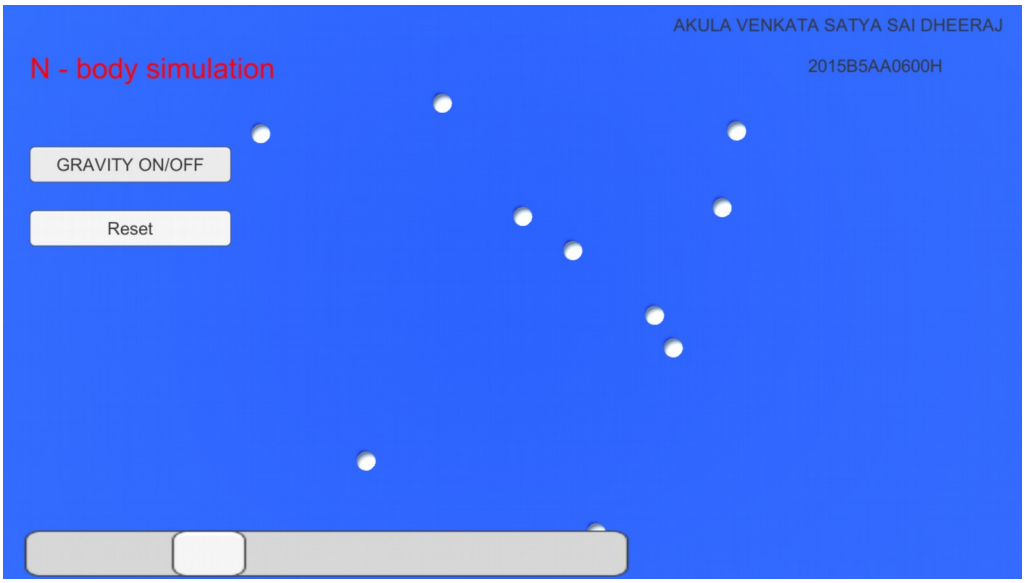
What is unity:

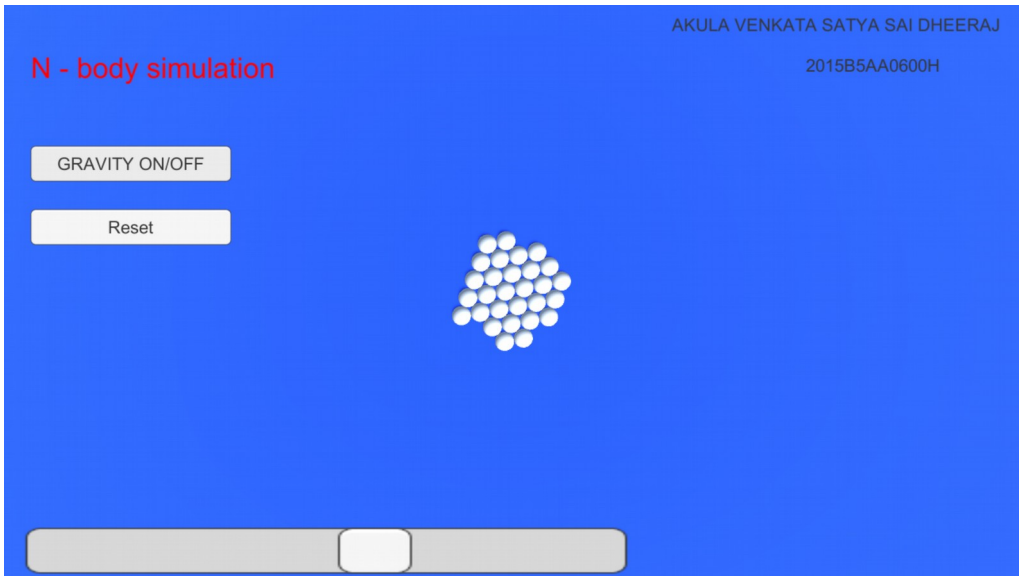
“Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc. Worldwide Developers Conference as an OS X-exclusive game engine. As of 2018, the engine has been extended to support 27 platforms. The engine can be used to create both three-dimensional and two-dimensional games as well as simulations for its many platforms.**”**

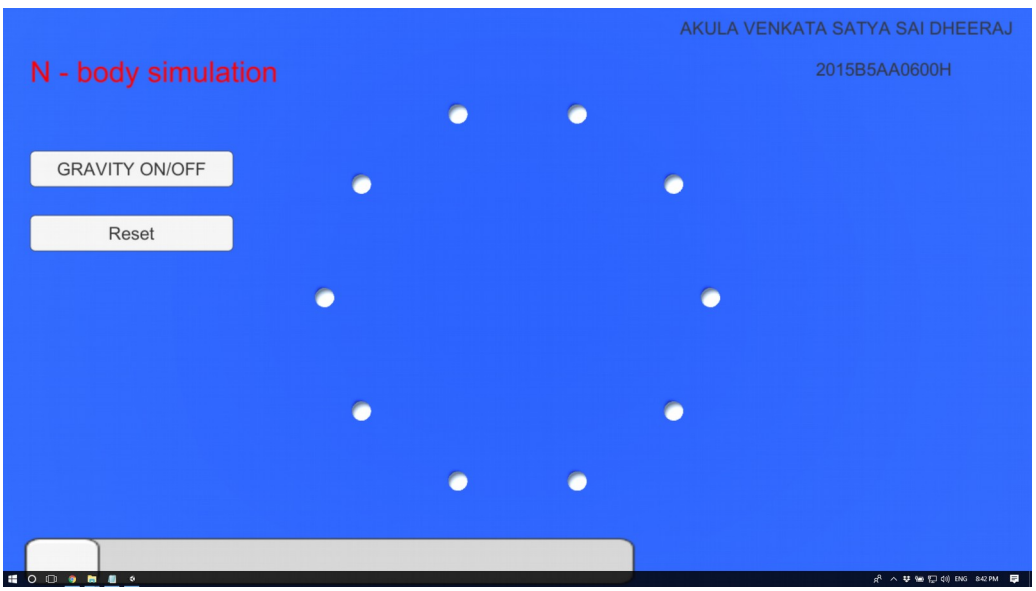
This is a direct quote from wikipedia about **unity** the game engine I used to simulate.

At first I implemented this approach in unity using the c# language.The position vectors are calculated in real time and then rendered.This caused frame lag after 50 particles because we need to calculate the results in real time. That means our frame rate would depend on our no of particles.This approach was not feasible since it provided variable results depending on no of particles.

I hereby attach the snapshots of the simulation done in unity on next page.The simulation is done completely in 2d. All the code is written from scratch no modules are used.


AKULA VENKATA SATYA SAI DHEERAJ
N - body simulation
2015B5AA0600H
GRAVITY ON/OFF
Reset

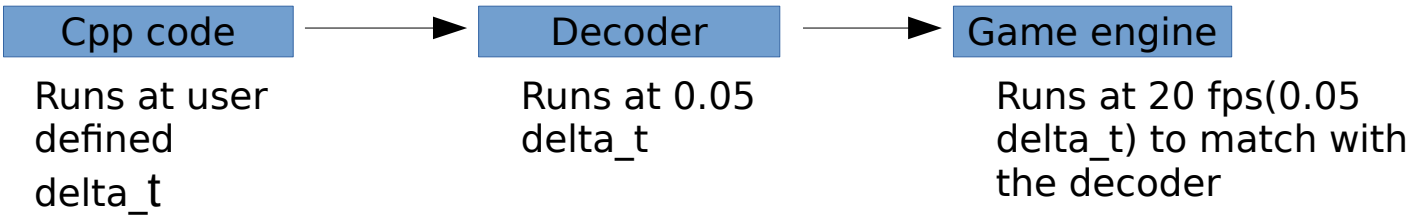

AKULA VENKATA SATYA SAI DHEERAJ
N - body simulation
2015B5AA0600H
GRAVITY ON/OFF
Reset


AKULA VENKATA SATYA SAI DHEERAJ
N - body simulation
2015B5AA0600H
GRAVITY ON/OFF
Reset

Time for Tinkering:

The problem with the previous method is that the frame rate is depending on number of particles to overcome this problem I came up with an alternate solution.First, we use a script to generate the simulation at a fixed delta_t. Then this is send to a decoder which runs at 0.05 delta_t which produces output at 0.05 delta_t. Decoder is nothing but a abstraction in our code which takes a input and converts it into output by taking the relevant input and excluding the undesired one. This ensures that the renderer or game engine runs at a constant frame rate.

The first script part is realized by a naive cpp code and the decoder as a function in this code.



Trail2:

What is Godot:

“Godot is a 2D and 3D cross-platform compatible game engine released as open source software under the MIT license. It was initially developed for several companies in Latin America before its public release.The development environment runs on Windows, macOS, Linux, BSD and Haiku (both 32 and 64-bit) and can create games targeting PC, mobile and webplatforms.**”**

This is a direct quote from wikipedia about godot the game engine I used in my trail 2.

The hunt for open source (The dissapointment that followed):

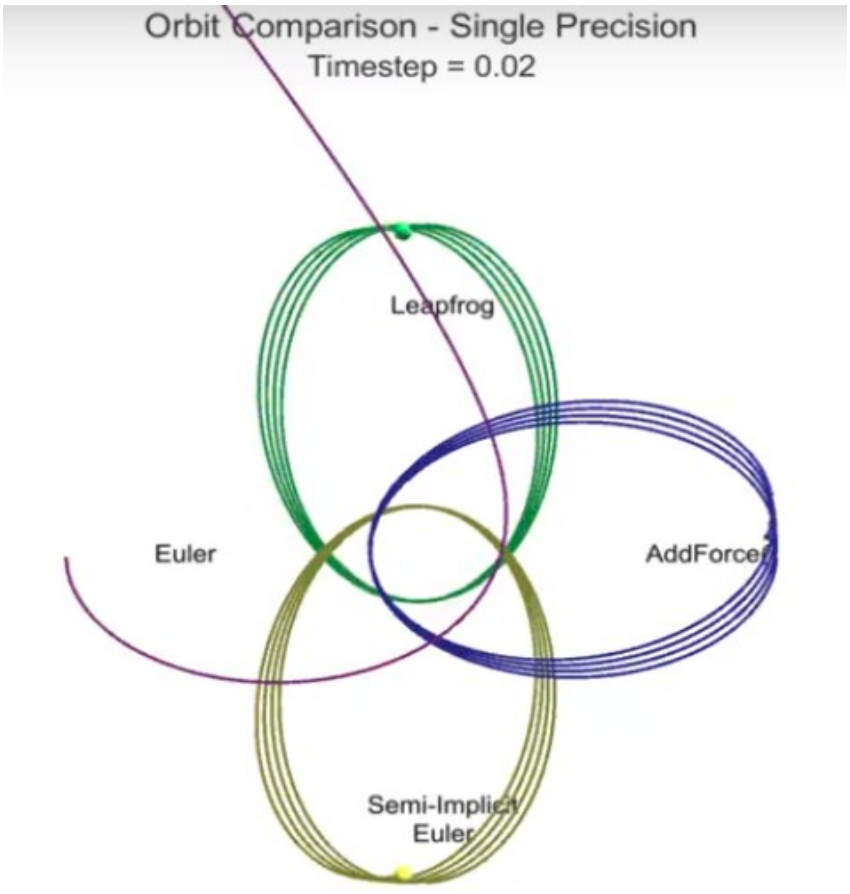
This time the **godot game** engine was choosen. The main reason for changing the game engine is that unity source code was private and godot on other hand is open source.Godot is a unflourished engine and presented many problems while integrating with the nbody problem.This took most of the valuable time of the project.Godot even has its own programming language called GdScript.

Other difficulty faced during this approach is that while writing to a file the code must be structured so that the switch between io cycles and cpu cycles must be minimum.I did not know this and proceeded without caution which damaged my hardrive.(since there were more than 10^6 io/cpu switches carried out in a single run in my original code).The tip of hardrive should get turned on and off 10^6 times .....

Algorithms for updating velocity and position

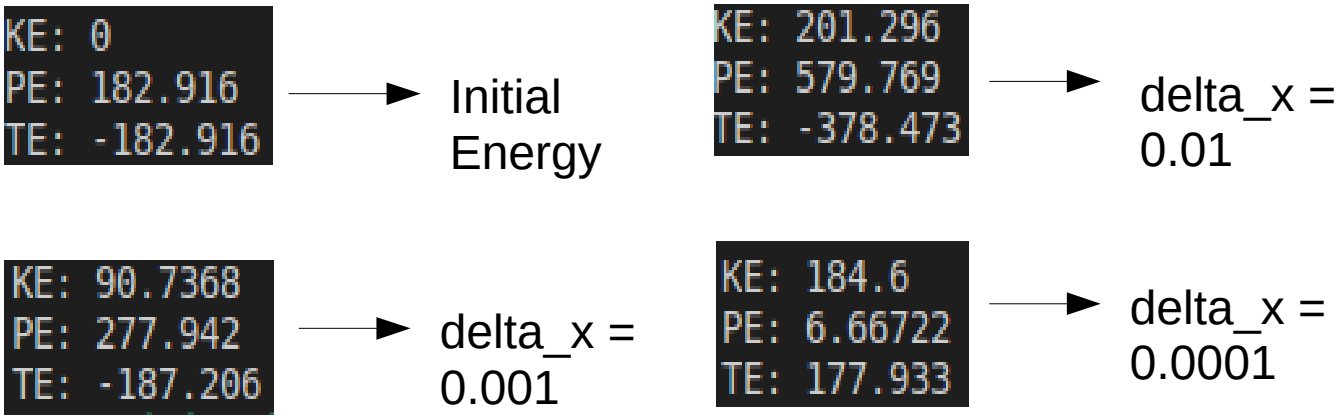
```
// Euler update
r = r + v * dt;
v = v + a * dt;
//Semi Implicit Euler
v = v + a * dt;
r = r + v * dt;
// Leapfrog
v = v + a * 0.5 * dt;

r = r + v * dt;

v = v + a * 0.5 * dt;
```

We could find that the Euler is worse because it is not even a closed orbit. Where as leap frog and semi implicit Euler give almost similar results. So i went with semi-implicit Euler which is giving pretty accurate results for less complexity.



Energy calculations:

If we use ideal formula of gravitation the Energy is not being conserved .This is because when particles become close to each other the denominator of the ideal gravitational force tends to zero and acceleration tends to infinity ,to prevent this from happening we introduce a small change in the denominator of the actual gravitational force formula we add it by a factor delta_x.
To conserve energy delta_x must be such that it must not be too small and not too large if it is too small the ke of the system is abnormally high.
By trail and error I founded that keeping delta_x around 0.001 is yielding best results.
Which was in fact the delta_t of the system this was an eureka moment of this project which suprised me.After rigorous calculations i found out it must be infact delta_t to reduce the error.

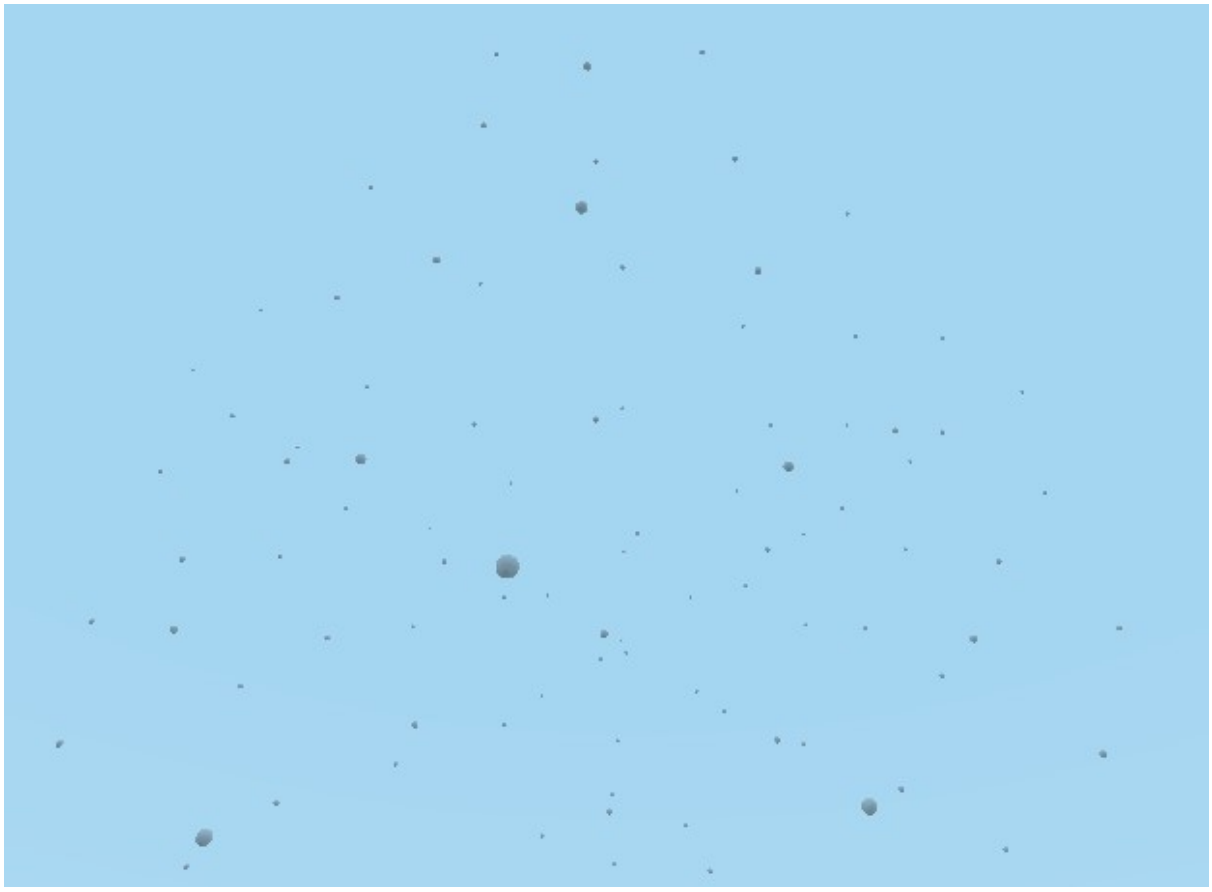


Initial Formations:

If we give a perfect cube as input to the no of particles then the initial formation will be a cube. If we give input which is not a perfect cube then it forms a random structure.

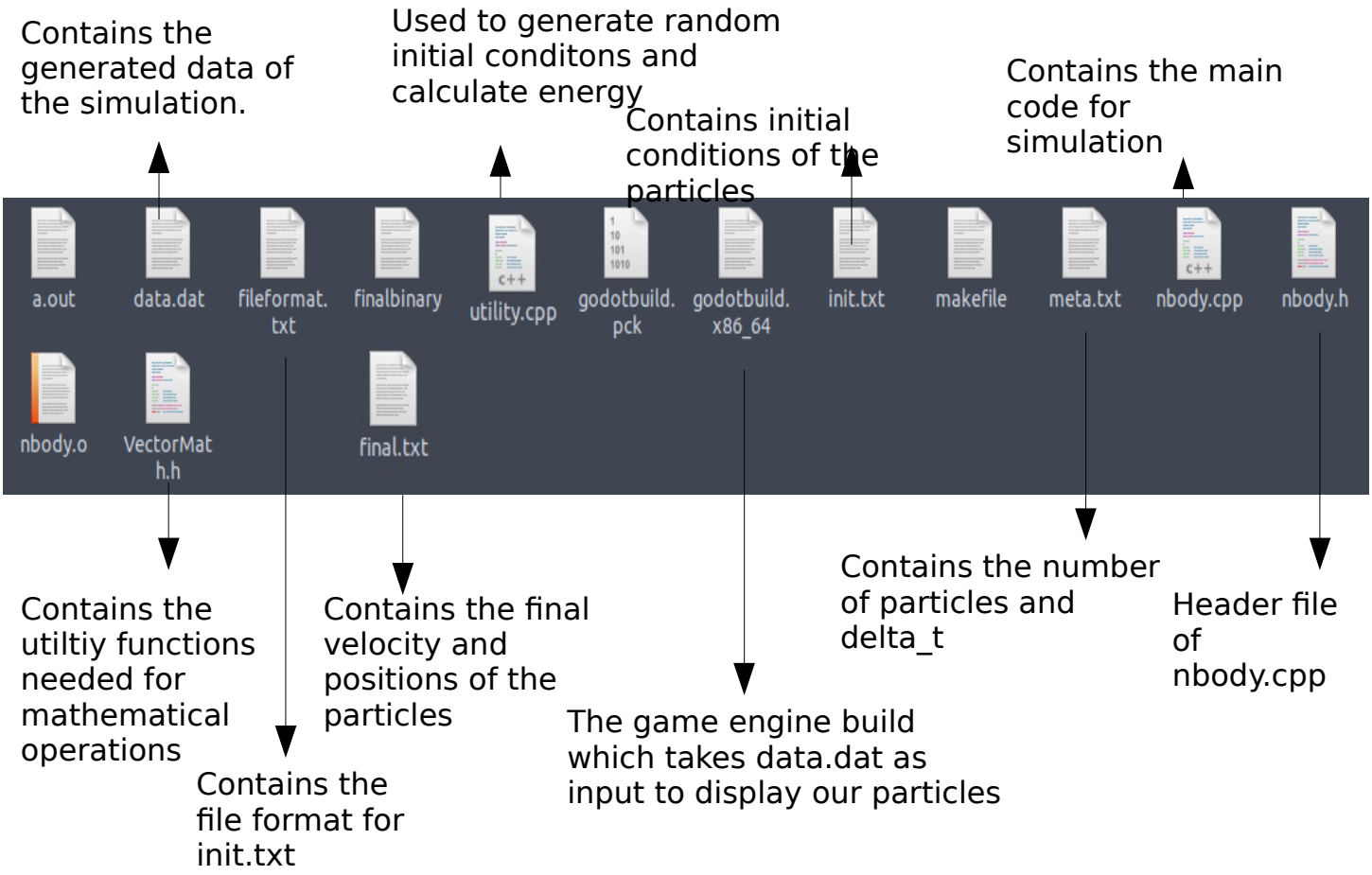
An interesting phenomena is observed when the initial formation is a cube a number of particles tend to form a cluster. To increase the number of clusters I added a loss of energy term in the code when the particles are very near to each other. This did not increase the number of clusters but the no of particles in the clusters increased. The Total energy of the system took a toll after adding the loss term there is significant change in the total energy after our default time = 100s.

Below is the picture of n = 125(5^3) particles:



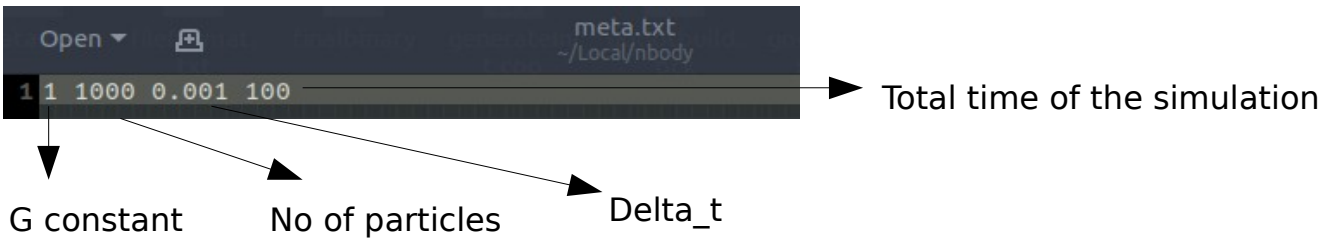
Project Structure & where to get the code:

Structure:



Steps to run the project:

1)Edit the meta.txt as per the simulation conditions.



2)Generate the init.txt file by running a.out.

```
dheeraj@jarvis:~/Local/nbody$ ./a.out i
```

This generates the init file with initial positions and mass of each particle assigned as random values.

```
2
6 7 5
0 0 0
6
5 6 2
0 0 0
```

Mass of each particle
Position of each particle in x,y,z direction
Initial velocity of the particle

```
<Gravitational Constant> <Number of bodies(N)> <Time step> <Time>
<Mass of M1>
<Position of M1 in x,y,z co-ordinates>
<Initial velocity of M1 in x,,y,z components>
...
<And so on for N bodies>

x,y,z denote position components in 3-space, vx,vy,vz denote velocity components)
```

Format of the init.txt file

3)Run the simulation by running final binary or typing make in your command line.This generates the data.dat file.

```
dheeraj@jarvis:~/Local/nbody$ make
```

```
5.9760 6.9241 4.9960
4.9857 5.9668 2.0676
1.0925 2.0474 6.9520
8.8606 3.0433 5.9644
5.9910 2.0791 5.9731
```

Position of first particle in x,y,z direction
Position of second particle in x,y,z direction

Position of n-th particle in x,y,z direction

4)To visualize the final results in 3d space run the godotbuild.x86_64.

```
dheeraj@jarvis:~/Local/nbody$ ./godotbuild.x86_64
```

5)To Calculate Energy run ./a.out with arguments as ‘e’ and [name of the file]

```
dheeraj@jarvis:~/Local/nbody$ ./a.out e init.txt
into calculate energy
init.txt
KE: 0
PE: 3.61814
TE: -3.61814
dheeraj@jarvis:~/Local/nbody$ ./a.out e final.txt
into calculate energy
final.txt
KE: 0.830604
PE: 4.4509
TE: -3.6203
```

Where to get the code(open source MIT LICENSE)

The entire code is found at Github.

https://github.com/newb-7/nbodysimulation.git