6/8/2019

SIT321 - Project

N-Body Problem

Cameron Pleissnitzer - 217154498
DEAKIN UNIVERSITY

Project Scope

This project will be centered around solutions to a popular class of the "N-Body" problem, and involves simulating the individual motions of a number of objects that are interacting with each other gravitationally.

The project will be broken down into three main parts:

1. **Sequential** Brute-Force implementation for 3D space
2. **Sequential** Barnes-Hut Octree implementation for 3D space
3. **Parallel** Barnes-Hut Octree Implementation for 3D space

Each implementation will be accompanied with a graphical playback of the simulation, which will contain various controls for camera movement and zooming, tracking individual object motions, and manipulating the playback state.

A number of pre-configured simulations will be provided to demonstrate interesting motions that can be selected from a menu using the console. A user-defined simulation will also be provided which will allow the user to set the number of bodies in the simulation, creating bodies with random masses, starting positions, and starting velocities.

A simpler approach will be taken when simulating the motions as to allow a greater focus on the playback mechanism and the implementations of the algorithms. For example, collisions, solar wind, and other possible factors that may manipulate the motion of a body other than the sum of gravitational forces, will not be considered in the calculations for the simulation.

Project Design

Included below is a UML class diagram indicating the intended design for the system. Importantly, the NBodySimulation class serves as a base class carrying the necessary functions and constants for running the simulation using the same parameters, providing a pure virtual ***computeAccelerations()*** that is then implemented by each derived class using their respective compute algorithms (brute force for brute force, Barnes-Hut octree for Barnes-Hut).

Designing the simulation system in this way ensures that the as long as each algorithm computes the same acceleration values (within a reasonable margin of error) then the simulation will always simulate similar motions, irrespective of the algorithm used, differing only due to errors introduced by the chosen implementation (e.g. Barnes-Hut is more or less erroneous in its calculations based on the value of THETA, a higher THETA means more octants are used as a centre of mass, rather than individual bodies).

Additionally, a SimulationUI class has been added to help abstract the UI and rendering logic from the main function. The SFML Graphics library has been chosen to aid in creation and rendering of UI elements to promote development time and favour focus on algorithmic implementation.

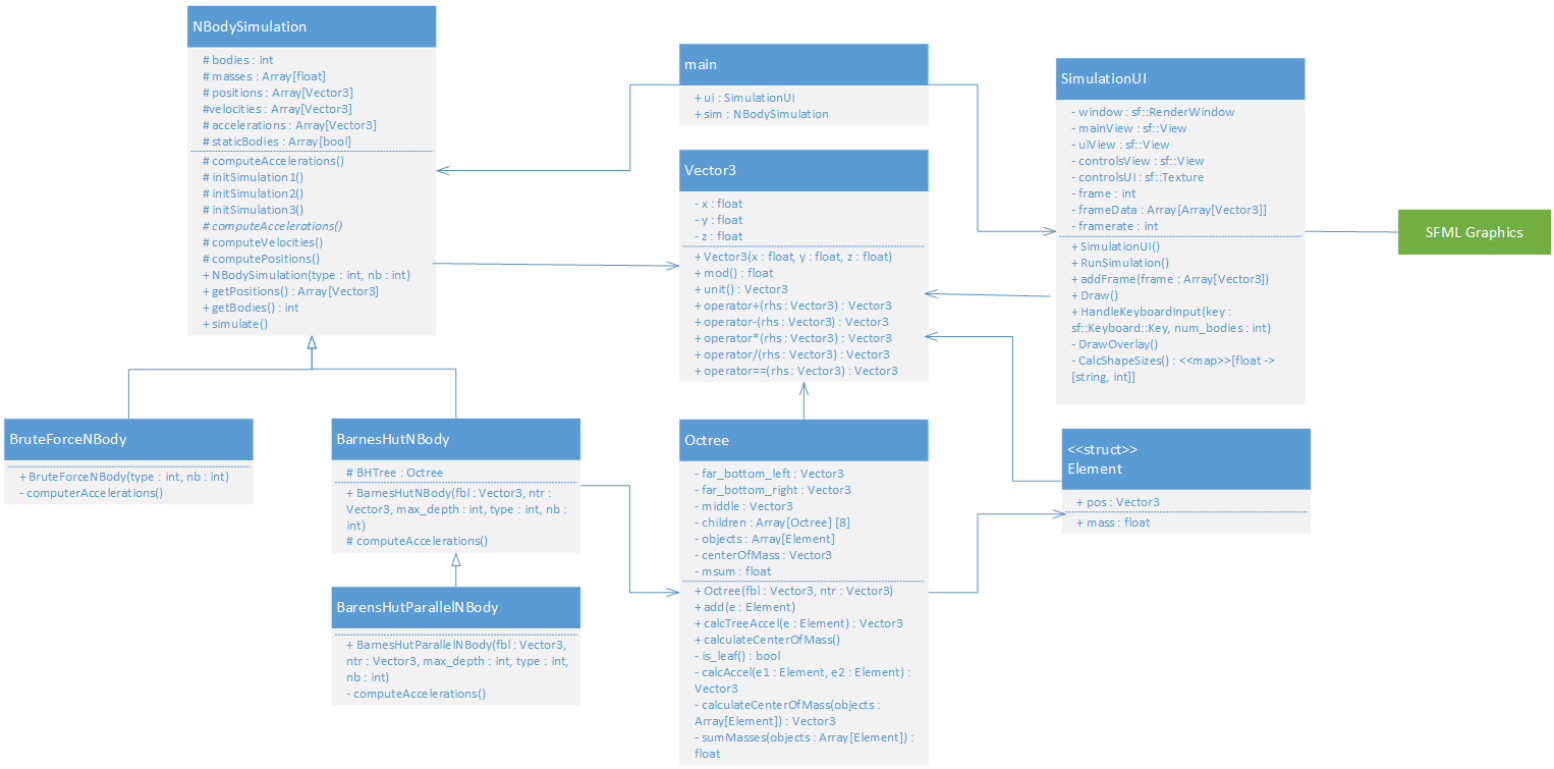


Figure 1 – UML Class Diagram of the NBody solution

Further to this, the foundation of each class is the Vector3 class. This class contains a custom implementation of a Vector in 3D space, and provides useful operations for calculating its size, unit vector, and operator overloads to make arithmetic operations between two vectors considerably easier and more readable.

Finally, the Octree class is perhaps the second most important aspect of the design. The Octree class will be used to implement a Barnes-Hut Octree, which will be able to calculate the centre of mass and sum of masses for each octant. The THETA value specified here will determine whether the octant is treated as a centre of mass during acceleration calculations on an element, or whether to recurse on the octant again (narrow to more individual points / centre of masses).

The main function will initialise the simulation method based on user input from the console (a menu choice) and also the type of simulation to run (user-defined or pre-defined) from another menu. Once the options are configured, the simulationUI and NBodySimulation will be initialised and the simulation is run. Finally, the UI is run after the simulation to playback the simulation frames and provide playback controls.

That concludes the high-level design description for the solution, the next section of this document will discuss the implementation, and what was achieved by the end of the project.

Project Implementation / Discussion

N-Body Brute Force Testing

With a significant amount of online resources to use as research, it was relatively easy to implement a basic NBody simulation problem with basic acceleration calculations due to gravitational forces. The hard part here was testing that the solution was correct, as without a graphical view of the simulation the numbers contained within each vector were hard to verify as accurate.

UI Build

The next phase involved building the UI using SFML Graphics. Initially, I performed this using simple fixed-size circle shapes representing each body, and only updated the x and y values in the space on each frame. This produced expected results and I was comfortable in moving on.

Due to starting the UI build, I decided to focus on this until it was finished to a reasonable standard. I modified the circle sizes using the z-value, and stored the sizes in a map to produce an ordered z-position list. This map is important to draw to the window with the correct z-ordering (larger objects appear above smaller objects), as SFML uses overdrawing rather than a z depth layer, meaning each element is drawn on top of each other in the order of drawing.

```
std::map<float, std::pair<const char*, int>> SimulationUI::CalcShapeSizes(int frame) {
    // Calculate render sizes (a map is used here to automatically order the data according to each bodies z position, used for SFML overdrawing later)
#pragma omp parallel for
    std::map<float, std::pair<const char*, int>> sizes;
    for (int i = 0; i < num_bodies; i++) {
        // Calc size
        float size;
        if (frameData[frame][i].z < -100)
            size = 1.f;
        else if (frameData[frame][i].z > 400)
            size = 40.f;
        else
            size = (((frameData[frame][i].z + 100) * (15.f - 0.01f)) / 200) + 1.f;

        // Add to map
        sizes[size] = std::pair<const char*, int>("body", i);
    }
```

Figure 2 - Calculation of shape sizes using z-position

After this, I added some rudimentary camera controls (zooming, panning) and playback controls (reset playback to frame 0, pause) to make testing and validation and bit easier, while also adding some nice functionality for later. I then added a ‘body tracking’ feature, which reveals a new UI to enter a number which represents one of the bodies in the simulation. The simulation then highlights the tracked body and draws a trail of where it has been.
Finally, with the new controls added, I created a controls screen and a basic texture to map to it using GIMP

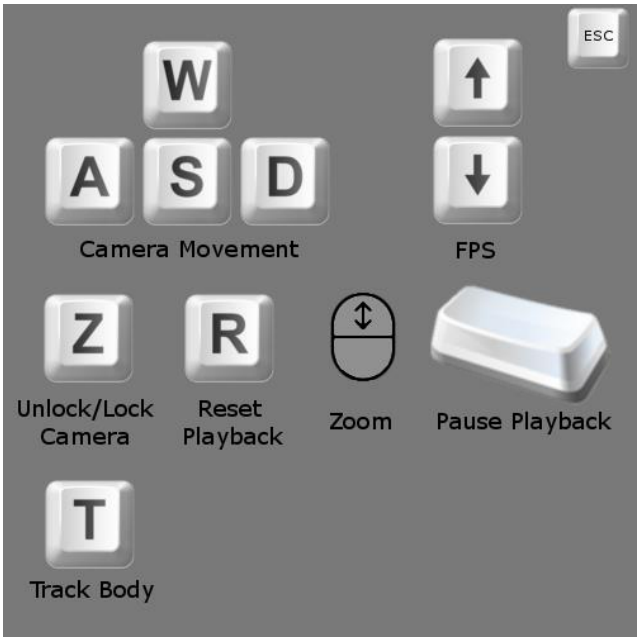


Figure 3 - Controls Screen

Barnes-Hut Algorithm

This section proved to be the most challenging. I was already very pressed for time, so it was difficult to both research and learn this complicated concept in a short period. I was able to implement a basic Octree data class as described in the design with relative ease, and then go from there.

Of course, a Barnes-Hut implementation is not simply just an Octree. It needs to calculate the centre of masses and sum of masses for each octant (and recursively on the child octants). These are then used in a recursive acceleration calculation function if a heuristically determined value (size of octant / distance to centre of mass) is greater than THETA, or just a normal two-body acceleration calculation if the value is less than THETA (this is where the centre of mass of the octant is used rather than the individual points within the octant).

The acceleration calculation was very difficult, as different papers and algorithms used different bases for their Octrees, and the pseudocode was sometimes very difficult to read. I introduced an error very early on, which softened intermediary calculations of acceleration, rather than the final output of the CalcTreeAccel function, which produced very strange behaviour and was difficult to pinpoint. It was only when I compared the acceleration values output by the Brute Force simulation and the Barnes-Hut simulation step-by-step did I notice the error.

Barnes-Hut Parallel

Unfortunately, due to the extended issues in implementing the above algorithm I did not have time to effectively parallelise this algorithm. Instead, I used OpenMP on the for loops with independent calculations (the acceleration calculation). This provides a slight performance boost, which will be discussed in the next section.

A popular method used by researches on this topic is a form of spatial partitioning called the LET method (Locally-Essential-Tree). It involves recursively subdividing the space into divisions with an equal number of particles until all processors are utilised, and each processor is assigned a rectangle.

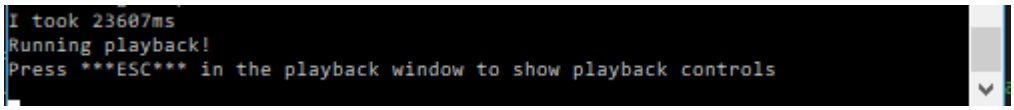
Each processor runs the Barnes-Hut algorithm on the bodies that it owns within the rectangle, but in order to do this it needs the subset of the octree that would be used for its own bodies, and also neighbouring tree nodes.

This was the intended progression for the project, but due to time constraints the implementation could not be performed.

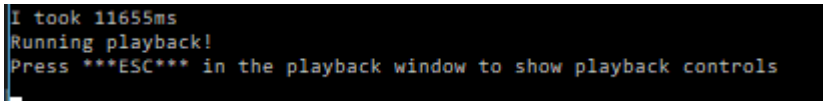
Performance

NUM_BODIES = 100; STEPS = 1000

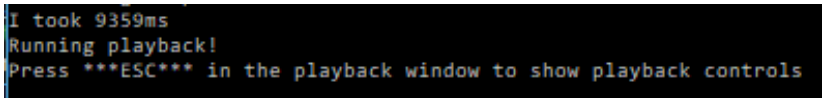
Brute Force



Barnes-Hut Sequential



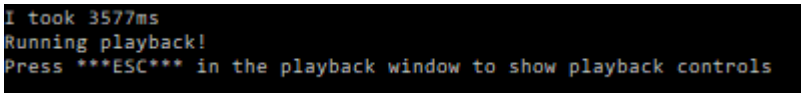
Barnes-Hut Parallel



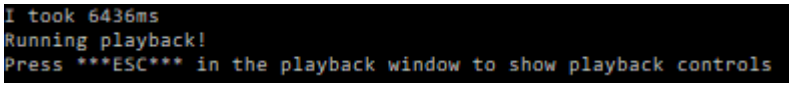
For a larger number of bodies, the Barnes-Hut algorithm has a near 100% potential speed-up, even more-so for the parallel implementation. This is likely due to greater usage of the centre of mass of an octant, rather than each individual body in the calculations.

NUM_BODIES = 10; STEPS = 10000

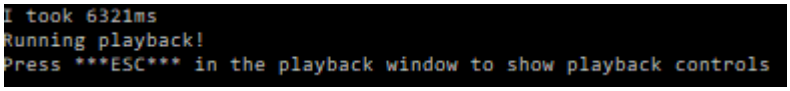
Brute Force



Barnes-Hut Sequential



Brute Force Parallel



For a lower number of bodies, the brute force implementation is the inverse and almost 100% faster than the Barnes-Hut solution. Given that this is a low number of bodies, it makes sense that the overhead of building the Octree would negate the very minimal amount of calculations that use the centre of mass.

Video Demonstration Link

Screenshots of the application running have not been included in this document as that is the purpose of the video demonstration. The video demonstration shows both usage of the project, and the project running.

Link: https://youtu.be/QW8MpOvB-6U