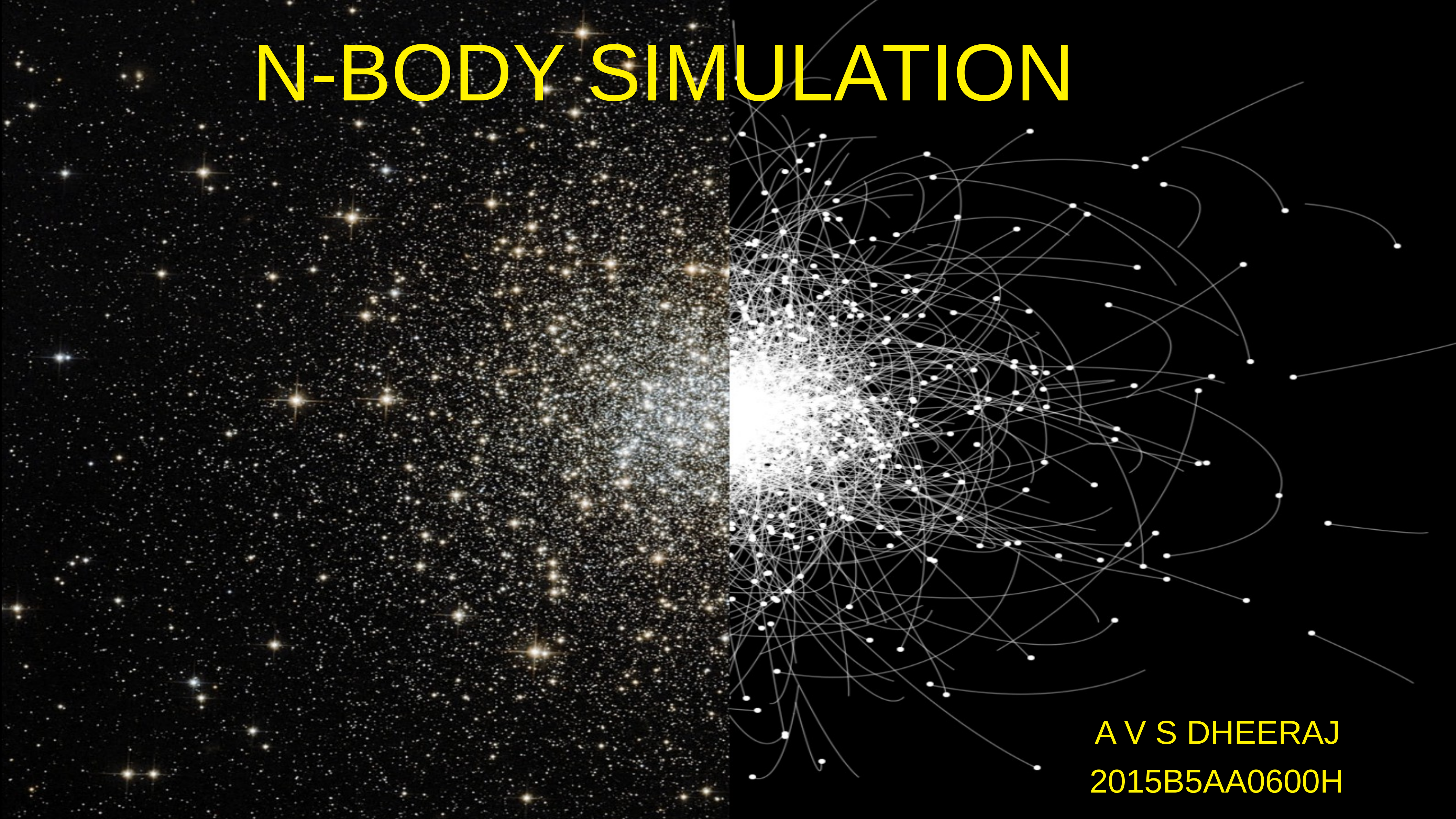
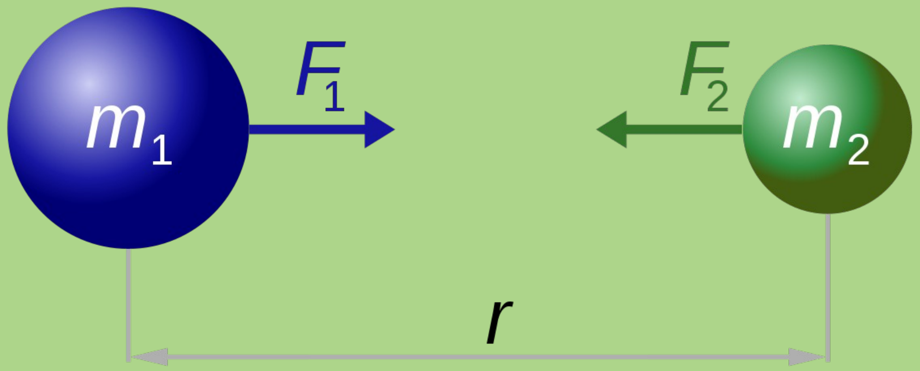
N-BODY SIMULATION



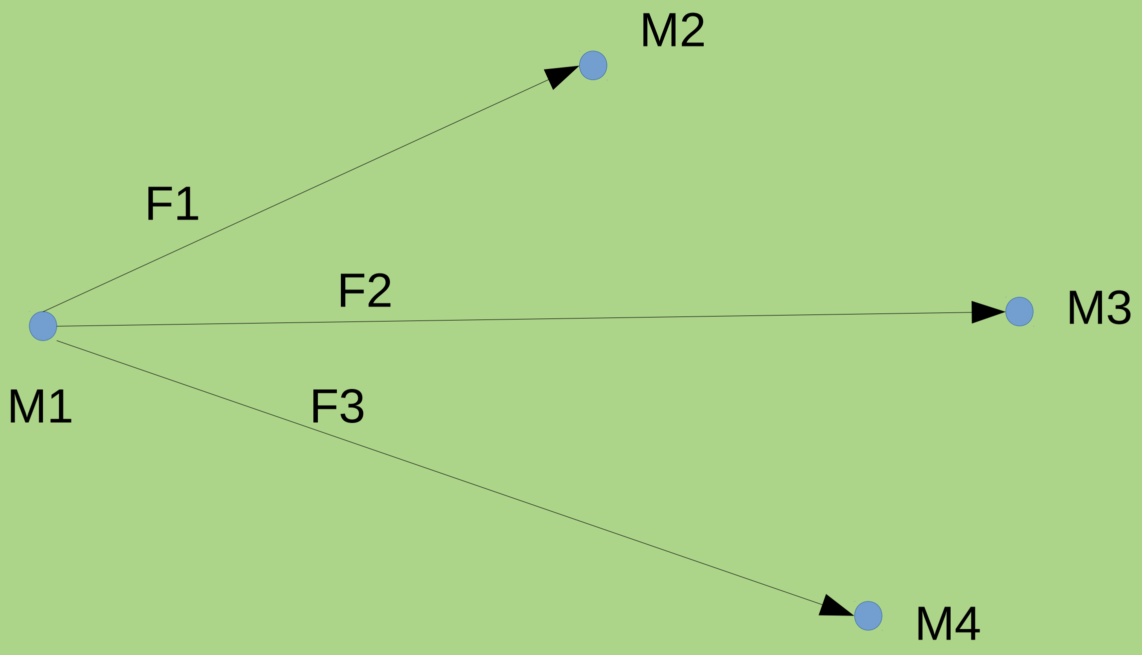
A V S DHEERAJ
2015B5AA0600H

APPROACH TO SIMULATION:

Force on each particle is calculated by summing up the forces due to all other particles.



$$F_1 = F_2 = G\frac{m_1 \times m_2}{r^2}$$



M1 calculations:

The forces due to all M2,M3 and M4 are calculated and vector added to find the resultant force.Then the accelaration can be computed from dividing it by mass.

This acceleration is used to change the resultant velocity vectors by the formula
V = V + a * delta_t
delta_t can be set by the user.

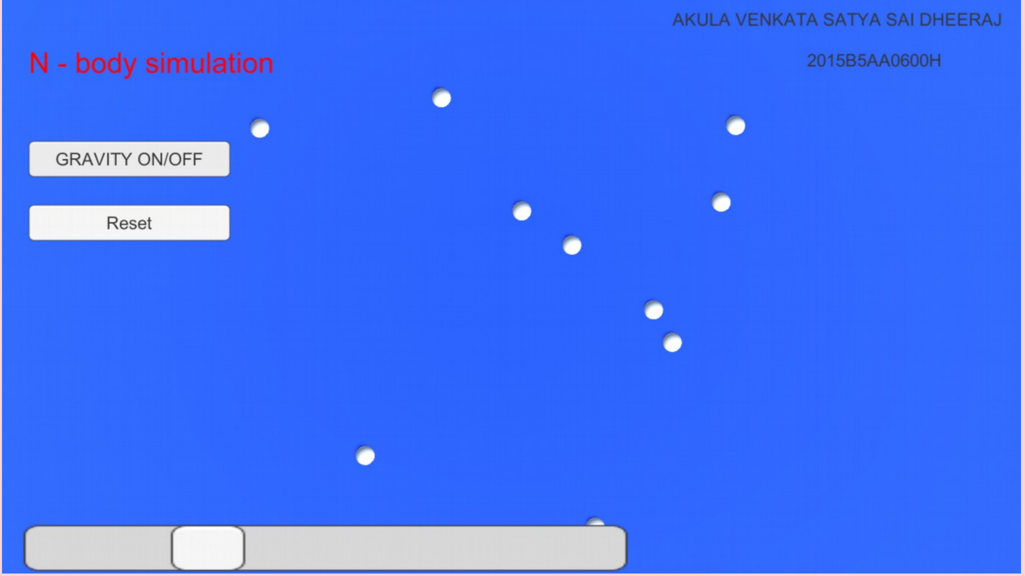
Then this velocity is used to update the position vectors by the formula
X = X + V * delta_t

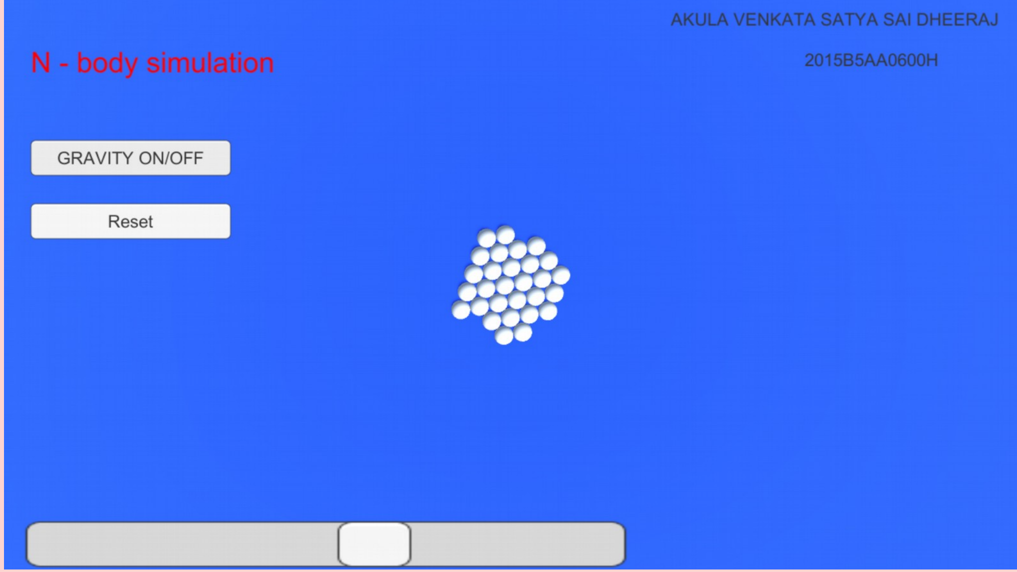
These calculations are done for every particle and the final positions are stored to a file for every delta_t.

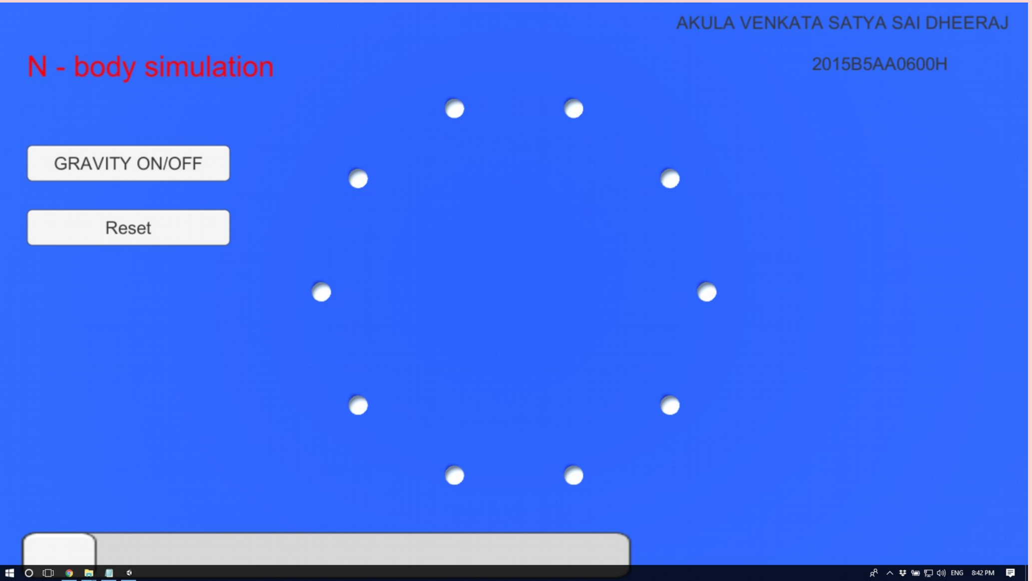
Implementing:

Method1:

At first I implemented this approach in a game engine called 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.

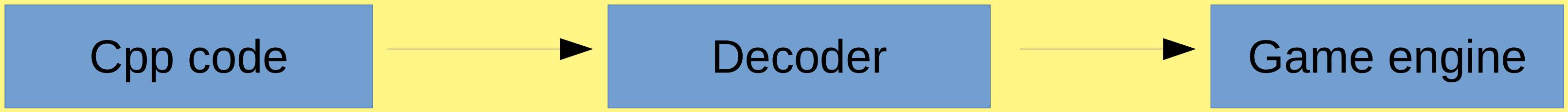

AKULA VENKATA SATYA SAI DHEERAJ
N - body simulation
2015B5AA0600H
GRAVITY ON/OFF
Reset


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

Method2:

So the next approach was to use naive cpp code to generate the position vectors and store them in a file.Then the game engine would decode this information and render the particles every frame.



Runs at user defined delta_t

Runs at 0.05 delta_t

Runs at 20 fps(0.05 delta_t) to match with the decoder

The decoder ensures that the match between game engine fps and output of our code.

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 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 every frame:

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

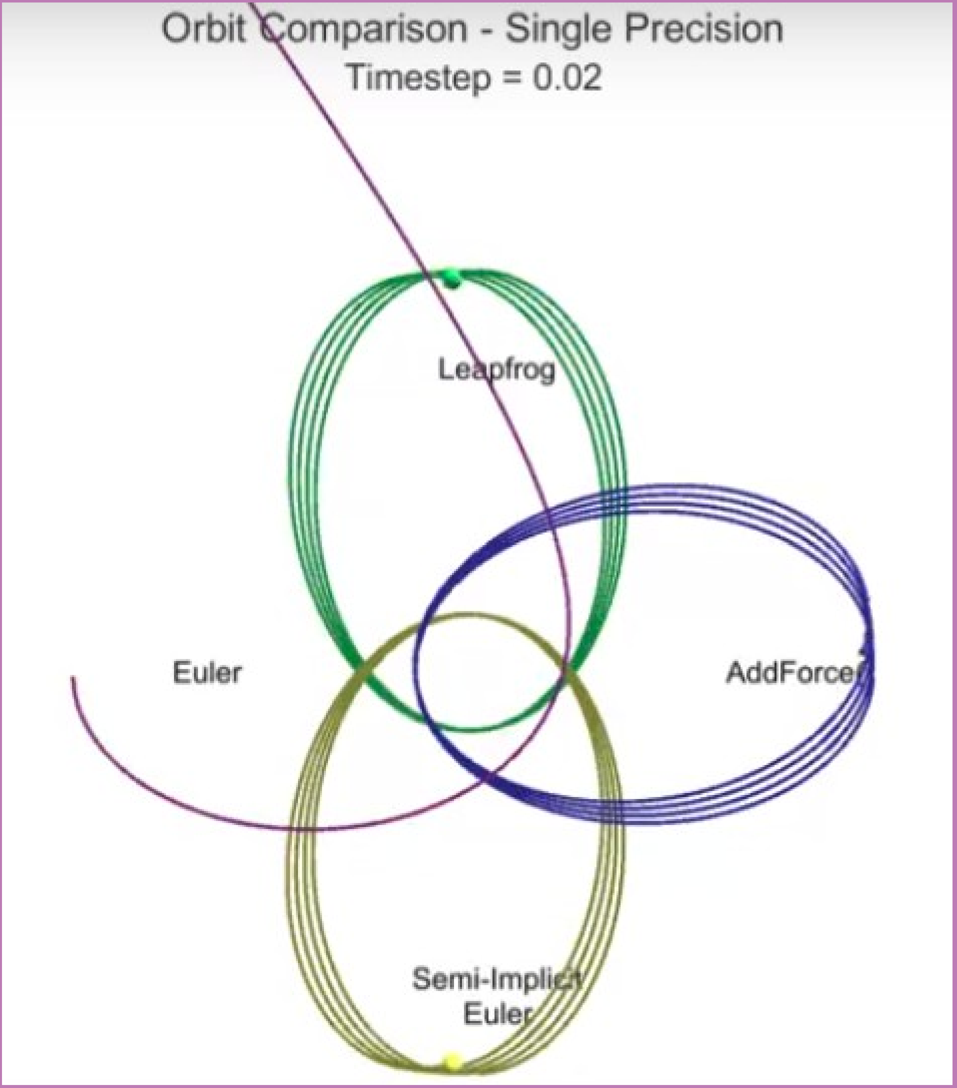
→ Euler Update

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

r = r + v * dt;

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

→ LeapFrog

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

→ Semi implicit-Euler



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.

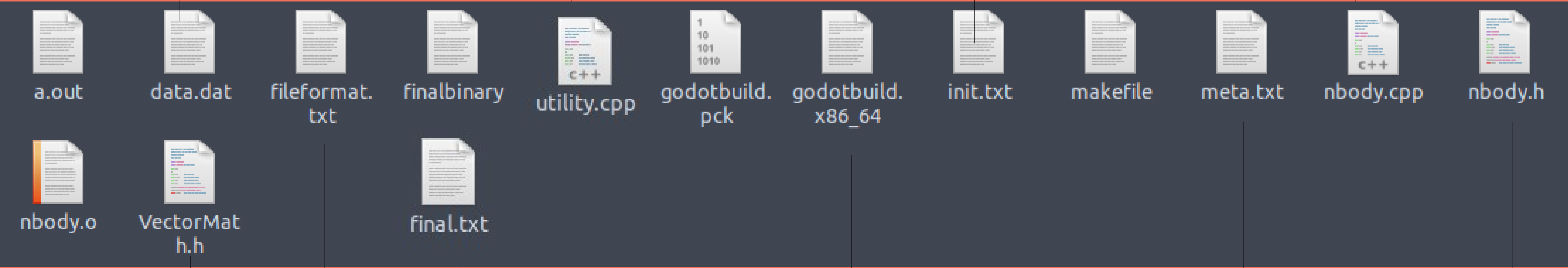
Project Structure:

Contains the generated data of the simulation.

Used to generate random initial conditons and calculate energy

Contains initial conditions of the particles

Contains the main code for simulation



Contains the utiltiy functions needed for mathematical operations

Contains the file format for init.txt

Contains the final velocity and positions of the particles

The game engine build which takes data.dat as input to display our particles

Contains the number of particles and delta_t

Header file of nbody.cpp

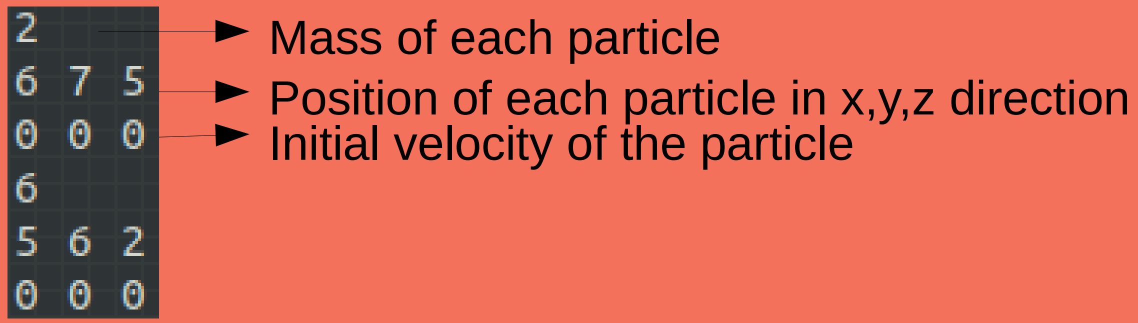
Steps to run the project:

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.



Init.txt

```
<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
```

File format of data.dat:

```
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
```

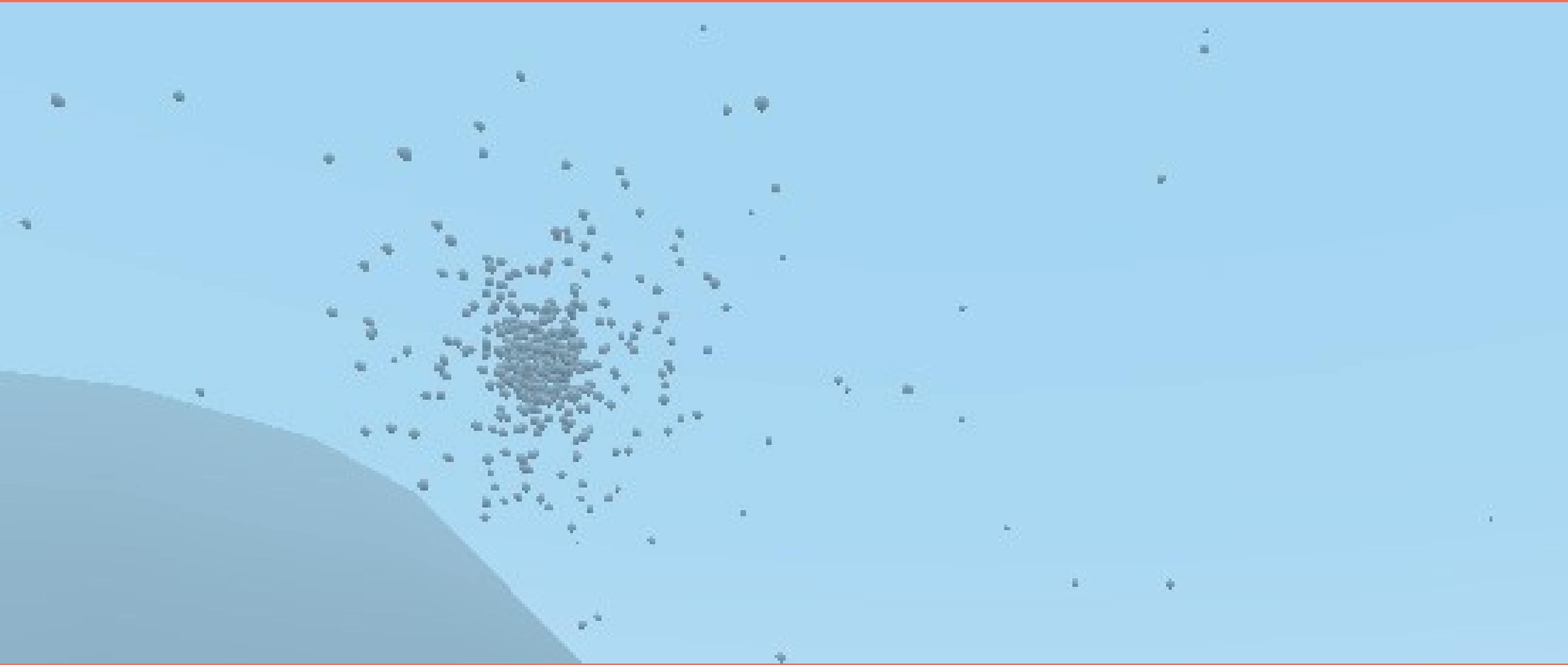
Position of first particle in x,y,z direction

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
```

Interesting phenomena observed during calculation of energy:

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 accelartion 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.

```
KE: 0
PE: 182.916
TE: -182.916
```

Initial Energy

```
KE: 201.296
PE: 579.769
TE: -378.473
```

delta_x = 0.01

```
KE: 90.7368
PE: 277.942
TE: -187.206
```

delta_x = 0.001

```
KE: 184.6
PE: 6.66722
TE: 177.933
```

delta_x = 0.0001