6.4420 Project Writeup
David Xiong
Gravitational Simulation for 3-Body Problems

The problem at hand is to simulate how objects move under Newton's laws of universal gravitation. For 2-body problems, the motions of the objects are well known and can be derived analytically. However, for 3-body problems, there is no closed-form solution and the movements are very chaotic. I aim to analyze the movements of the 3-body problems by creating a simulation space for n-body gravitational problems in Unity, with scripting in C#.

Previous work on the 3-body problem dates back to the 1700s. Euler discovered a stable solution for the 3-body problem where all objects remain collinear as they evolve. Lagrange later found another stable solution that involved the three bodies rotating in an equilateral triangle configuration. In 1889, Henri Poincare proved that no equation can fully describe the solution to the 3-body problem. Nowadays, both analytical and computational research is being done to try to more accurately describe the 3-body problem. Recently, many families of solutions have been found for special cases of the 3-body problem, such as the case when all three bodies start at rest. More complicated numerical methods have also been used, with some simulators taking into account EM interactions and relativistic regimes.

I chose to use Unity to create the simulation as it already has a lot of structure needed for the simulation. Because it is a game engine, it runs in real-time and has easy visualization methods. Each gravitational body was represented as a sphere model in the 3d space, with a gravitational script attached to it to calculate the gravitational forces from other bodies, as well as the position, velocity, and acceleration of the body at each time step. This model was turned into a prefab, which is a template to easily create as many gravitational bodies as needed.

The gravitational bodies were evolved in time using direct integration methods. I implemented two integrators, an Euler integrator and a Runge-Kutta 4 integrator. The RK4 integrator (http://spiff.rit.edu/richmond/nbody/OrbitRungeKutta4.pdf) evolves both position and velocity at the same time, and produces more accurate results than

the Euler integrator. In addition, Unity allows the use of public variables which can be set from the scene. With the initial velocity as a public variable, it allows the user to customize a scene however they want, by placing the bodies where they want and determining their initial velocities.

Different scenes were created for different experiments. I first tested my script with a 2-body system rotating around each other. This performed as expected, even when using the Euler integrator. I next used three bodies, and looked at many different configurations to see if there were any common patterns. For some configurations, two bodies would get too close to each other and then fly quickly off screen after colliding. Other configurations would have some chaos at the beginning, then two bodies would pair up to rotate around each other while the third flew off in the opposite direction.

Another interest of mine was to analyze some of the known stable configurations of the 3-body problem. Starting with the Lagrange configuration, I found that with masses that are a difference of 10 times each other, the largest body which the smaller ones rotate around would also move, and the system would evolve to be chaotic after some time. However, when the bodies were much different in mass (around 1000 times), then the largest body would no longer move. The smaller ones would also start to rotate as they should around the largest body, and the stable configuration would be preserved. In Euler's configuration, three equal masses were used, with a horizontally symmetric setup. However, despite being symmetric around the middle mass, the middle mass would still end up moving after some time, and the system would devolve to chaos. Only by setting the middle mass to be immobile would the system maintain the stable evolution. Finally a figure-8 stable configuration was tested. The simulation was unable to reproduce the stable configuration, and the bodies would become chaotic before one period.

In conclusion, the direct integration methods I implemented did not produce as much fidelity as I would have wanted. While it was expected for most 3-body configurations to be chaotic, the configurations which should be theoretically stable were often not in the simulations. Thus, the results were unexpected. For future work, implementing an implicit integrator to replace the direct integrator would likely get better results. This is because the implicit integrator would be better at conserving

momentum. However, it would likely take longer to calculate these results, and might be better done not in a real-time program like Unity.