

James F. Guillochon  
Mentor: Dr. James Bullock

## **Galactic simulation using a Barnes-Hut N-body algorithm**

*Abstract: In bound galactic clusters, the primary force that controls the motions of the constituents is gravity. While analytical equations exist for the two-body problem, there is currently no exact analytical method for computing the future positions of a system with more than two objects. Therefore, the only way to predict how a system of  $N$  objects behaves is to calculate the force each exerts on the other using discrete time-steps. The goal of this study was to write an  $N$ -body program using an algorithm that accurately simulates a large number of objects, while also running in a reasonable amount of time on a home computer. It was found that by using the Barnes-Hut tree code algorithm, systems of up to a million objects could be realistically simulated. The flexibility of the code also allowed many astrophysical systems to be analyzed in great detail over long periods of time.*

The development of galaxies has remained a mystery since their discovery by Hubble in 1924. Theoretical models have been created to recognize the various stages of galactic development, and these models have had stunning success. Observational data has also improved by leaps and bounds over the past decade, and with the help of the Hubble Space Telescope we can see distant galaxies with more detail than ever before. However, while both theoretical and observational techniques have improved, the nature of these methods neglects much of the finer details of galactic development. Analytical theories rely on the bulk properties of galaxies, and observationally we cannot resolve most individual stars from galaxies that are not our own. To fill in the gaps, we can use numerical simulation to predict how galaxies interact with each other. Given the shortcomings of the other approaches, there is still much that can be learned by running computational simulations that could not be studied otherwise. These  $N$ -body simulations allow us to peer into the hearts of galaxies and study how they behave, without having to deal with the rough approximations of empirical models or the problems associated with observational data. In this paper various algorithms will be explored, and two will be analyzed in great detail. Those codes will then be used to simulate physical systems to show that they work properly, and that the simulation options are useful scientifically.

There are several different classes of stellar simulation codes, each which have their own advantages and disadvantages. A list of the more well-known algorithms are given below<sup>1</sup>:

- Particle-Particle (PP)
- Particle-Mesh (PM)
- Particle-Particle/Particle-Mesh (P3M)
- Particle Multiple-Mesh (PM2)
- Nested Grid Particle-Mesh (NGPM)
- Tree-Code (TC) Top Down

- Tree-Code (TC) Bottom Up
- Fast-Multipole-Method (FMM)
- Tree-Code Particle Mesh (TPM)
- Self-Consistent Field (SCF)
- Symplectic Method

The two codes that will be explored in this paper are Particle-Particle (PP) and the Tree-Code (TC) Top Down, which is also known as the Barnes-Hut Algorithm. The PP algorithm is the simplest in terms of mathematical complexity, but it also has many features that make it desirable in certain situations. The primary draw to the PP code is its accuracy when used to simulate a system with few collisions. The TC algorithm is similar to the PP, but it allows for approximations to be made when objects are far away from each other to speed up the computation. This is desirable when working with a system that has many sub-systems, e.g., a cluster of galaxies. Roughly speaking, taking the number of objects in a system to be  $N$ , the PP method scales as  $N^2$ , while the TC method scales as  $N \log N$ . A thorough analysis of the computation time for the PP method is given in Hockney and Eastwood<sup>2</sup>, and while their result has an additional term, at large  $N$  computation time is proportional to  $N^2$ . Typically, if one wants to simulate a single galaxy, a Particle-Particle code is used, while the Tree-Codes are used when many galaxies are being simulated.

### **Writing the Algorithms**

The Particle-Particle code was written first, mostly because it is fairly simply to understand and to use. The code was written in Fortran, chosen because it is efficient for scientific computations and was the most convenient to use with the hardware that was provided. The Muon computer cluster, which consists of a cluster of 10 Pentium III CPUs, was used to run the PP code.

The basic idea behind the code is to pick one of the constituent objects of the system, calculate the force each other object exerts upon that object, then adjust the object's velocity and position, and then repeat the process with the next entity. A step-by-step synopsis of the code's order of operations is given below.

1. Define the system with a set of initial conditions. This sets the number of particles, the mass of each particle, where the particle lies, and how the particle is moving. This data can either be generated from a distribution function or read in from a file.
2. Calculate the force exerted on a particle by all the other particles, and combine all the accelerations to determine a net acceleration for the selected particle. Repeat this step for all of the particles in the system.
3. Change the velocity of each particle by multiplying the acceleration by the step-time and adding the resultant velocity to the velocity vector.
4. Change the positions of each particle by multiplying the total velocity by the step-time and adding the resultant to the position vector.
5. Go back to step 2 and repeat.

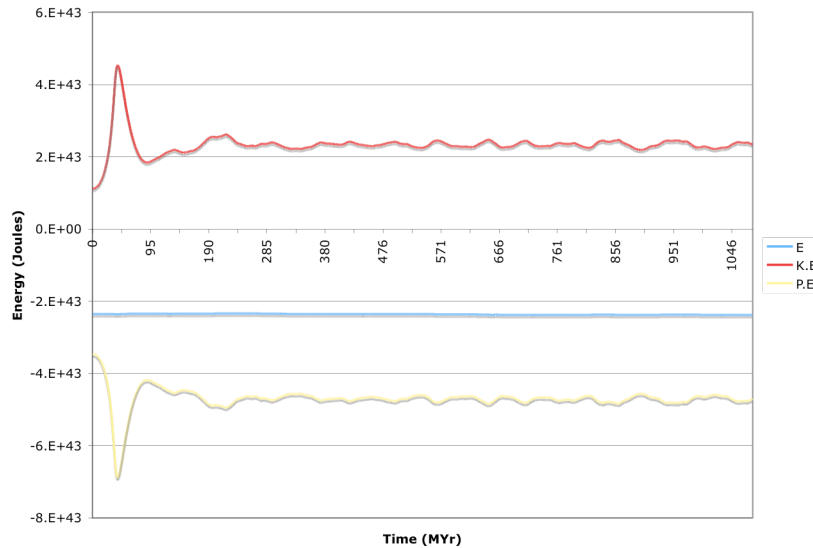
Formulaically, each object is subject to a force that is the summation of the forces all the other objects exert on it. This causes the velocity of the particle to change, as shown below.

$$\begin{aligned}\Delta\vec{v}_M &= (\vec{a}_1 + \vec{a}_2 + \dots)t = \sum_{n=1}^N \vec{a}_n t \\ \vec{a}_n &= \frac{Gm_n}{|\vec{r}_{Mn}|^2} \\ \therefore \Delta\vec{v}_M &= \sum_{n=1}^N \frac{Gm_n}{|\vec{r}_{Mn}|^2} t\end{aligned}\tag{1.}$$

The  $t$  in the above equations is the *step-time*. The step-time method is an approximation; it assumes that both the object that is producing the force and the object that is being acted upon by the force do not move very far from their original positions over the time interval. Therefore, the step-time must be small enough so that no particle moves very far over the course of a step. Ideally, we would want the step-time to approach zero for the best possible simulation, but practically, it is found that the simulation takes much too long if the step-time is too small. A proper step-time can be determined by running test simulations and checking that energy is conserved. If a system of objects is placed in a non-collisional system, the sum of the potential and the kinetic energies should always equal a constant<sup>3</sup>:

$$T(t) + U(t) = E_0\tag{2.}$$

It is desirable for the step-time to be set to a value that conserves energy within  $\pm 1\%$  of the initial energy over the course of a simulation. This allows for the simulation to run quickly, while at the same time deviates little from the ideal situation. The value of the step-time can be inferred from the initial conditions of the system. An example of a system that conserves energy is shown below.



**Figure 1:** An example system consisting of 2000 objects that conserves energy.

Suppose a particle has a velocity  $v$  and is a distance  $d$  from another particle. If  $v$  times  $t$  (the step-time) is on the order of  $d$  for most interactions, the step-time that has been chosen is too large. Energy in this situation will not be conserved. The solution is to pick a smaller value for the step-time.

Eventually, the code was parallelized to allow it to run on a cluster of computers. The code's operation changes by splitting step 2 across a number  $M$  CPUs, assigning  $1/M$ th particles to each "slave" computer. Step 4 is run on a "master" computer, which acts as an arbiter for communications between all the systems. The master machine sends out a fraction of the objects to each slave machine, and waits for the slaves to complete the computation listed in step 2. Once the slaves finish, they send the new velocities back to the master machine, which updates the positions and velocities of all the particles accordingly. The process is then repeated until the application is exited.

There were several downsides to having written the code in Fortran. Firstly, the syntax of the language is not as intuitive as the more modern programming languages, which made the programming process difficult at times. Fortran also lacks the ability to dynamically allocate and deallocate memory during the run-time, making it difficult to remain memory efficient if the characteristics of the system change during the course of a run. Beyond those annoyances, the primary downside to using Fortran is that several of the more advanced N-body codes require the ability for functions to call themselves, *recursion*, which is a feature the language does not support. Therefore, it was decided that the next generation simulation should be written from scratch in C++. Moving to the more modern language permitted the creation of a tree-code.

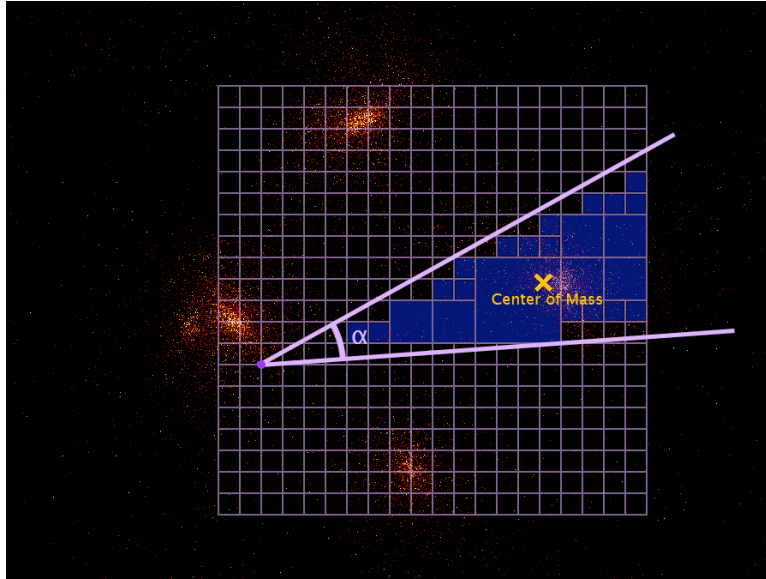
The Top-Down Tree-Code is one of the more intuitive approaches to the problem, and mathematically it is identical to the Particle-Particle code when two objects are near one another. The main benefit of the TC is that groups of objects that are far away from the object in question can be approximated, this is done by treating them as one object with the mass of all the constituents, placed at the group's center of mass. The code splits the space into eight octants, and then splits each of those eight octants into eight more

octants, repeating the process until a specified tree depth is reached. Center of mass approximation is used when the width of the box that contains the objects exceeds the distance between the two objects times a critical opening angle<sup>4</sup>:

$$\frac{l}{d} < \theta \quad (3.)$$

This means that the further away an object is, the larger the box used in the center of mass approximation. In his paper on tree-codes, Hernquist suggests that the angle theta should be less than one degree to keep the errors below 1%. In the simulations that I ran, the angle was set to five degrees. This is permissible because the width of the smallest box was comparably large to what is typically used in supercomputer clusters, and thus the approximation is used less often. This results in a less efficient computation cycle, but requires substantially less computer memory. The itinerary of a Top-Down TC is given below:

1. Define the system with a set of initial conditions.
2. Create an octree of a pre-defined depth.
3. Assign each particle to the sub-octant that includes its spatial position.
4. Calculate the centers of mass of all the sub-octants contained in the tree.
5. Calculate the force exerted on a particle by all the other particles, and combine all the accelerations to determine a net acceleration for the selected particle. If the particle's distance times a specified critical angle exceeds the width of any of the sub-octants that contain the object that contains the particle, use the center of mass of that box to perform all force calculations associated with any particles that box contains. Repeat this step for all of the particles in the system. A visual representation of this process is shown in figure 2.
6. Change the velocity of each particle by multiplying the acceleration by the step-time and adding the resultant velocity to the velocity vector.
7. Change the positions of each particle by multiplying the total velocity by the step-time and adding the resultant to the position vector.
8. Go back to step 3 and repeat.



**Figure 2:** A diagram of how the TC algorithm determines which box size to use when calculating gravitational interactions.

An attractive feature of this code is that both the octree depth and the critical angle are completely arbitrary. When the critical angle is set to zero degrees, the code performs identically to a Particle-Particle algorithm. The depth of the octree can also determine calculation speed. If the octree is very shallow, the angle approximation will not be used very often, as the sub-octants never get very small. If the octree is several layers deep, the center of mass approximation will be used frequently, even when objects are relatively nearby. If too many approximations are used, the simulation ceases to conserve energy, and becomes less feasible for practical problem solving.

### **Code Refinements**

One of the large issues an N-body simulation faces is what to do when two objects come very close together. In a two-body model, the paths of the particles are curved, and forces are integrated over infinitesimal time divisions. In the N-body case, object movements are not curves in space; the paths consist of hundreds of short line segments. When the distance a particle travels in a single step exceeds the distance between itself and another object, a “collision” is said to occur, and the error produced by the interaction is substantial. A single event can double the total kinetic energy of the entire system and dangerously violate energy conservation. A solution to this problem is to implement “force softening.” Force softening reduces the force between two objects when they come close together, preventing the objects from drastically changing each other’s energies. This is done by adding a *smoothing parameter* to the force equation:

$$|a| = \frac{Gm}{r^2 + \epsilon^2} \quad (4.)$$

This smoothing parameter was set to one-tenth the average distance between neighboring particles, which was found to conserve energy satisfactorily. Ideally, one would want to check the distance between two objects before using the smoothing parameter, but performing this check adds substantial computation cycles to the algorithm. It is therefore better to calculate the average distances before the simulation is run and just use that smoothing parameter for all interactions. In the limit that  $r \gg \epsilon$ , equation (4) approaches the second line of (1).

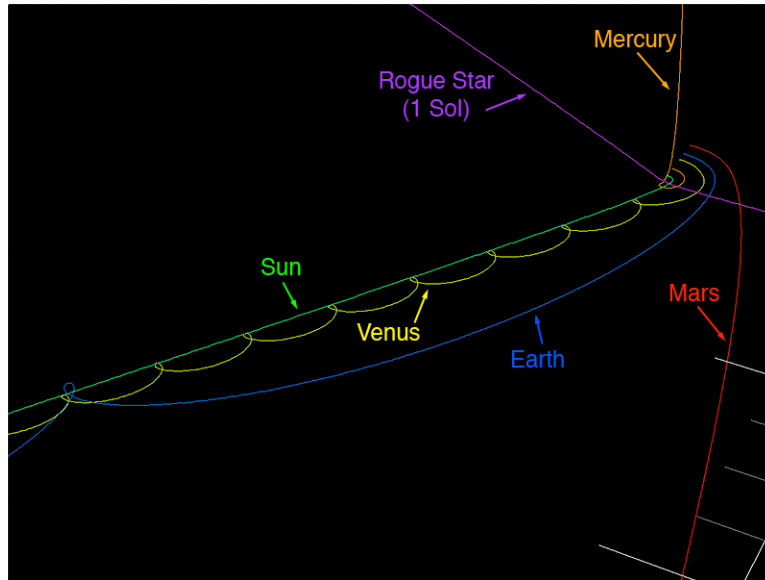
### **Algorithm Outputs**

Once the Barnes-Hut code was written, the graphical output of the program was the next area that was focused on. One of the benefits of writing the code from scratch was complete freedom over the output of the program. This allows for detailed logging of the events of the system while the simulation is running, which helps to determine if the initial conditions were set correctly. The convenience of being able to visually see what the system is doing after each step allows for changes to be made before much CPU time is wasted doing a bad simulation.

The output of a typical run includes data files listing the positions, velocities, and masses of each object at each step, and images of the density, kinetic energy, and potential energy of the system from three different angles. While all of the analysis is done on the numerical output of the simulation, the images direct what should be analyzed. Using a 3<sup>rd</sup> party program, we can stitch together these sequences of images into an animation.

### **Testing the Algorithms**

To make sure that the algorithms were working properly, a series of real-world and hypothetical tests were devised. The Particle-Particle algorithm is well suited for small simulations where precision is required to be high. An example of such a situation is a simulation of the solar system, which was one of the first systems tested using the PP code. For such a small simulation, we can allow the step-time to be set in the range of seconds, allowing for a super-precise representation of the solar system. The initial conditions for the solar system were taken from NASA's planetary fact sheet<sup>5</sup>, which allows us to define the orbits exactly as they are in reality. We can verify that the simulation is behaving correctly by tracing the paths of the orbits with Mathematica and comparing the shapes to the known orbital shapes. Once we are confident that the simulation is running properly, experiments can be run on the system as further validation. The example shown below is the result of a rogue star with the mass of the Sun passing between the Sun and Mercury. The behavior of the system shows that even under extreme conditions the simulation will operate realistically.



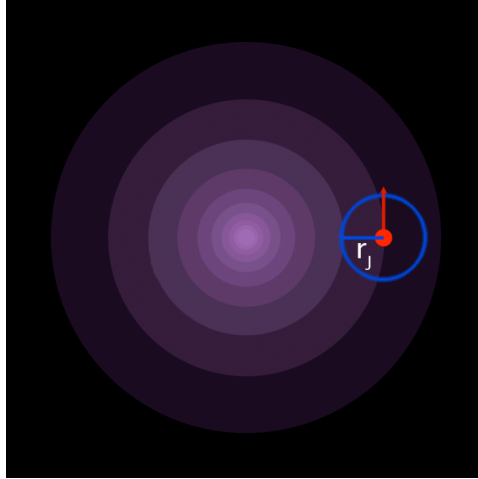
**Figure 3:** A rogue star passes between the Sun and Mercury

This sort of testing only gives us a visual confirmation of the code's success. A more mathematical test is to check that energy is conserved in the system, a process that was described beneath equation (2).

### **Applications: Satellite Falling Into a Dark Matter Distribution**

An interesting problem that can be investigated with the code is how light and dark matter react to tidal forces, and how a satellite with both types of matter will evolve over time when subjected to these forces. When a satellite object passes by a more massive object, particles beyond a certain distance from the satellite will cease to be bound to it. This distance is known as the *Roche Limit*, which is defined below<sup>6</sup> for a dark matter distribution where the density is proportion to  $r^{-2}$ , where  $r$  is the distance from the center of the distribution,  $m$  is the mass of the satellite, and  $M(<r)$  is the mass of the dark matter distribution within a sphere of radius  $r$ :

$$r_J = r \left[ \frac{m}{2M(<r)} \right]^{\frac{1}{3}} \quad (5.)$$



**Figure 4:** The Roche limit.

Since  $M(<r)$  is proportional to  $r$ ,  $r_J$  is proportional to  $r^{2/3}$ . Any objects that stray beyond this distance from the satellite will be stripped by the dark matter distribution and become bound to the larger object.

The setup of the simulation begins with a dark matter distribution with density proportion to  $r^{-2}$ , with a mass defined to be  $10^{12}$  solar masses within a radius of 300 kpc. The dark matter is treated as a simple potential, and is considered to have no sub-structure. The satellite object is a simple spherically symmetric dwarf galaxy with a mass of  $10^6$  solar masses and constant density out to a radius of  $\sim 300$  pc. In the real world, the presence of the satellite mass would perturb the distribution of the dark matter, but since a million times less massive than the dark matter distribution, the dark matter's properties remain constant over the course of the simulation. Prior to the main simulation, the particles are assigned peculiar velocities according to the Maxwell-Boltzmann particle distribution, as described in Hernquist's 1993 paper on compound galaxies<sup>7</sup>. The Maxwell-Boltzmann distribution is defined as:

$$F(v,r) = 4\pi \left( \frac{1}{2\pi\sigma^2} \right)^{3/2} v^2 \exp\left(-v^2/2\sigma^2\right) \quad (6.)$$

The velocity dispersion is taken to be:

$$\overline{v_r^2} = \frac{1}{\rho_h(r)} \int_r^\infty \rho_h(r) \frac{GM(<r)}{r^2} dr \quad (7.)$$

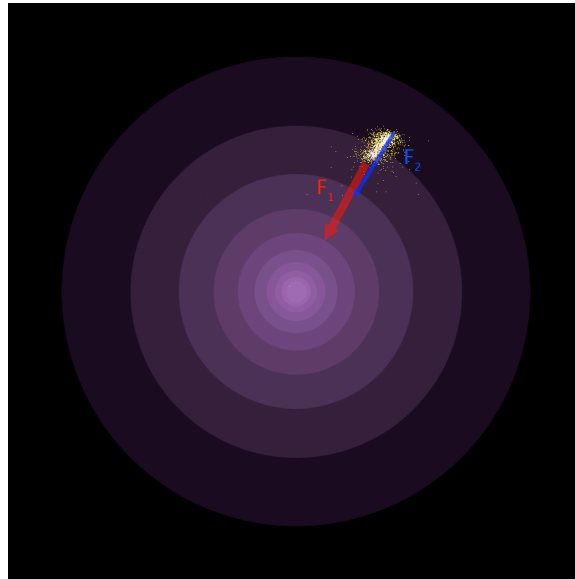
This simplifies greatly when the density  $\rho_h(r)$  is assumed to be constant. Particles are randomly assigned velocities that are fitted to the distribution function above. The maximum allowed velocity is at 95% of the escape velocity; if a particle is randomly assigned a velocity greater than this critical, a new velocity is randomly chosen. The formula for the escape velocity of an object orbiting a point mass is<sup>8</sup>:

$$\sqrt{\frac{2GM}{r}} \tag{8.}$$

As admitted by Hernquist, this distribution is not perfect, but the error is small and the inclusion of even one higher-order term produces good results. Even with near-perfect initial conditions, it's still a good idea to let the system evolve naturally for a while prior to inserting it into the simulation environment. Therefore, the initial conditions of the satellite galaxy are not terribly important beyond being able to define the size of the object a priori.

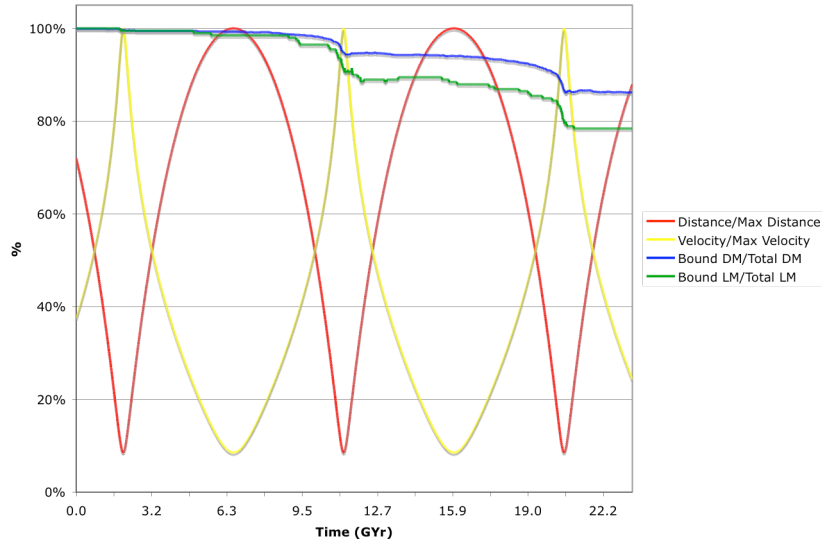
The particles are assigned velocities based on this distribution, and allowed to settle over the course of several relaxation times until the system is in equilibrium. The satellite's constituent objects are then tagged based on their kinetic energies; those in the lower 10% are tagged as light matter particles, while the remaining 90% is tagged as dark matter. Next, the satellite is placed near the dark matter distribution at position  $(-300 \text{ pc}, -100 \text{ pc})$  in  $x, y$  coordinates, and given an initial velocity in the  $x$  direction of  $10^5 \text{ m/s}$ .

The simulation is then allowed to run until the satellite has made several orbits of the parent dark matter distribution. As the satellite circles the dark matter, we keep track of the number of objects that remain bound to the dwarf galaxy. The criteria used to determine if an object is bound or not is if the velocity of the object is less than the escape velocity at that distance from the object. Objects that are unbound typically spend most of their time far away from the center of the satellite; therefore, we can take the satellite to be a point mass when considering how to calculate the escape velocity. For the mass of the object in the escape velocity calculation, we use the total bound mass.



**Figure 5:** Tidal forces tear apart the satellite object.

Below is a graph showing the bound light and dark matter as a percentage of the total light and dark matter, respectively. Also displayed are the satellite object's position and velocity as a percentage of its maximum position and velocity:



**Figure 6:** Bound mass and satellite position and velocity as a function of time.

Mass loss in the system is noted to be monotonic and periodic. An important feature is that the closer the satellite object is to the center of the system, the higher the rate of mass loss. This is expected; the tidal forces are greatest when the satellite passes the perigee. However, there are certain results shown that are not so obvious. One would expect that the mass loss would only depend on the mass of the object and the distance from the center, but mass loss ceases almost entirely once the object passes the perigee. The explanation for this behavior is revealed by looking at equation (5); as the object approaches the perigee, the Roche limit decreases because it is proportional to  $r^{2/3}$ . As the object passes the perigee and begins to travel away from the center of the dark matter distribution, the Roche limit increases. Therefore, while tidal forces continue to act on the satellite object, few objects are pulled beyond the Roche limit as the satellite begins to make its way to the apogee of its orbit.

Another facet that is difficult to explain is the dark matter vs. the light matter mass loss rate. From the results, it appears that a higher percentage of the dark matter remains bound to the satellite object. Given that the light matter objects were tagged because they had the least amount of energy in the system, and therefore the hardest time escaping the satellite's gravitational well, this result seems counterintuitive. An explanation for this behavior could be related to the small number of objects in the system. With only 200 light matter objects, it is possible that random chance has caused these objects to wander beyond the Roche limit at a higher rate than expected. Another possibility is that while the satellite was allowed to come to equilibrium over the course of several dynamical expansions and contractions, the system never fully reached equilibrium. This means that the light matter objects tagged as the ones with the lowest energy would gain energy as the satellite changed in size during the satellite's approach. While the cause of this discrepancy is still unknown, both of the problems mentioned above can be addressed by changing some of the simulation parameters; in this case, increasing the number of objects and allowing the satellite a longer period of time to reach equilibrium.

## Conclusion

The two codes that were written over the past year function as expected. Both of the codes that were written satisfy the main criteria of this endeavor: To build a flexible, accurate simulation that can run on a home computer in a reasonable amount of time. The codes have their weaknesses, but both have a niche that they fill rather well. However, there is plenty of room for improvement. The number of objects that can be simulated in a reasonable amount of time on a home computer is only on the order of  $10^4$ , and while the PP code was written to take advantage of a small computer cluster, both codes could benefit greatly with better multi-processor capabilities. To resolve many of the physics problems we encounter in this field, we definitely require that the simulations run with more objects. However, scientific study is still useful with the small set of objects that a home computer can handle. The results of the galaxy/dark matter interaction detailed above show how a small simulation can still be analyzed with great detail and compared to physical theory. The results of the simulations run over the course of the past year agree strongly with established physical behavior, and the harmony between the two allows for the project to be considered a success.

---

<sup>1</sup> Amara Graps. N-Body/Particle Simulation Methods.

<http://www.amara.com/papers/nbody.html>

<sup>2</sup> Roger W. Hockney and James W. Eastwood. Computer Simulation Using Particles. McGraw-Hill, 1981.

<sup>3</sup> Stephen T. Thornton and Jerry B. Marion. Classical Dynamics of Particles and Systems. Thomson Learning, Belmont California, 2004.

<sup>4</sup> Lars Hernquist. Performance Characteristics of Tree Codes. The Astrophysical Journal Supplement Series, 64:715-734, 1987 August

<sup>5</sup> David R. Williams. Planetary Fact Sheet.

<http://nssdc.gsfc.nasa.gov/planetary/factsheet/>

<sup>6</sup> Linda S. Sparke and John S. Gallagher. Galaxies in the Universe. Cambridge University Press, Cambridge United Kingdom, 2000.

<sup>7</sup> Lars Hernquist. N-Body Realizations of Compound Galaxies. The Astrophysical Journal Supplement Series, 86:389-400, 1993 June

<sup>8</sup> James Binney and Scott Tremaine. Galactic Dynamics. Princeton University Press, Princeton New Jersey, 1987.