Name:	Jude Onyia
Student ID:	V00947095
Course:	ECE 596C
Due Date:	June 26, 2020
Deliverable:	Project Proposal

Coarse and Restricted Simulation for the N-Body Problem

1. Problem

The n-body problem is the complication involved in predicting the movements of celestial bodies within a system as they interact with each other gravitationally [1]. Predictions involving systems with more than two bodies do not cover a complete analytical solution [1]. Current solutions provide only a coarse prediction of the system's behaviour due to factors that are still inconclusive in general relativity [1]. This project will contribute in realizing an estimated prediction of the motions within an arbitrary system, and it will provide visual aid in understanding these dynamics while under certain constraints. The most accurate solution to the n-body problem is the direct integration approach [2]. This involves acquiring the force on each body by taking the summation of forces produced by the remaining N-1 bodies. This results in the solution having a time complexity of $\Theta(N^2)$ [1][2]. The approach requires initial positions, velocities and accelerations of the N bodies with distinct masses. Each time step will require moving each body according to its velocity, updating its velocity according to its acceleration, and calculating a new acceleration based on the summation of forces from other bodies [3]. To accomplish this computation within a short time step, a constraint on the size of N is required for a general-purpose computer.

2. Introduction

The individual force $\boldsymbol{F}_{ab}$ on a body with mass m_a by a body with mass m_b is defined in equation 1, based on Kepler's three laws [3]. Where x and y are the two-dimensional coordinates of the center of each body [3]. All individual forces acting on a body will be reduced into one collective force by taking the summation of the i or x components and the summation of the j or y components of all individual forces. These components correspond to the x and y coordinates, respectively.

$$\boldsymbol{F}_{ab} = -\frac{G \times m_a \times m_b}{r^2} \times \hat{\boldsymbol{r}} \tag{1}$$

$$\boldsymbol{r} = (x_a - x_b)i + (y_a - y_b)j$$

$$r = ||\boldsymbol{r}|| = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

$$\hat{\boldsymbol{r}} = \frac{1}{r} \times \boldsymbol{r}$$

$$G = 6.67408 \times 10^{-11}\ \frac{m^3}{s^2 kg}$$

A new oriented acceleration for a body can be calculated using Newton's second law of motion in equation 2, where $\boldsymbol{F}$ is the collective force on the body, and m is the mass of the body [4]. Within each time step, the assumption that the acceleration is constant is held prior to the calculation of a new acceleration at the end of the time step. This assumption helps reduce computation, though it does add a minuscule error proportional to the time step [2]. Equation 3 describes how the velocity of a body is updated considering constant acceleration, where $\boldsymbol{v}_i$ is the current velocity and $\boldsymbol{v}_f$ is its new value [4]. Similarly, the velocity is assumed to be constant within a time step prior to its update at the end of the time step. This incurs some error as well that can be negligible for the scope of this project. The position of a body is updated as shown in equation 4, considering constant velocity [4]. In equation 4, $\boldsymbol{x}_i$ is the current position of a body in the x and y plane, and $\boldsymbol{x}_f$ is its new position.

$$\boldsymbol{a} = \frac{\boldsymbol{F}}{m} \tag{2}$$

$$\boldsymbol{v}_f = \boldsymbol{v}_i + \boldsymbol{a}t \tag{3}$$

$$\boldsymbol{x}_f = \boldsymbol{x}_i + \boldsymbol{v}t \tag{4}$$

3. Aim

This project will employ the equations described above to simulate an estimated trajectory of arbitrary celestial bodies bounded in a two-dimensional space and under certain constraints regarding contact. This simulation will be beneficial in providing some visual understanding of interactions within any arbitrary system desired. The space analysed is two-dimensional to ensure simple visualization.

The goal of the project is to develop a program that will first read from a text file the number of bodies in the system, their masses, initial positions of the center of the bodies, velocities and accelerations. The text file, provided with the program, will be altered by the user of the program to fit their specifications for the system. Details of the text file are explained further in the report. The program will open a window and create a set of two-dimensional circular objects representing the celestial bodies in a system, similar to the objects shown in **Figure 1**. The radii of these objects will be proportional to the masses of the bodies, but limit with maximum and minimum values. The program will map the range of positions of the bodies, relative to an origin, to pixels on the window. The circular objects will be placed in positions on the window corresponding to the physical positions of the bodies they represent. The program will use the equations described earlier and the mapping algorithm to update the positions of the circular objects during each time step, producing an animated depiction of the interaction between the bodies.

In the case of an overlap between circular objects (collision between bodies), the program will have the bodies exchange x and y velocity components to roughly emulate a deflection. For the program to determine whether any object overlaps with another object, it needs to check that the radius of every object is less than the x and y components of the distance between that object and all other objects at each time step. This does not add significantly more computation because the distance $\boldsymbol{r}$ between each object and all others must be calculated anyway according to equation 1. In the case of an object reaching the edge of the window, the program will multiply the object's x and y velocity components with -1 to emulate the object deflecting off the edge of the window. For the program to determine whether an object reaches an edge of the window, it needs to check that the value of the x and y positions plus the radius of the object does not go beyond the values corresponding to the edges of the window.

4. Methodology

For the program to handle very large numbers for masses of bodies and distances between bodies, as well as very small numbers such as the gravitational constant G, the project will involve defining a C++ class for representing logarithm base 10 numbers. This class will have two key data members corresponding to the scalar and the exponent of a logarithm base 10 number. For example, the gravitational constant G defined in equation 1 can be expressed with an object of this class where the scalar is 6.67408 and the exponent is -11, the stream extractor overload for this class will require the format: *6.67408e-11* for G as an example. This will allow computation with numbers that would have otherwise been rounded off as zero or overflowed. An example of the need for this class would be multiplying G with 3.5×10^{30}. With this class, the computation would involve multiplying 6.67408 with 3.5, and adding -11 and 30, a similar action is performed for division. For addition and subtraction, the steps would involve identifying the number with the lesser exponent, changing the representation of that number (i.e: *2.0e2 = 0.02e4*) so that its exponent matches the larger number, then adding or subtracting the scalar and taking the common exponent. If changing the representation of the number results in the scalar being rounded to zero, this will have little consequence because this means the other number involved in the operation is so large that the number being changed is negligible.

The program will need to handle the states of each body, i.e. the mass, position, velocity and acceleration. This can be done by defining a C++ class for representing bodies with these states as data members. This class would have useful member functions to check if a circular object is overlapping with another, and if it reaches an edge of the window. The program will also define a class for containing x and y components, with useful member functions such as one that obtains the norm. Mathematical operations with this class will be define appropriately.

To achieve the visual aspect of this project, OpenGL and the GLUT library will be used. The project requires some functionalities of the libraries such as creating a window, displaying the window, and reshaping the window based on a specified width and height. It also requires creating primitives (objects made form interpolating vertices/pixels), translating primitives, and providing different colors to different primitives. The project requires the GLUT library's timer function, and the library's system of switching frames every specified fraction of a second. The project also requires the library's functionality of linking a specified arbitrary grid to the window's pixel grid. **Figure 1** shows pictures from an animation produced by a preliminary program developed for this project to test the required functionalities of the GLUT and OpenGL libraries.

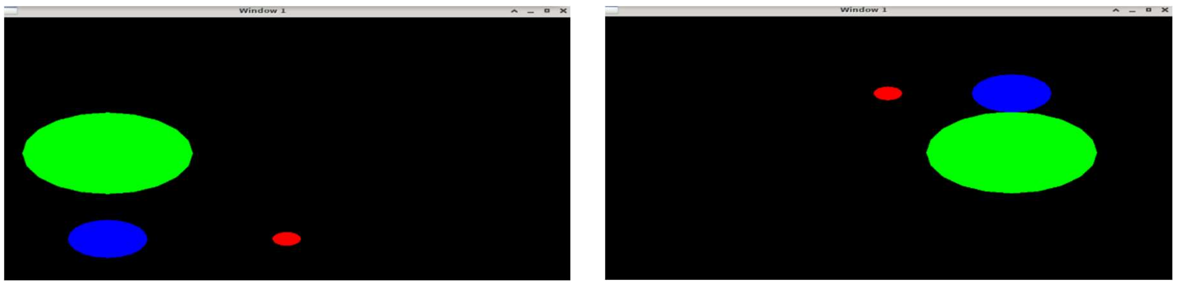

Figure 1: Animation of Preliminary Program testing GLUT and OpenGL functions

The text file needed to hold the initial states of the N bodies must have the format shown in **Figure 2**. The text file must also be called *initial_state.txt* because that is the name that will be written in the program, and it must be placed in the same directory as the program. In **Figure 2**, N refers to the number of bodies in the system, m_n is the mass of the nth body, x_n is the x component of its initial position, and y_n is the y component of its initial position. vx_n is the x component of the body's initial velocity, vy_n is the y component of its initial velocity, ax_n is the x component of its initial acceleration, ay_n is the y component of its initial acceleration. **Figure 3** shows an example of how the text file should look like; this is also the content of the sample text file provided with the program. This allow the user to have a reference to the format and a test case available. Notice that all values except *N* follow the format of the logarithm base 10 number class stream extractor, each value is separated by a single space.

N

$m_1\ m_2\ m_3 \cdot\ \cdot\ \cdot m_N$

$x_1\ x_2\ x_3 \cdot\ \cdot\ \cdot x_N$

$y_1\ y_2\ y_3 \cdot\ \cdot\ \cdot y_N$

$vx_1\ vx_2\ vx_3 \cdot\ \cdot\ \cdot vx_N$

$vy_1\ vy_2\ vy_3 \cdot\ \cdot\ \cdot vy_N$

$ax_1\ ax_2\ ax_3 \cdot\ \cdot\ \cdot ax_N$

$ay_1\ ay_2\ ay_3 \cdot\ \cdot\ \cdot ay_N$

Figure 2: Format of text file *initial_state.txt*, where N is the number of bodies in the system, m_n, x_n, y_n, vx_n, vy_n, ax_n, and ay_n are the mass, position, velocity, and acceleration of body n, respectively for $n = 1,2, \dots, N$.

```
8

1.989e30 3.285e23 4.867e24 5.972e24 7.347673e22 6.4171e23 1.898e27 5.683e26

0.0e0 5.791e10 1.082e11 1.496e11 1.496e11 0.0e0 0.0e0 1.35255e12

0.0e0 0.0e0 0.0e0 0.0e0 -3.633e8 2.2792e11 7.785e11 0.0e0

0.0e0 0.0e0 0.0e0 0.0e0 0.0e0 2.65e4 1.31e4 0.0e0

0.0e0 4.74e4 3.54e4 3.0e4 3.0e4 0.0e0 0.0e0 9.68e3

0.0e0 0.0e0 1.3e2 0.0e0 1.022e3 0.0e0 0.0e0 9.68e3

0.0e0 0.0e0 2.1e1 0.0e0 0.0e0 0.0e0 0.0e0 0.0e0
```

Figure 3: An example of the text file *initial_state.txt* with the appropriate format.

Table I shows the planned schedule that will be followed to complete the project within the remainder of the semester. This schedule is structured to provide some slack time between deliverables in case of unforeseen delays. The schedule purposefully covers broad goals to allow flexibility in implementation.

Table I: Schedule for the Project

Week	**Deliverable**
Week of July 6th, 2020	Implementation of the logarithm base 10 number class
Week of July 20th, 2020	Implementation of the body class and the algorithms needed in predicting the movement of bodies within a system
Week of August 3rd, 2020	Completion of the OpenGL animation for visualizing the n-body problem simulation
Week of August 10th, 2020	Creation of more test cases (new values for *initial_state.txt*) to help fix bugs in the program and demonstrate functionality

5. Conclusion

The motivation for this project is to improve students' ability to write C++ code and apply it to a practical problem, this will be satisfied by the work needed to accomplish this project. The objective of the project is to provide a simple visualization for a simulation of the n-body problem. This is achieved by using equations based on Newton's seconds law of motion and Kepler's three laws. The project will also require functionalities found in the GLUT and OpenGL libraries. The project can be deemed a success if a functioning animation is made that depicts the movements of arbitrary celestial bodies within a bounded system.

6. References

[1] L. Greengard, "The Numerical Solution of the N-Body Problem," American Institute of Physics, 1990.

[2] E. Vasiliev, "N-body simulations tutorial," Space Science Reviews, May 7, 2013.

[3] U. o. Minnesota, "Newton's equation with gravitational force," IEEE, 2014.

[4] M. S. University, "NWHS Physics Equations," Michigan State University, East Lansing, 2010.