Henry Jin, Alex Kashi, Albert Ge
AM205: Advanced Scientific Computing Numerical Methods
Fall 2021

Methods on N-Body Simulation

AM205 Final Project

Introduction & Motivation

The *n*-body problem is a well-studied branch of astrophysics concerning the prediction of individual motions of a group of celestial objects. Each individual entity in a group of objects exerts a number of forces on each other object, largely governed by gravitational force. Solving for the motion of a group of bodies has been useful to practitioners and enthusiasts alike - astronomers who observe planetary systems want to understand behavior and interactions between different clusters of objects, and hobbyist stargazers who want to observe the positions of notable objects within our solar system in the night sky.

As its name suggests, the *n*-body problem scales in complexity and scope as the number of celestial objects, *n*, increases. Beginning with just n ≥ 3, the *n*-body problem becomes extremely difficult to solve analytically. The three-body problem, in particular, has been particularly of interest, perhaps in part due to its deceptively simple problem formulation, and has a centuries-long dedication of effort to its solution [2]. Recent research has shown that it is actually theoretically possible to calculate the solutions to the *n*-body problem as an infinite series [3], but this is neither feasible nor practically useful in making a realistic prediction of the motion of bodies.

The alternative is to perform a computational approximation of the objects' motion. The general idea is that, knowing the differential equations of object motions, the equations of object position can be numerically approximated via integration with an appropriate step size. A variety of numerical integration methods exist in the literature [4] [5] for solving the n-body problem, with varying degrees of accuracy and stability. Overall, numerical integration has been shown to be both effective and reliable for predicting motions of planetary systems.

Another dimension of these numerical methods for solving the *n*-body problem is to consider computational cost. The dynamics of a group of celestial objects requires that, at every time step of the computation, the gravitational forces exerted must be recomputed for every pair of objects in the system. This results in computing the pairwise interaction of forces between each such pair of objects, and this becomes increasingly expensive to compute as *n* increases. There are also methods aimed at improving the computational efficiency of such programs, showing that these simulation methods are also scalable in size and scope of many-body systems. [5] [6] [7].

Our report presents a survey of different integration schemes and simulation algorithms. In particular, we choose to highlight the Barnes-Hut method, which is a simulation method that scales as *n*-log-*n* - that is, almost linearly with the number of bodies observed in our system. To

qualitatively compare various integration strategies we designed an optimized C++ gui to render the results of large n-body simulations.

Our report is organized as follows: we first present a survey of different numerical integration schemes, and report error and stability analyses for each such scheme. We then dedicate the remainder of this paper to showing and discussing performance benchmarks for different simulation methods.

Background & Context

N-body Problem Formulation

A simple mathematical formulation of the n-body problem is described as such:

Assume each celestial object can be represented by a point mass at its center of gravity. The *n*-body problem considers *n* point masses m_i, *i = 1, 2, …, n.*

From Newton's second law of motion, the force on an object, mass times its acceleration, is equal to the sum of the forces on the object. The force on object *i* exerted by object *j* can be written as

$$F_{i,j} = \frac{Gm_i m_j}{||q_j - q_i||^2} \cdot \frac{q_j - q_i}{||q_j - q_i||} = \frac{Gm_i m_j (q_j - q_i))}{||q_j - q_i||^3}$$

Where *G* is the gravitational constant, q is the position of each respective object *i, j*, and

$$||q_j - q_i||$$

Is the norm of the distance between the two objects.

Applying Newton's second law of motion, the force exerted on object is the sum of the forces on the object:

$$F_i = \sum_{j \neq i}^{n} \frac{Gm_i m_j (q_j - q_i)}{||q_j - q_i||^3}$$

Importantly, note that the force *F* is composed of multiple components. In two dimensions, each component of the force can be written as

$$F_{i,x} = \sum_{j \neq i}^{n} \frac{Gm_i m_j (q_{j,x} - q_{i,x})}{||q_j - q_i||^3}$$

$$F_{i,y} = \sum_{j \neq i}^{n} \frac{G m_i m_j (q_{j,y} - q_{i,y})}{||q_j - q_i||^3}$$

Where

$$(q_{j,y} - q_{i,y})$$

Represents the difference in the y-component for the positions of objects *j* and *i*. A simple extension can be made for generalizing to three dimensions.

Numerical Integration

A simple numerical integration scheme for solving the above *n*-body problem, can be described with the following algorithm:

1. Solve for the force *F*, using the equations above
$$F_i = \sum_{j \neq i}^{n} \frac{G m_i m_j (q_j - q_i)}{||q_j - q_i||^3}$$
2. Since F=ma, use the force *F* to solve for acceleration.
$$a = \frac{F_i}{m_i}$$
3. Use acceleration to update the velocity by a small timestep, dt.
$$v_{t+1} = a_t dt$$
4. Use velocity to update the position by a small timestep, dt.
$$x_{t+1} = v_t dt$$

Symplectic Methods and Hamiltonian Systems

While not within the scope of our project, we briefly discuss the importance of symplectic integrators and their applications on simulating the *n*-body problem.

A Hamiltonian system is a formulation of Hamiltonian mechanics which describes how a physical system evolves over time. Hamiltonian systems are often used to describe complex time-evolving systems, such as planetary systems. The connection between Hamiltonian systems and the symplectic integrator is that the system contains symplectic properties.

The advantage of a symplectic method is that it conserves physical properties like angular momentum and energy of the system. It's been shown that certain numerical integrators, such as the leapfrog method, is symplectic [11], and this motivates the following section comparing the stability and accuracy of various integration schemes.

Numerical Integration Methods

Forward Euler

From the numerical integration algorithm outlined above, the forward euler method updates the position at a time $t + \Delta t$ with the velocity from the previous timestep. We will refer to this integration method throughout this report as the naive method.

Leapfrog Method

From the numerical integration algorithm outlined above, the leapfrog method updates the position at a time $t + \Delta t$ with the velocity from the the midpoint, $t + \frac{\Delta t}{2}$. This method is symplectic, since it preserves the symmetry of the Hamiltonian equation.

The following rules give us one method for implementing the leapfrog method.

$$\begin{aligned} v_{i+1/2} &= v_i + a_i \frac{\Delta t}{2} \\ x_{i+1} &= x_i + v_{i+1/2} \Delta t \\ v_{i+1} &= v_i + a_i \Delta t \end{aligned}$$

At any given time step *i*, we set the midpoint velocity by using the acceleration and a small timestep $\frac{\Delta t}{2}$. We can then update the new position of the object by using the midpoint velocity, and then update the new velocity as normal. In this way, the position and velocity are kept at a half-timestep out of phase, with the velocity slightly ahead.

Simulation Methods

Pairwise Interactions

The naive method for simulating the gravitational interaction between many bodies entails calculating every possible pairwise interaction. For a system of n bodies, this is precisely *n(n-1)* possible pairs. This algorithm therefore grows in $O(n^2)$, making it a relatively intractable solution to the many bodies problem, particularly with large *n*. The algorithm is iterative, and simply uses two for loops as follows:

1. For each body A:
    a. For every other body B:
        i. Calculate the force on A by B and add it to A's force

Due to the poor performance of this method, we use Barnes Hut Algorithm to gain efficiency.

Henry Jin, Alex Kashi, Albert Ge
AM205: Advanced Scientific Computing Numerical Methods
Fall 2021

Barnes Hut Algorithm

The Barnes Hut Algorithm is a recursive algorithm that allows for simulating the many bodies problem in *O(n log n)* complexity by the number of bodies. Compared to the naive solution of calculating all possible pairwise interactions, which goes by $O(n^2)$ complexity, Barnes Hut Algorithm provides significant time improvement for very large *n*.

At a high level, the Barnes Hut algorithm provides speed improvement by aggregating many different bodies together when such approximation is sufficiently accurate. For instance, if we want to calculate the force on a body A caused by a cluster B of several bodies far away, it is sufficient to treat this cluster as one body by calculating its center of mass and total mass and computing a single force rather than each pairwise force with body A and each body in cluster B. To enable this aggregation, the Barnes Hut algorithm organises the state into a tree structure, where nodes represent regions of space. If a node is sufficiently far from the body A, then all bodies in the node are aggregated and one force calculation is made.

The algorithm works in a two step process: constructing the tree, and calculating the forces.

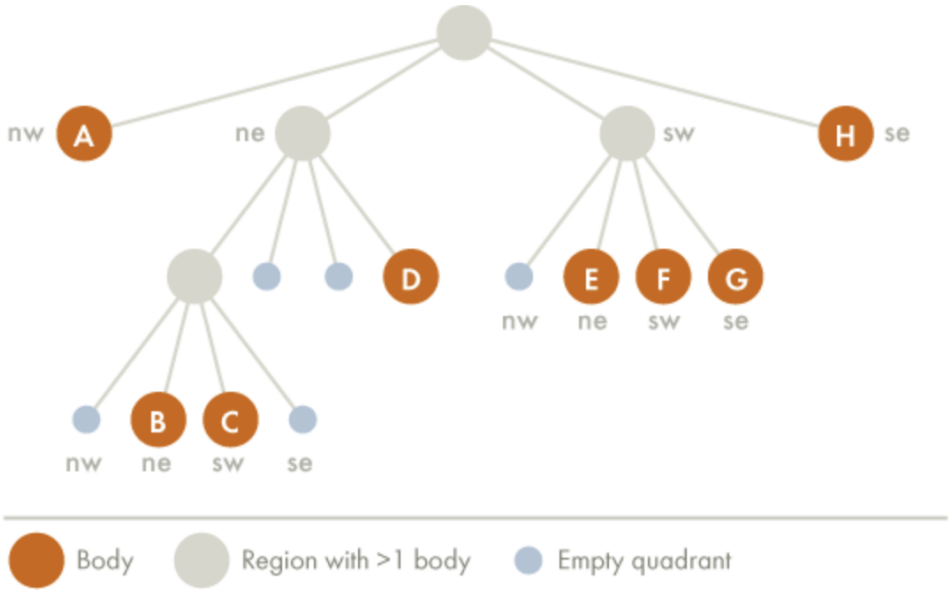


Figure provided by http://arborjs.org [10]

First, we discuss tree construction. Each node of the tree represents a subspace of the region. Each child of each node is a subspace of its parent node. As such, the root node represents the full space. In 2 dimensions, each node has four children (space is divided into quadrants). In 3 dimensions, each node has 8 children (space is divided into octants). The tree consists of three types of nodes: internal, external, and empty nodes. Internal nodes represent a region of space that has more than one body (coloured gray in the image above). External nodes are leafs and can only represent a single body (coloured orange). Empty nodes are regions of space that have no bodies (coloured light blue). In order to construct such a tree, we simply initialize the

root node, then we add bodies sequentially to the root node, which recursively adds the body to the appropriate children nodes. The recursion for tree construction is as follows:

1. If the current node is an empty node, place the body here.
2. If the current node is an internal node, update its center of mass and total mass, and add the body to the appropriate child node.
3. If the current node is an external node (which already contains a body), convert the current node to an internal node, update its center of mass and total mass, initialise children nodes, and add the new body and old body to the appropriate children nodes.

The second part of Barnes Hut is about calculating the force on a body. This process is also recursive, starting from the root node. The recursion for calculating the force on a body A is as follows:

1. If the current node is an external node (that is not body A), calculate the force exerted by the current node on body A and add this to body A's force experienced.
2. If the current node is an internal node, calculate the width of the node's region and divide it by the distance between body A and the node's center of mass. If this quantity is less than a threshold theta (which is usually set to 0.5), then we use the current node's center of mass and mass to calculate a single force approximation for all the nodes below the current node.
3. If the current node is an internal node, but the width over distance quantity is greater than theta, then we call the force calculation function on each of the current node's children.

To simulate the evolution of a system of bodies then, we follow the following steps:

1. Initialise the system of bodies by initialising their masses, positions, and velocities.
2. For each iteration:
    a. Construct the Barnes Hut tree
    b. For each body:
        i. Calculate the force on each body
    c. For each body:
        i. Update position
        ii. Update velocity

The computational improvement over the naive method comes from the aggregation step when calculating the force on each body. When the approximation condition is met, the recursion is terminated and the aggregate of bodies in that region is calculated using the center of mass and total mass of the corresponding node. This is what enables a complexity of O(n log n).

We demonstrate the barnes hut algorithm for both the 2D and 3D cases, on a five-body simulation. Initial conditions were randomly drawn from a uniform distribution, to test a variety of

planetary trajectories. The following figures show a plot of their trajectories in both 2D and 3D plots, respectively. Of note, we observe that for randomly-sampled initial conditions, the system appears to drift slowly from the origin away from time. This may likely be due to a nonzero momentum in our initial conditions - thus the system will move away from the origin of space.

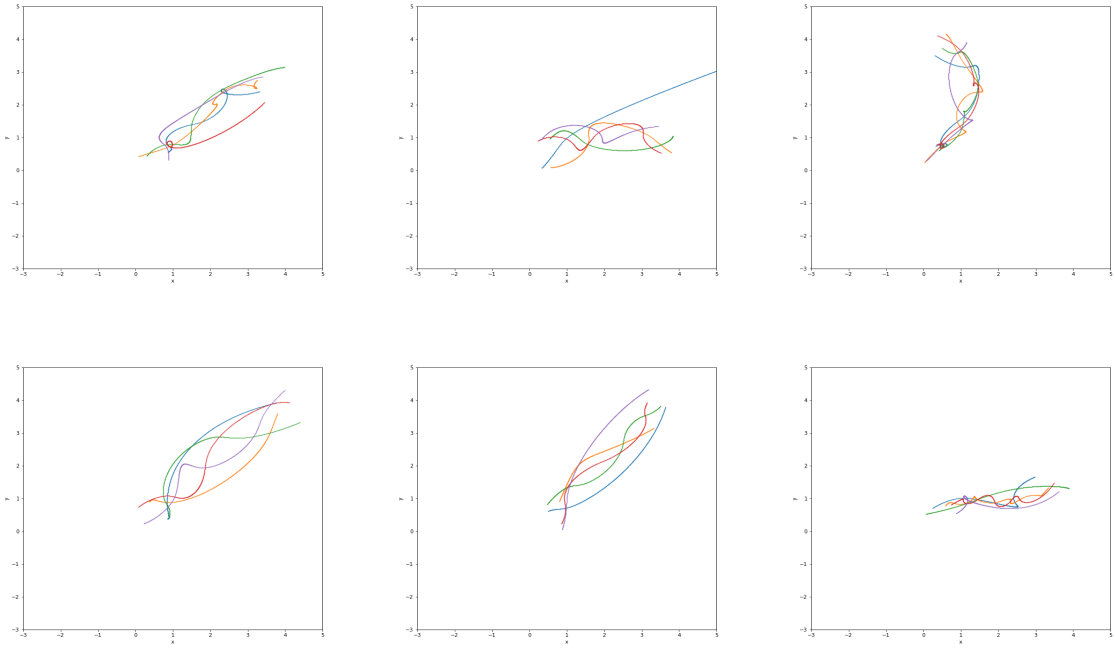


Figure: 2D simulations of 5 bodies

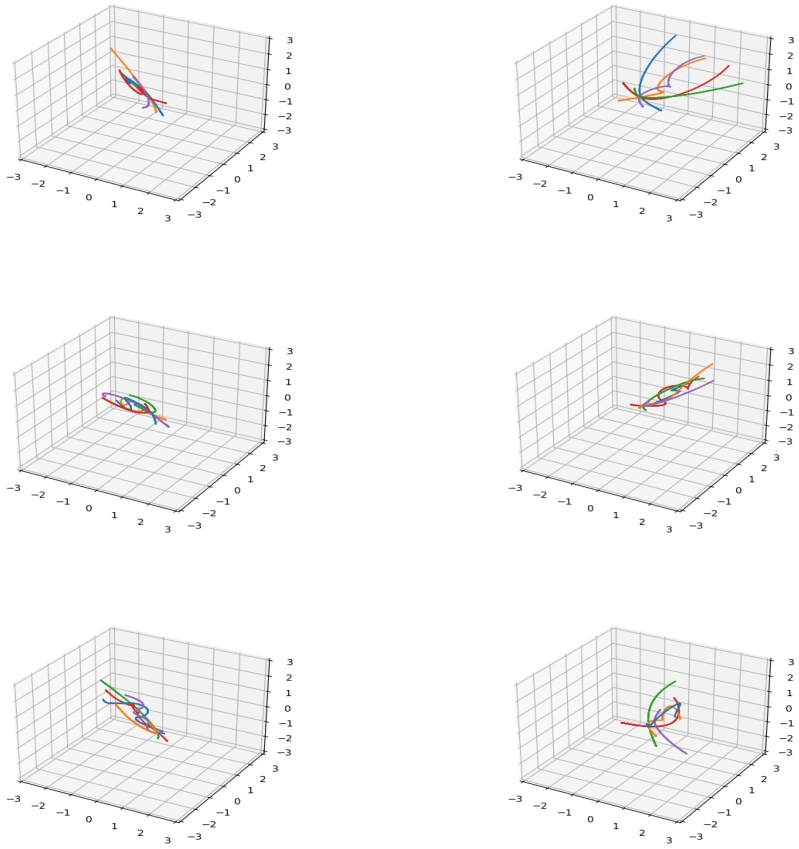


Figure: 3D simulations of 5 bodies

Error and Stability Analysis

Choosing an ideal numerical integration scheme motivates an error analysis of each method, and defining a metric for determining its accuracy. Here we define two ways of examining numerical integration error, and demonstrate these metrics applied to a variety of model systems.

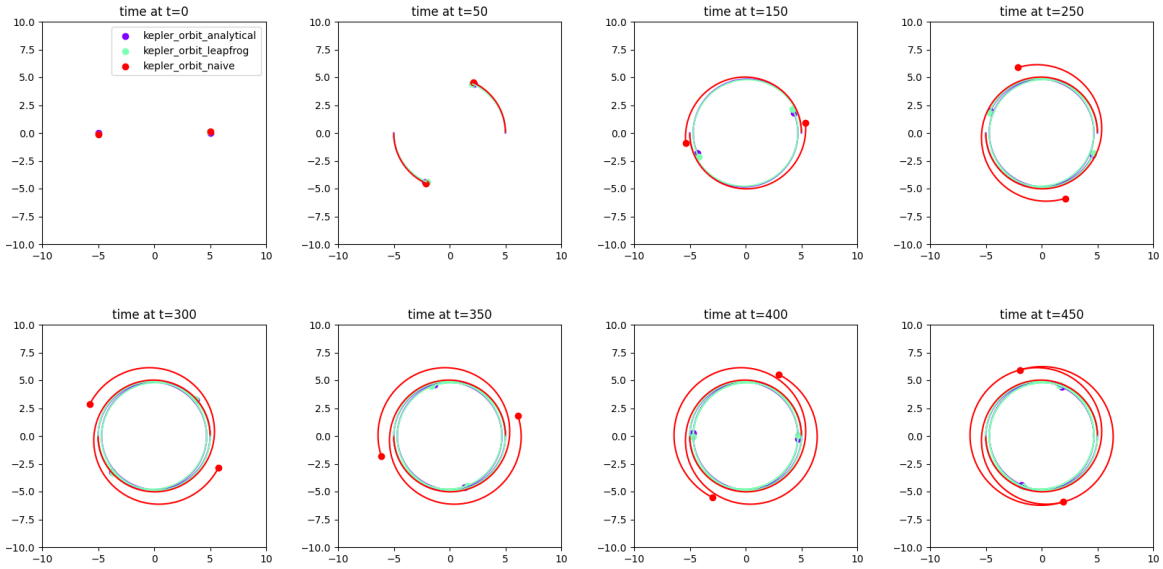
Two-body problem

We first compare the visual differences between numerical integration and an analytical solution for the two-body problem. Since there exists a closed-form solution for the two-body problem, we interpret this as our ground truth, and can directly compare the different integration schemes against it.

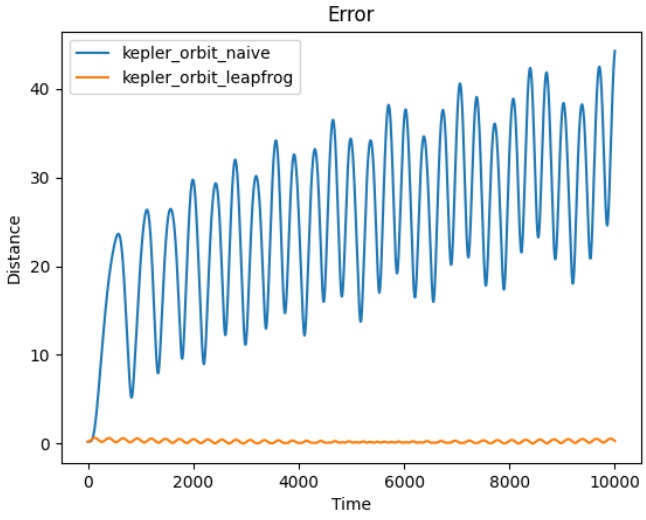
To solve for the analytical solution of the two-body problem, we use the equations for gravitational force, and utilize SciPy's odeint routine to numerically solve for the positions as a function of time.

Henry Jin, Alex Kashi, Albert Ge
AM205: Advanced Scientific Computing Numerical Methods
Fall 2021

The following plot shows snapshots of the different two-body simulations, taken at equally spaced points in time. The figure shows, at the onset of the simulation, all three methods (leapfrog, naive, and analytical), have the same initial conditions, and over time, the naive method drifts apart, while the analytical and leapfrog method remain closely in sync. An interesting feature about the naive method is that the simulation appears to fan-out gradually over time, increasing in radius from the barycenter of the two-body system.

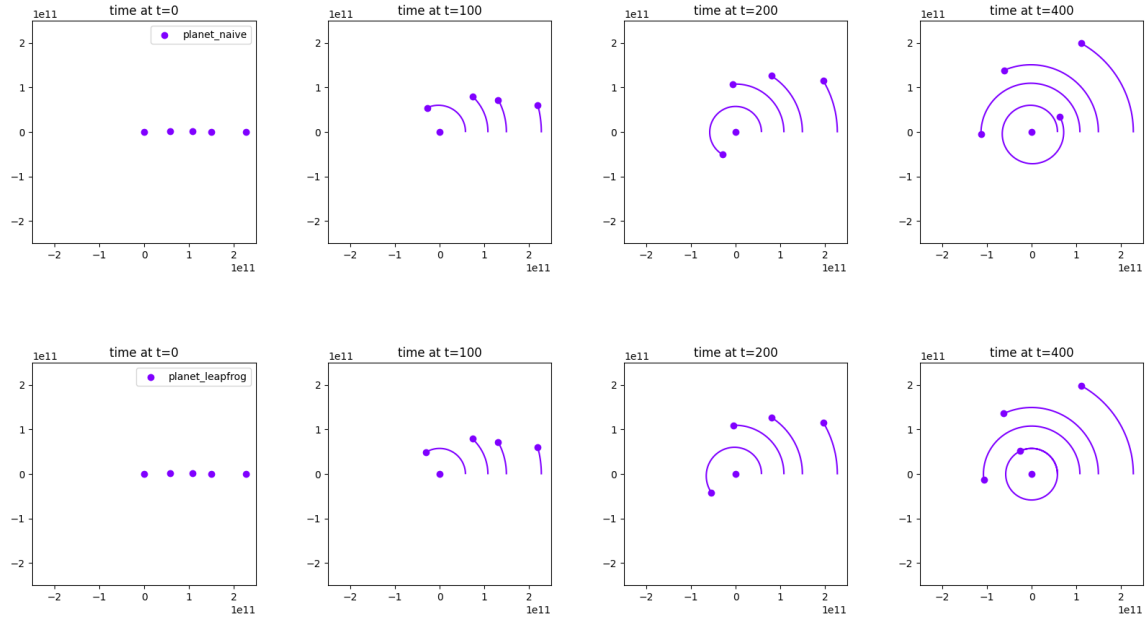


To help quantify these visual trends, we additionally plot the error as Euclidean distance between the numerical integration schemes and the analytical solution. We observe that there is an oscillatory pattern between the naive method integration and the analytical solution, due to the differences in periodicity. This image also shows a clear picture of the fan-out effect - gradually, the naive solution will drift away from both the analytical and leapfrog methods.



Solar System

We now observe the differences between the two integration methods for more complex planetary systems. We simulate a 5-body system modeled after our Solar System, and record their dynamics.



Similar to the two-body problem, we observe that the leapfrog method looks fairly stable - the planets orbit the sun in a closed trajectory. On the other hand, the naive method appears to spiral out away from the sun over time.

Without a closed-form solution for the *n*-body problem with more than 2 bodies, we cannot use the same error metric as before in comparing numerical methods to the analytical solution. Thus we motivate the following section on energy conservation to define a metric for physically accurate simulations.

Conservation of Energy

For a closed and isolated system, its total energy must be conserved. For an astronomical system of bodies, the total energy (*E*) consists of gravitational potential energy (*PE*) and kinetic energy (*KE*). This is given by the following equations, where *i* represents the index of each body.

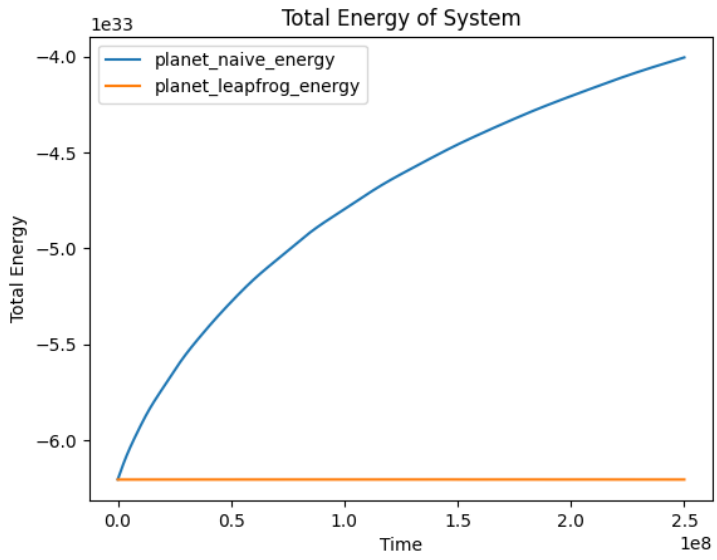
$$KE = \frac{1}{2}\sum_{i=1}^{n} m_i v_i^2$$

$$PE = -\frac{1}{2}G\sum_{i}^{n}\sum_{j\neq i}^{n}\frac{m_i m_j}{|r_i - r_j|}$$

$$E = KE + PE$$

A body's kinetic energy is simply one half of its mass times its velocity squared. A body's gravitational potential energy is negative one half of the gravitational constant times the sum of every pairwise mass product divided by the distance separation. The total kinetic energy and potential energy for the system then is the sum of each body's kinetic energy and gravitational potential energy. This total energy of the system should remain constant through time.

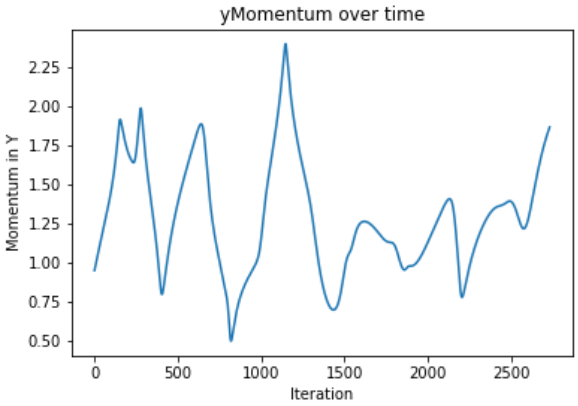
The following figure shows the plotted total energy of the solar system example over time. Since the leapfrog method is a symplectic integrator, we observe that the total energy stays relatively fixed throughout the duration of the simulation. On the other hand, the forward Euler's method shows that the energy of the system monotonically increases over time. This shows, therefore, that the leapfrog method is a more physically accurate method for numerical simulation.

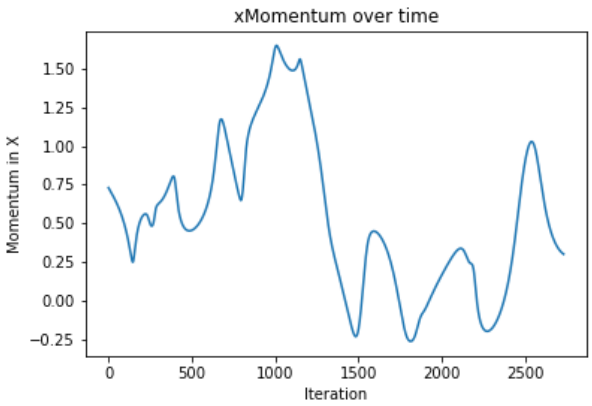


Conservation of Momentum

Similarly, for a closed and isolated system, its total momentum must also be conserved. For a system of many bodies, the total momentum in each dimension (x, y, z) is calculated by taking the sum of the product of each body's mass and velocity in said direction. Then, throughout a numerical evolution of this system, this value should remain constant.

The figures below show the x and y dimension momenta of a 2-dimensional simulation with Barnes Hut on a five-body simulation. We found that the momenta did not stay at a constant value throughout. However, the momenta seemed to hover within a bound, and did not grow out of control. This is reasonable considering that Barnes Hut is an approximation.



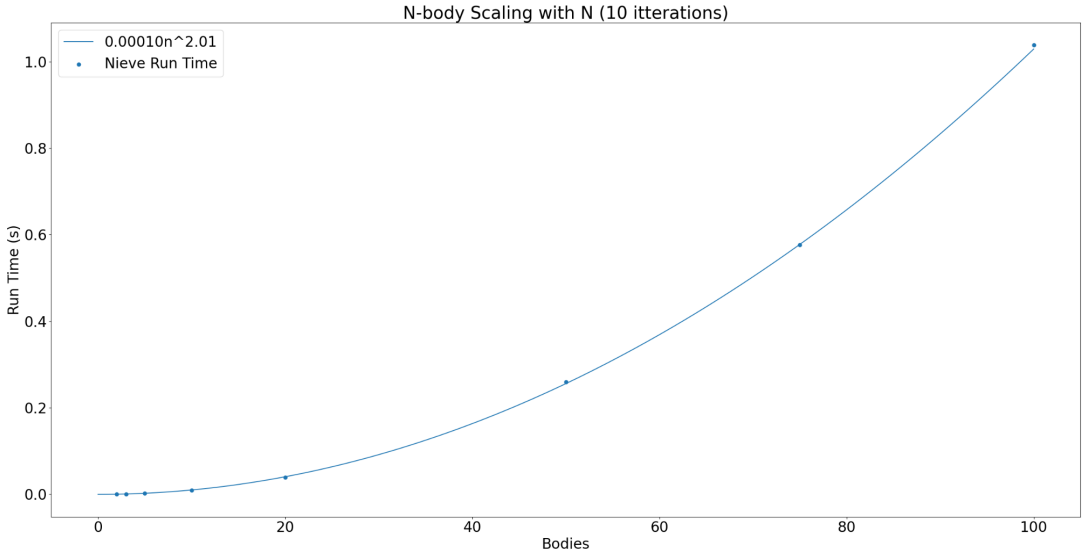


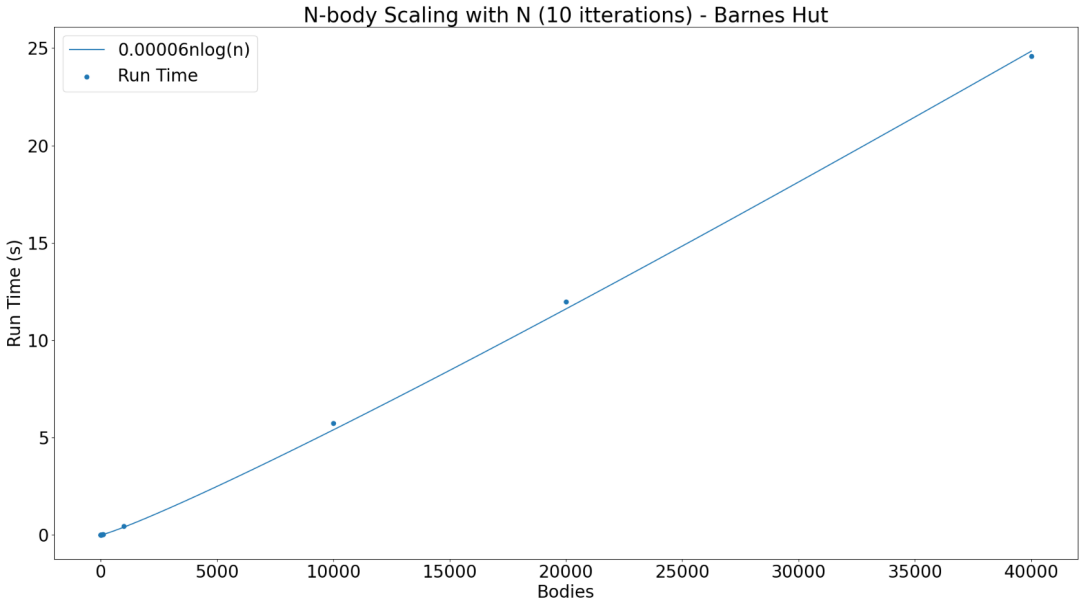
Henry Jin, Alex Kashi, Albert Ge
AM205: Advanced Scientific Computing Numerical Methods
Fall 2021

Performance Benchmarks

Runtime

We performed a runtime analysis on our python implementation of the brute force n-body problem and the Barnes hut algorithm. We confirmed that the Naive algorithm indeed scales as $O(n^2)$ and Barnes Hut scales as O(nlog(n)) where n is the number of bodies. Curves were fitted using linear least squares using a method similar to AM205 HW2 Q4.





3D Simulations

We designed a 3D Galaxy Simulator C++ using Qt [9]. Qt is the leading product for creating high-performance graphics applications across many platforms. It comes with an IDE and is known to have a steep learning curve and strict licensing requirements. However, when mastered, Qt is second to none in performance and is deployed on many real-world systems.

Our simulator loads a CSV file containing the position of all of the simulated bodies at every timestep resulting in a matrix that has dimensions of Nx3B. And the columns are the positions of each body at that time step: [b_{1x}, b_{1y}, b_{1z}, b_{2x}, b_{2y}, b_{2z}, ... b_{nx}, b_{ny}, b_{nz}]. Our largest simulation of 3000 iterations of 25000 bodies resulted in a CSV file nearly 6GB in size. Substantial performance gains could be realized by storing the data in a binary format, this is one extension and would be critical for scaling the simulations beyond 100k bodies.

Galaxy Initialization and special parameters

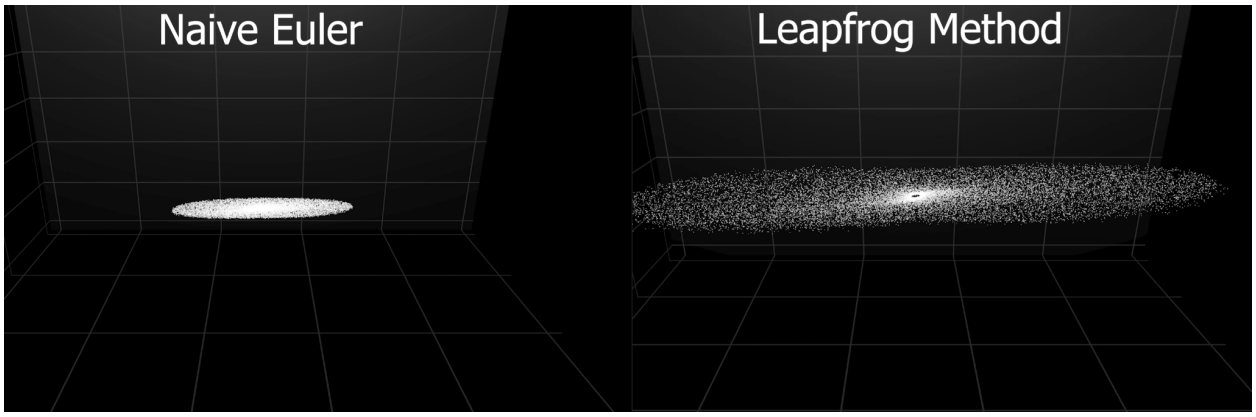
To initialize a galaxy, we randomly distribute bodies of a unit mass radially around an origin point, using polar coordinates. We also provide the bodies with a random linear velocity tangent to a circular trajectory. While the x and y components of the position and velocities are uniformly distributed, the z position and acceleration is normally distributed, to provide a more disk-like galaxy formation. At the center of each galaxy, we placed a body of m=500 500x greater than every other body in the system (m=1). This center mass was placed to help the galaxies hold their shape better while being pulled apart by other more massive clusters.

We created a parameter ε that would limit the distance between any two bodies to ameliorate any numerical instabilities caused by bodies coming too close together. Empirically, we found that setting ε=1 provided a stable simulation when N > 6000. Where lower epsilon values would cause many bodies to be ejected when a large number of bodies were being simulated, in a simulation of this size, you would expect at least some bodies to be ejected. However, we chose a value of epsilon that limited this because the root of the Barnes Hut tree covers a finite region, and expanding the region comes with an additional computational cost. To compensate for the low mass of the particles, we set Newton's gravitational constant G to 100/B, where B is the number of particles.

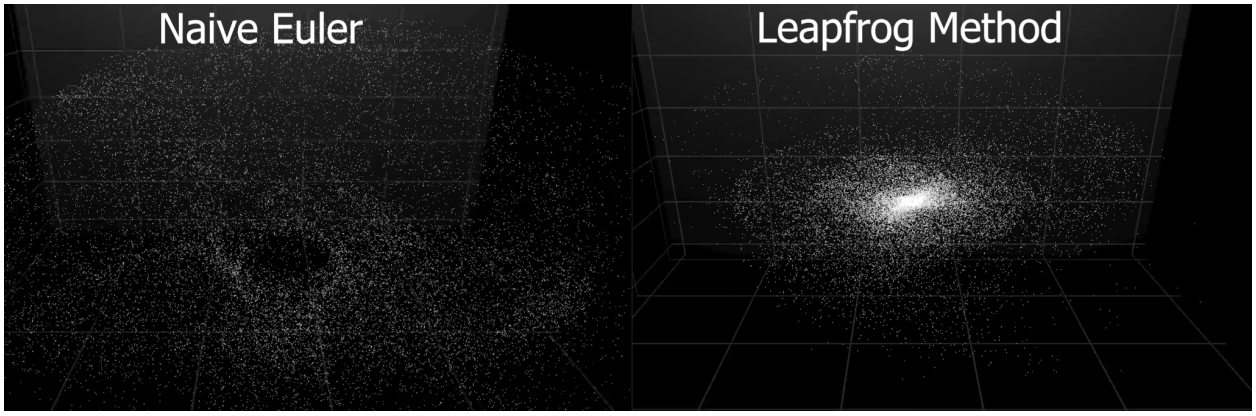
Henry Jin, Alex Kashi, Albert Ge
AM205: Advanced Scientific Computing Numerical Methods
Fall 2021

One Galaxy Simulation B=25000 N=3000 dt=1e-3

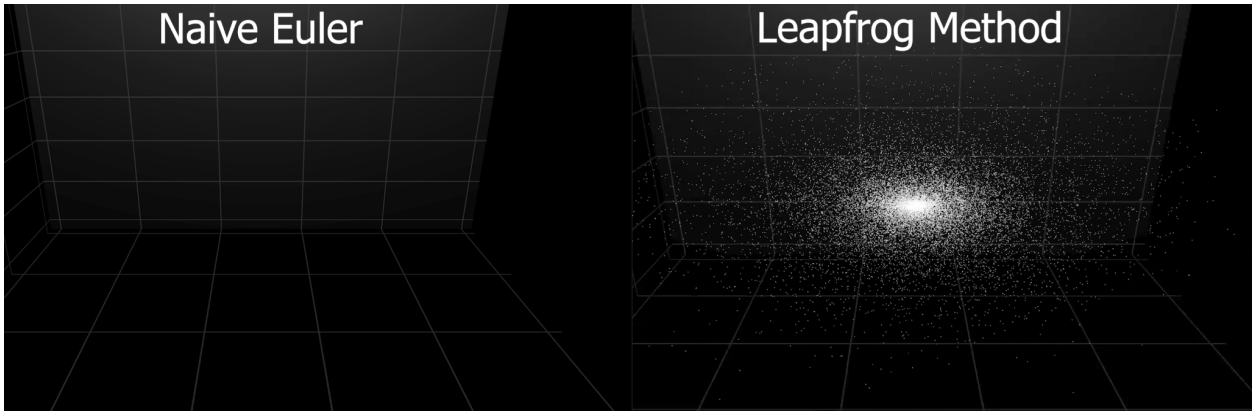
Galaxy Simulation Euler vs Leepfrog method - Harvard AM205 Fall 2021



One Galaxy initial conditions



One Galaxy Midpoint



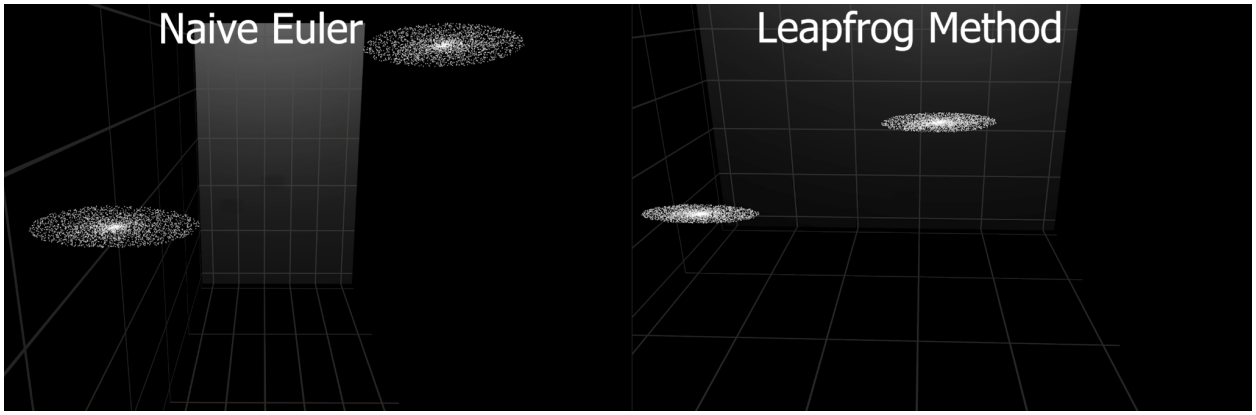
One Galaxy End

In this simulation, we compare Euler's method and the Leapfrog method for a one galaxy case. The naive approach tends to expand and drift, which is consistent with an increase in the

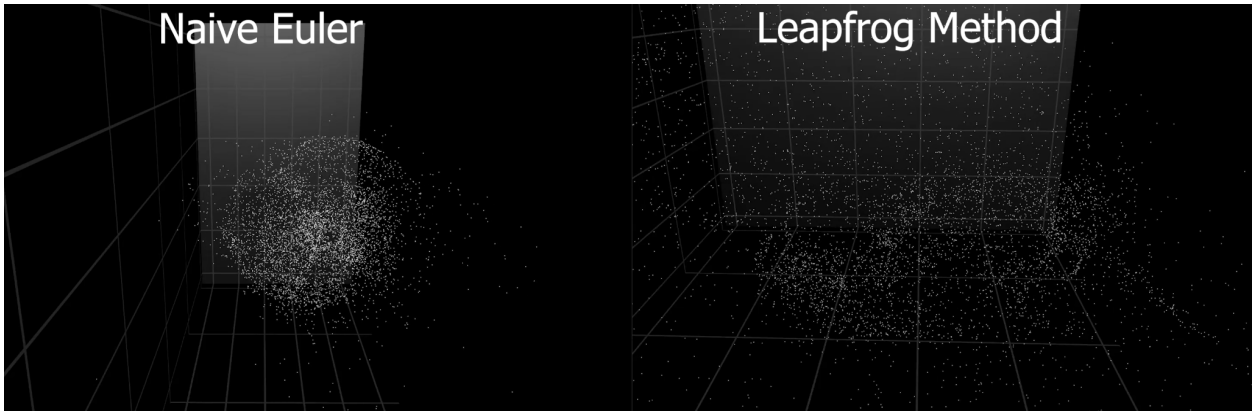
system's total energy, as reported earlier in this manuscript. The leapfrog reaches a stable orbit with many bodies clustered near the center of mass.

Two Galaxy Simulation B=6000 N=1000 dt=1e-3

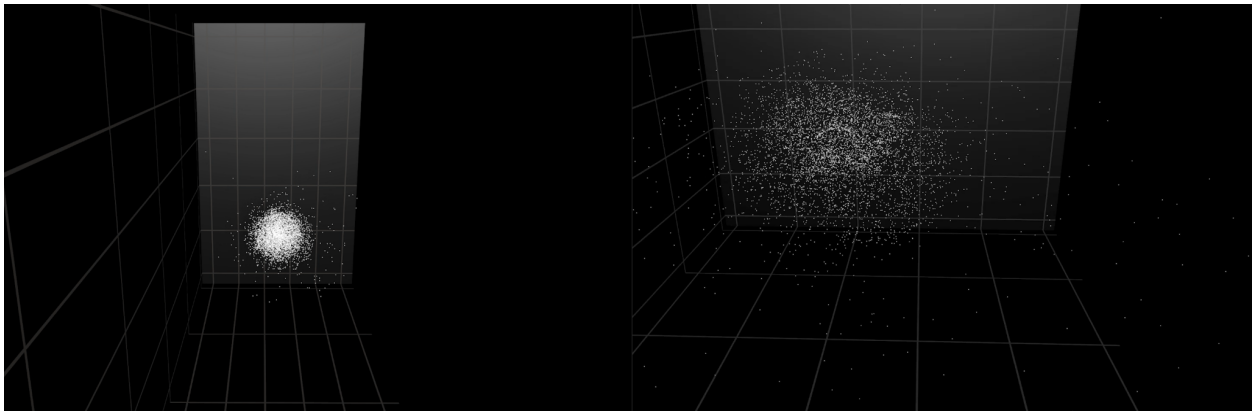
Two Colliding Galaxies - AM205 Fall 2021



Two Galaxies initial conditions



Two Galaxies Midpoint



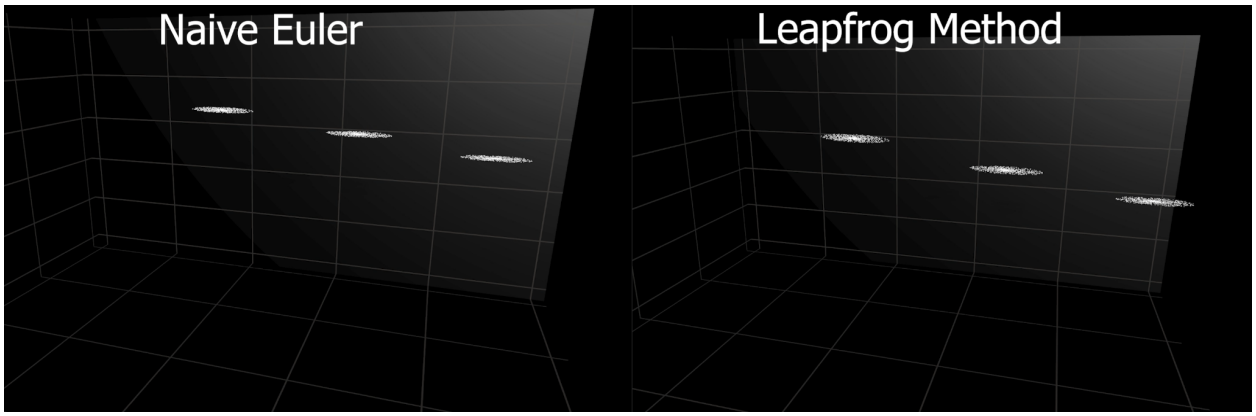
Two Galaxies End

Henry Jin, Alex Kashi, Albert Ge
AM205: Advanced Scientific Computing Numerical Methods
Fall 2021

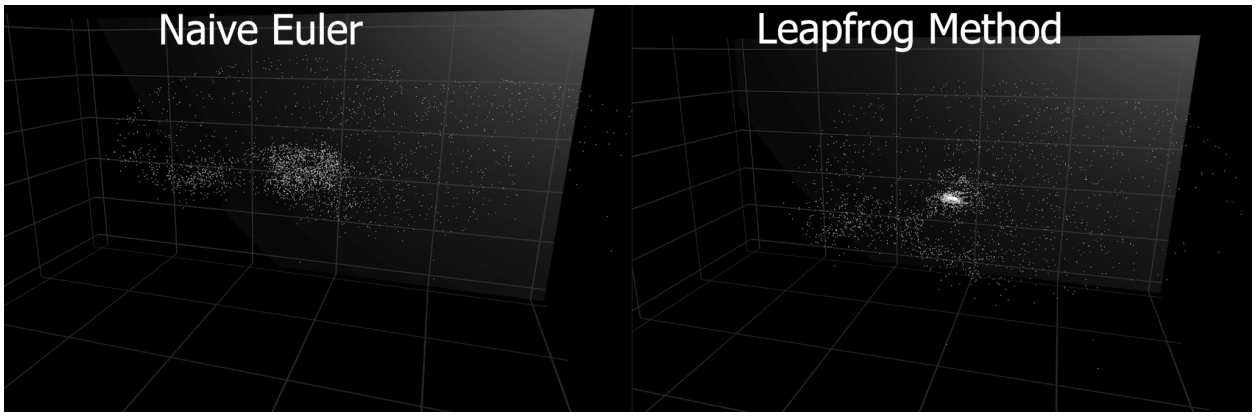
In this simulation, we compare how Euler's method and the Leapfrog method perform while in the case where two galaxies collide. The simulation using the naive forward Euler integration results in a system where the whole cluster drifts with a large positive velocity, while the center of mass in the leapfrog method stays in a relatively fixed location.

Three Galaxy Simulation B=6000 N=1000 dt=1e-3

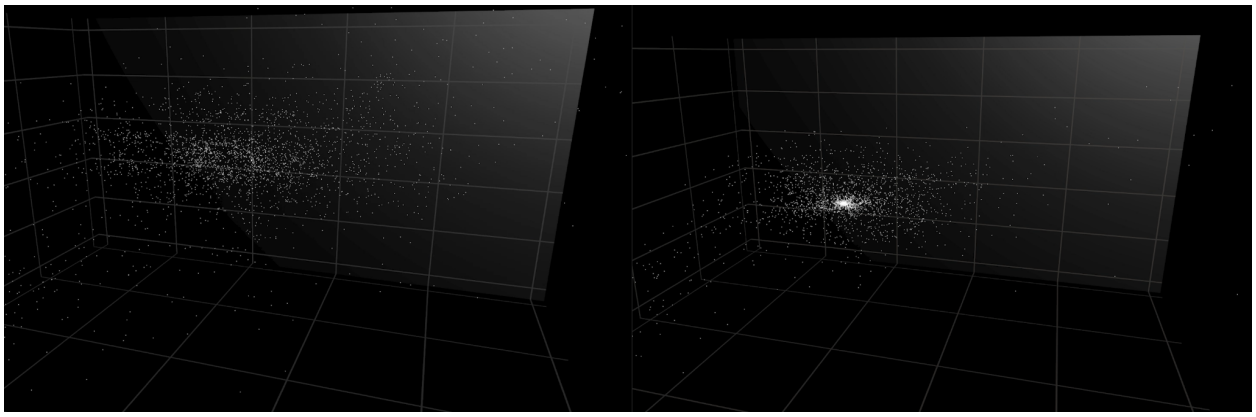
Three Colliding Galaxies (6000 bodies) - Harvard AM205 Fall 2021



Three Galaxies initial conditions



Three Galaxies Midpoint

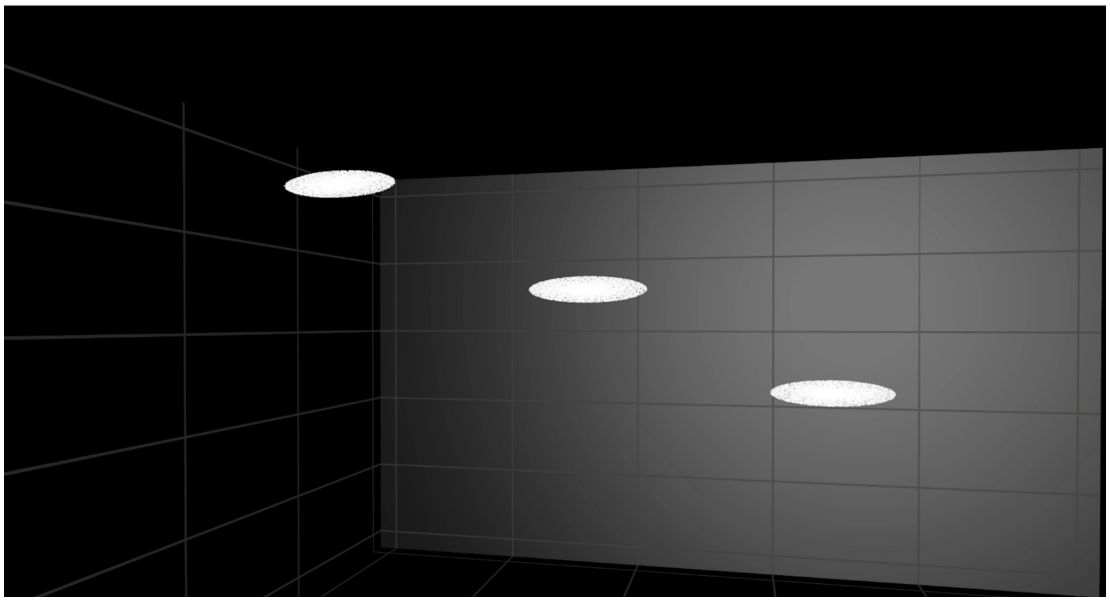


Three Galaxies End

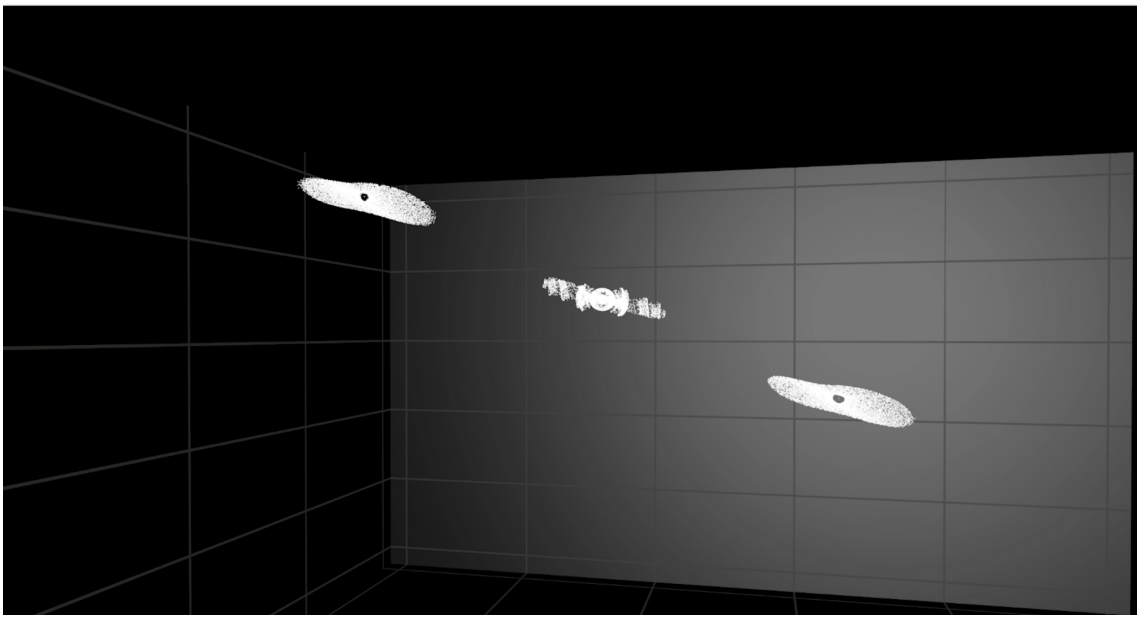
In the three galaxy case, the naive Euler method tends to expand over time, signifying an increase in total energy caused by numerical issues, whereas the leapfrog method provides a much more realistic simulation with a dense cluster of bodies near the center of mass and where most of the bodies reach a stable orbit.

Large Three Galaxy Simulation B=60000 N=1000 dt=1e-3

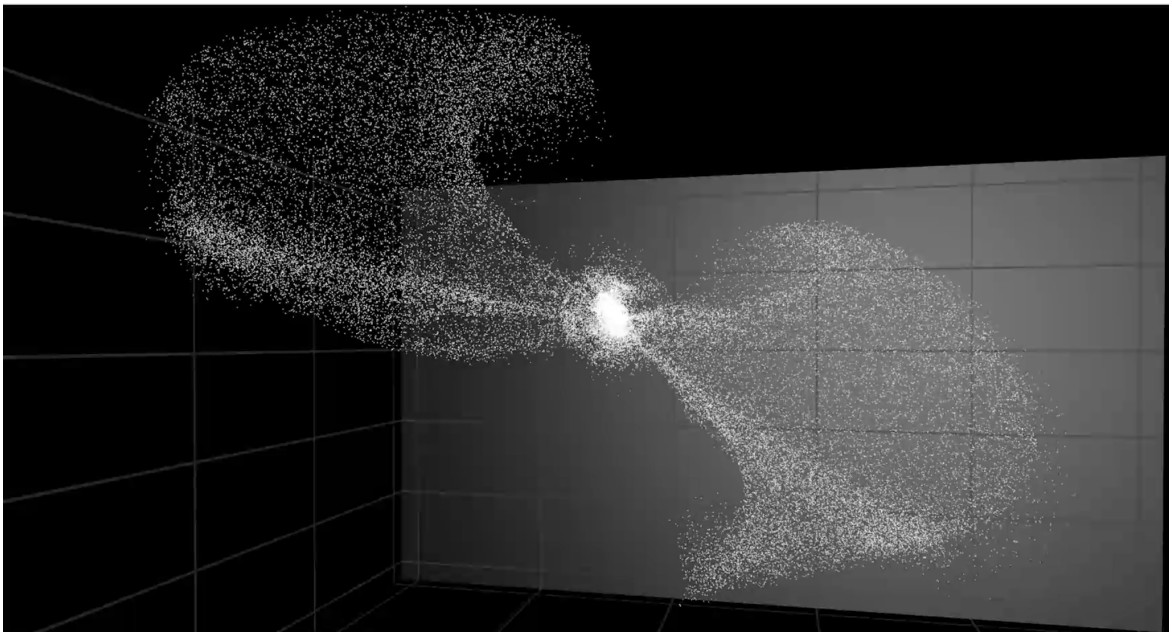
Galaxy Simulation 60,000 Bodies - Harvard AM 205 Fall 2021



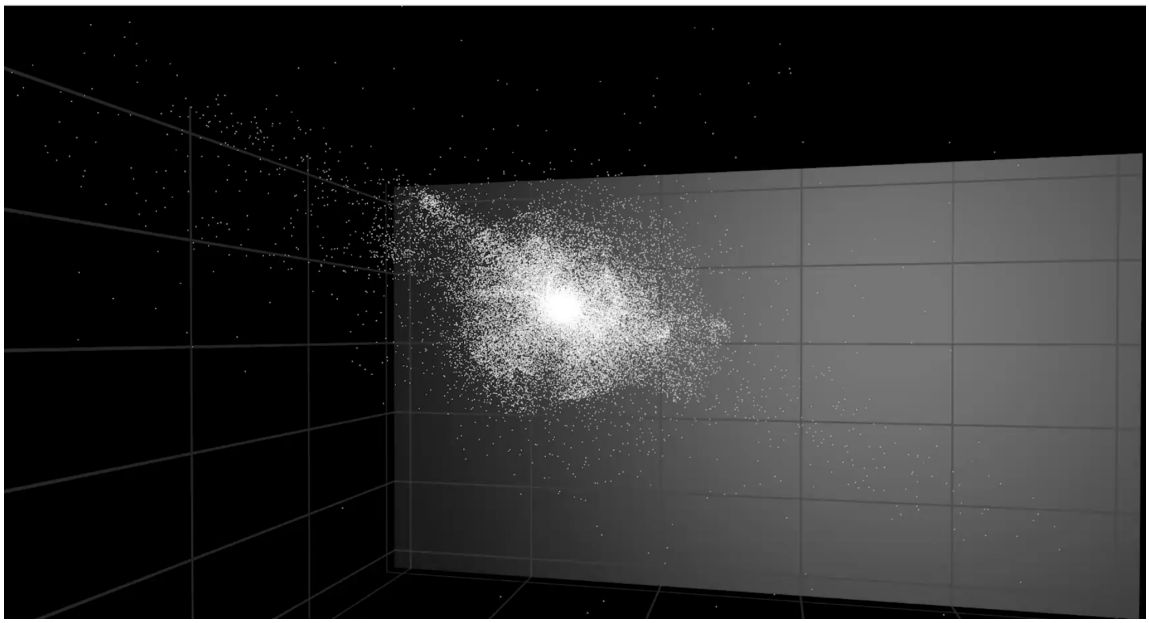
Three Galaxies initial conditions



Three Galaxies Midpoint A



Three Galaxies Midpoint B



Three Galaxies End

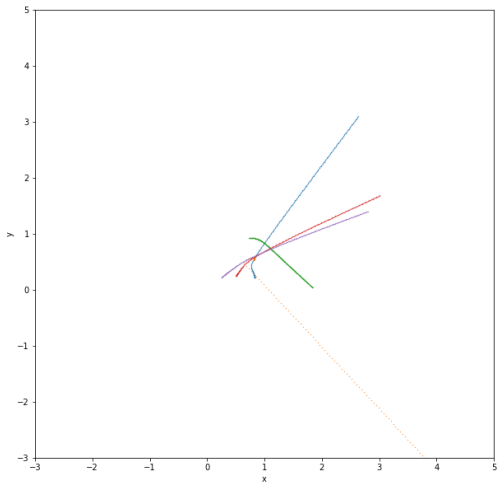
Finally, we stressed the simulator by simulating 60,000 bodies using the leapfrog method of integration. It took approximately one hour for our python implementation of the 3d Barnes Hut algorithm to complete 1000 iterations generating ~6GB of data. Once the data was imported into our Qt gui, the simulation was still able to run in real-time if we disabled particle shading.

Discussion

In demonstrating conservation of energy, we noticed that while the leapfrog is able to preserve energy much better than the forward Euler's method, a zoomed-in image of the leapfrog method's total energy has some variation. We believe an explanation for this is that there is still some error in a numerical approximation of a complex dynamical system. Further work may include investigating other integration methods to observe further improvements to the physical accuracy of our simulation.

One issue we encountered when implementing Barnes Hut was bodies experiencing extremely large forces, and then shooting off into the distance. For some time, we were unable to figure out the cause of this, but we eventually discovered that it was due to small distances between bodies. These led to numerically unstable force calculations, where a big force is added, leading to a big velocity, and causing bodies to shoot away in a straight line.

Another issue was the treatment of bodies leaving the defined space. This was undefined behaviour that was leading to errors in our code, since the entire Barnes Hut algorithm starts from the root node, so leaving the root node's domain would require creating a larger root node. The following figure visualizes this issue: a body's trajectory, represented by the dotted yellow line, is ejected from the system, and this leaves the range of the root node.



One way to handle this exception was to "wrap" the body, such that leaving one edge of the root node leads to the body wrapping into the space from the opposite edge. But this was a rather unphysical treatment. Instead, we opted to simply increase the domain space to minimize bodies leaving the domain and including an assertion in the code to ensure every body is within the domain.

Our simulator can display 60,000 bodies in real-time. However, to handle upwards of 100k particles, we would need to further optimize the pipeline. Some extensions could be to aggregate nearby bodies so fewer particles would need to be displayed, as well as removing all lighting effects. The most critical bottleneck in creating a large simulation is generating the data file and loading it into the simulator. One substantial improvement would be to rewrite the Barnes Hut algorithm in c++ and parallelize it. A relatively simple and major speedup would be to write the data file in a binary format, decreasing the time it takes to save and load simulation files.

Conclusion

We demonstrate the physical stability of symplectic integrators, such as the leapfrog method. The leapfrog method leads to more physically planetary orbits as compared to the forward Euler's method, and we show that it conserves total energy of a closed system, even for long periods of time. Additionally, the leapfrog method approximates known analytical solutions well, and is able to stay within a very close range to the ground truth positions of a system.

Additionally, we show the effectiveness of the Barnes-Hut algorithm in improving computational runtimes for our simulations. The Barnes-Hut algorithm has much better time complexity scaling than naively computing the pairwise force interactions between celestial objects. Thus enables us to run large-scale n-body simulations, and for long periods of time, on modern hardware.

Finally, we implemented a 3D galaxy simulator in Qt to visualize our simulations. These simulations demonstrate a variety of galaxy conditions, ranging from a single stable galaxy, to multiple galaxy collisions.

Acknowledgements

We thank the AM205 teaching staff for providing resources and guidance for this project. Thank you for the great semester!

References

[1] https://www.wolframscience.com/reference/notes/972d

[2] Barrow-Green, June (2008), "The Three-Body Problem", in Gowers, Timothy; Barrow-Green, June; Leader, Imre (eds.), The Princeton Companion to Mathematics, Princeton University Press.

[3] Wang, Q. (1991). "The global solution of the n-body problem" in Celestial Mechanics, 73-88.

[4] Ahmad, A., Cohen, L. (1973). "A Numerical Integration Scheme for the N-Body Gravitational Problem". JOURNAL OF COMPUTATIONAL PHYSICS 12.

[5] Boekholt, T., Portegies Zwart, S. On the reliability of N-body simulations. Comput. Astrophys. 2, 2 (2015). https://doi.org/10.1186/s40668-014-0005-3

[6] J. Barnes and P. Hut, "A Hierarchical O(n log n) Force Calculation Algorithm," Nature, v. 324, December 1986.

[7] A.W. Appel, "An Efficient Program for Many-Body Simulations," Siam, J. Sci. Stat. Comput., 6 (1985), 85-103.

[8] Leslie Greengard, "The Numerical Solution of the N-Body Problem, " Computers in Physics, Mar/Apr 1990, pp 142-152.

[9] https://www.qt.io/

[10] http://arborjs.org/docs/barnes-hut

[11] http://physics.ucsc.edu/~peter/242/leapfrog.pdf

Appendix

Our code for the project can be found at https://github.com/lbertge/am205-barnes-hut.