Galactic Dynamics

Steve Rotenberg
CSE291: Physics Simulation
UCSD
Spring 2019

Gravitational Simulations

- Gravity alone can model several interesting systems such as:
  - Planetary systems
  - Asteroid belts
  - Extended systems (Oort clouds, etc.)
  - Ring dynamics
  - Orbital mechanics
  - Simple galaxy simulations
  - Dark matter

Collisions

- If we add collisions between bodies, we can do things like:
  - More accurate/complex ring dynamics
  - Planet formation (mass agglomeration)
  - Interplanetary impacts
- As the gravitational computations already have to loop through all nearby objects and compute the distance, the collision detection phase can be combined with the gravitational calculation
- Bodies could be approximated as spheres for collision physics, or they could be modeled as full geometric rigid bodies

Gas Dynamics

- Many of the larger scale astrophysics phenomena involve gas dynamics
- This includes:
  - Star formation
  - Star death, supernovas
  - Galaxy motion
  - Galaxy formation
- Gas dynamics are typically modeled with uniform grid or particle-based methods such as SPH
- In fact, SPH was originally invented for astrophysics simulations and only later adopted for water modeling

Gravity Trees

Inverse-Square Gravity

- If we are modeling orbital mechanics, planetary systems, or galaxies, we need to consider the full inverse-square law of gravity acting between two bodies

$$\mathbf{f}_{gravity} = G\frac{m_1 m_2}{d^2}\mathbf{e}$$

- Where *G* is the universal *gravitational constant* (2014 version):

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

- *d* is the distance between the two bodies: $d = |\mathbf{r}_1 - \mathbf{r}_2|$
- And **e** is a unit length vector pointing in the direction of gravitational attraction (i.e., towards the other body)

Inverse-Square Gravity

$$\mathbf{f}_{gravity} = G\frac{m_1 m_2}{d^2}\mathbf{e}$$

- We need to consider the gravitational force acting on every *pair* of bodies
- In a system of *n* bodies, this means we need to compute a gravitational force $n(n-1)/2$ times
- In terms of algorithm performance, this implies O(n^2) performance, which is potentially slow for large values of *n*

Gravitational Dynamics

- Celestial simulations typically involve modeling the gravitational interactions between multiple bodies
- If the number of bodies is small, this isn't a big deal
- If the number of bodies is large, the N^2 nature of the problem becomes a bottleneck
- We ran into a similar problem when we considered collisions between large numbers of objects. In some cases, we were able to use grids or spatial hash tables to optimize the collision detection to linear performance
- However, gravity interacts over very large distances, and so can't be optimized in the same way that short-range collision interactions can

Octree Methods

- But there's good news!
- We can use octree-based methods to reduce $O(n^2)$ down to $O(n \log n)$, which is a huge gain in performance
- One catch is that these methods do introduce approximations, however we can control the accuracy of the approximation and can achieve good accuracy at a good speed

Gravity Octree

- The basic method starts by assuming all bodies are 0-dimensional point masses. All bodies simply require a 3D position and a mass
- In each frame of the simulation, we construct an octree around all of the points
- Then, we do a bottom-up traversal of the tree, computing the aggregate mass and center or mass of each node in the tree. When we get to the top of the tree, we have the total mass and a single point representing the center of mass of the entire system
- Then, to compute the gravitational force at any point, we do a top-down traversal of the tree. At each node, we decide if the single point approximation is good enough or if we have to traverse further. We explore a portion of the tree deep enough to produce a good enough approximation of the total gravitational force

Gravity Octree

- Here is a summary:

ComputeGravity() {

1. Build octree
2. Compute aggregate masses
3. Compute forces

}

- Once we have the forces, we can integrate the motion as usual

Build Octree

- We want to build an octree that fits around all of the points, so we start by computing the bounding box of the points
- Computing a bounding box is linear in time and should be a fast operation
- However, if we are simulating millions of points, this will involve looping through a lot of memory, leading to poor cache performance
- To minimize cache misses it is best to perform the box computation in the motion integration stage of the previous frame, when the positions of all particles are being updated (and are already in the cache)
- Once we have a rectangular bounding box, we just turn it into a cube by using the largest of the 3 dimensions

Octree Data Structure

- It’s nice to have a top-level data structure for the octree that stores the bounding box dimensions and has some high-level control functions
- There should be a separate data structure for a node in the tree. The node should store a total mass (float), a center of mass (vec3), and an index to the 8 child nodes
- Individual nodes really should not have to store their box dimensions, as they can be very quickly computed during the traversal process as needed. This saves memory and (more importantly) memory bandwidth
- Also, instead of explicitly storing pointers to all 8 child nodes, one can instead just store an index value for the first child and assume all 8 are stored contiguously in memory. A single 32-bit index value can work for trees with over a billion nodes

Build Octree

- Once we've set up dimensions of the top level cube in the tree, we can start adding points
- We simply loop through the points and insert them one-by-one into the tree
- To insert a point, we add it to the top node
- If the node is an interior node (not a leaf node), then we determine which of the 8 sub-boxes the point is in and pass the insert operation to the appropriate child node
- This process is repeated recursively until we get to a leaf node
- If the leaf node is not full (i.e., has fewer than MaxPointsPerLeaf), the point gets added and the node stores the position and mass of the point
- Otherwise, we need to split the node by creating 8 child nodes
- The points that were already there are moved to the appropriate child
- The new point is then added to the appropriate child leaf node as well
- If this new leaf is full, the split process repeats recursively
- Otherwise, we add it to the leaf and we're done

Octree Construction

- The performance of the point insertion operation is relative to the number of nodes visited, which is going to be proportional to the average depth of the tree
- Insertion of a single point is therefore logarithmic in performance
- Insertion of n points is therefore $n \log n$

Mass Aggregation

- Once we've finished constructing the octree (step 1), we can move onto step 2: mass aggregation
- In this step, we do a bottom-up traversal from the leaf nodes
- At each node we visit, we loop over the 8 children and add up their mass and centers to compute a total mass and center of mass
- These values are then stored in the node
- Eventually, we will get up to the top node in the tree and have a single mass and center of mass for the entire set of points
- Every node stores the mass and center of mass of all of its child nodes. Leaf nodes contain explicit point masses, and interior nodes represent aggregations of multiple point masses
- This operation visits all nodes in the tree. The number of nodes is roughly a constant ($1 + 1/8 + 1/64 + 1/512 + \ldots = 8/7$) times the number of point masses, making this operation linear in overall performance

Computing Forces

- Once we have built the tree and computed the aggregate masses, we can use the tree to approximate the gravitational force anywhere that we want
- To compute the force at a point **p**, we first examine the top level node in the tree
- We want to determine if the aggregate mass is a good enough approximation for estimating the gravity force
- The quality of the approximation is going to improve the further we are from the box (i.e., the better it can be approximated as a single point)
- As a simple metric, we can compute the distance to the center of the box and divide by the size of the box. If this is above some tolerance, then we accept the approximation
- Otherwise, we recursively loop through the 8 children and add up the gravity force from each
- In total, the number of nodes visited will be roughly proportional to log n
- If we use this process to compute the force on all n points, the total performance of this step is n log n

Gravity Octree

- Step 2 is linear in performance and steps 1 and 3 are $n \log n$, making the total performance $n \log n$
- This is very good, as it is really just slightly worse than linear for a large range of n
- This method can be used to compute gravitational interactions between billions of individual bodies
- It requires a single tolerance value in the force computation step. Tuning this value can trade between performance and accuracy

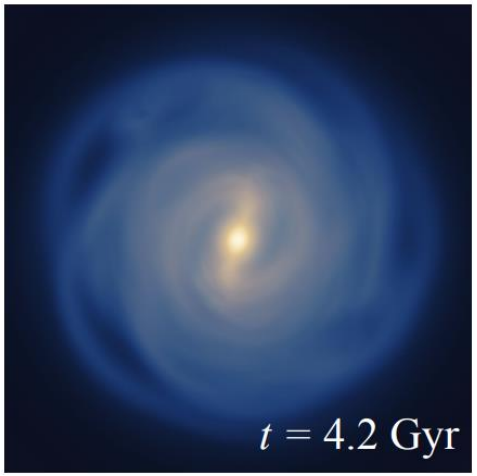
Barnes-Hut Algorithm

- This method based on the Barnes-Hut algorithm
- “A Hierarchical O(N log N) Force-Calculation Algorithm”, Barnes, Hut, 1986
- There have been several enhancements and other algorithms developed since 1986, but this algorithm is still at the heart of many modern galactic simulations

Barnes-Hut Algorithm

- As it is at the core of many astrophysics simulations, the Barnes-Hut algorithm has been optimized for GPUs and supercomputers
- In 2014, 18600 GPUs were used to compute a Milky Way galactic simulations with up to 242 billion particles
- To simulate 6 billion years takes about 4 days, using a minimum time step of 75,000 years (taking about 4.5 seconds per step)

"24.77 Pflops on a Gravitational Tree-Code to Simulate the Milky Way Galaxy with 18600 GPUs", Bedorf, Nitadori, Gaburov, Ishiyama, Fujii, Zwart, 2014



Gas Dynamics

Interstellar Medium

- The *interstellar medium* (ISM) is the matter and radiation that exists in the space between star systems in a galaxy
- This mainly includes:
  - Ionic gas: mainly hydrogen H^+
  - Atomic gas: mainly hydrogen H
  - Molecular gas: mainly hydrogen H_2
  - Dust: solid grains of mainly Fe, Si, C, H_20 ice, and CO_2 ice
- In addition to hydrogen gasses, there is some helium and trace amounts of carbon, oxygen, and nitrogen

Gas Phases

- It is typical to model interstellar gasses as having three distinct phases:
  - Cold dense phase (T<300 K) consisting mainly of atomic and molecular hydrogen
  - Warm phase (T ~10^4 K) consisting of atomic and ionized hydrogen
  - Hot phase (T ~10^6 K) consisting of gas that has been shock heated by supernovas
- Several physical processes are responsible for the kinetic and thermal energy exchanges between these phases

Star Formation

- In dense regions of the ISM, the self-attraction from gravity causes areas to collapse
- The collapse will ultimately lead to protostar with a pressure-supported core that continues to collect mass from the nearby cloud
- This will ultimately ignite and begin nuclear fusion
- The star will settle into equilibrium between the continuous nuclear explosion and the gravitational contraction

Stellar Feedback

- Stars also return matter and thermal energy back to the ISM through the processes of *stellar feedback*, mainly through stellar winds and supernovas
- Stars heat the ISM through stellar feedback and the ISM cools mainly through radiative cooling
- Areas with newly forming stars cause strong pressures in the ISM, leading to expanding bubbles as the new stars blow the nearby gas out

Galactic Simulation

Galaxy Simulation

- Many galactic simulations are focused on exploring the processes of galaxy formation and evolution
- Galaxies start as large collapsing clouds of gas
- Stars form as local pockets of the gas collapse and gradually, much of the gasses are consumed by star formation
- From a simulation point of view, this implies a large scale compressible fluid dynamics simulation with thermal processes and gravity

Gas Modeling

- The gas dynamics are typically modeled either with a uniform grid or with a particle method like SPH
- Unlike the fluids we've looked at so far, these gasses are compressible and their densities will vary over time and space
- We also can account for the gravitational forces in the gas clouds using the Barnes-Hut algorithm or other methods
- If we are using SPH, we will probably require that a single particle represent a mass of 1000s of suns or more

Sink Particles

- It is also common to use *sink particles* to represent point masses like stars or black holes
- Sink particles can be used in both SPH and grid based simulations
- Sink particles can be created automatically in areas where the gasses collapse enough to trigger star formation
- The sink particles will interact with overlapping gasses through stellar feedback and mass accretion
- They will also interact gravitationally with everything else and will exchange mass in both directions with the gasses

Dark Matter

- Roughly 5% of the total mass-energy of the universe is ‘ordinary’ matter and energy
- Roughly 27% is *dark matter* which interacts with ordinary matter (and itself) through gravity only
- The remaining 68% is hypothetical *dark energy* which is responsible for the current acceleration of the expansion of the universe
- For galactic modeling, it is very important to include the effects of dark matter, as it is responsible for roughly 85% of the mass of a galaxy

Dark Matter Modeling

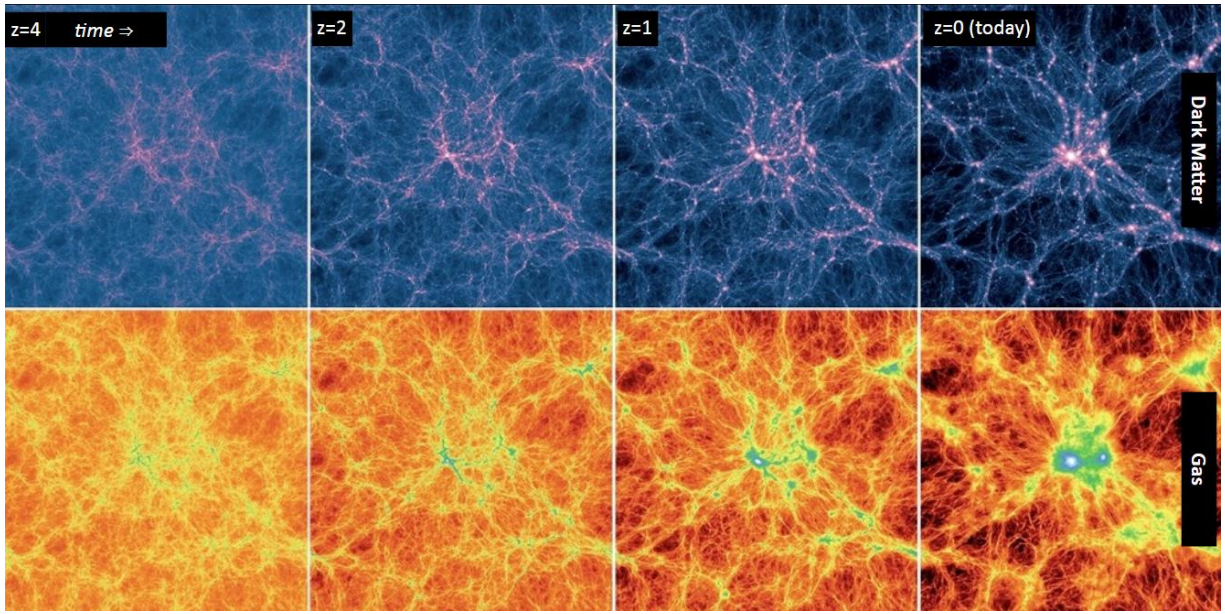
- Dark matter is typically modeled in one of two ways:
  - As additional particles that only interact through gravity
  - As a static field that mimics the approximate distribution of dark matter in a galaxy
- The first method is used in simulations of galaxy formation and evolution, whereas the second method is limited to simpler simulations of later stage galaxies

Big Bang Modeling

- There have been huge supercomputer simulations that model the Big Bang and formation of the universe using billions of particles
- They model the universe as a box that wraps in each dimension, tiling infinitely, and starting from an initial uniform distribution of hot hydrogen gas particles
- Typical cube sizes are 50-300 Mpc (1 megaparsec = 3.262×10^6 light years)
- The uniform cosmological expansion in all directions is modeled by changing the distance scale of the simulation appropriately over time
- As the simulation progresses, gravity will cause some regions to collapse down, leading to galaxy formation and ultimately leading to a distribution of galactic clusters throughout the universe
- The simulations can be run from the big bang to the present day (13.8 billions years) and ultimately lead to galaxies that resemble real examples
- In this way, different cosmology models can be simulated and the results compared to real observations

Illustris Project

- Width of box: 302.6 Mpc = 987M light years
- Smallest matter particle: 110,000 solar masses
- Smallest dark matter particle: 590,000



FIRE Project

- The FIRE (Feedback in Realistic Environments) Project modeled the expansion of the universe leading to the formation of individual galaxies to the present day
- These galaxies can then be rendered with a physically based renderer (Latte) to produce images that are remarkably similar to observed galaxies

FIRE-2

- “FIRE-2 Simulations: Physics versus Numerics in Galaxy Formation”, Hopkins, et. al., 2018