Sean Springer Fall 2025 ss28@pdx.edu

Black Hole Universe

An N-Body Simulator of A Peculiar Universe

https://github.com/seandspringer/Black_Hole_Universe

Black Hole Universe will be a N-body gravitational simulator for a special spherical universe filled with black holes (and a few other objects)! The simulation will be built using Bevy (https://bevy.org/) – a free and open-source game engine built in Rust and will use primitive shapes and meshes to represent cosmic objects. Can you build structure out of chaos?

Introduction

Black Hole Universe is intended to be both a simulation and a game: natural physical laws of the cosmos will be simulated while a game-like goal to create a universe that could contain life will be emphasized. Clocks will be used to measure the length of time that the simulation has been running as well as the length of time that a hospitable planet exists within the universe; both of which can be used for score keeping. An ideal outcome of the simulation would be to find a starting configuration which produces a stable (or semi-stable) orbit for one or more planets around a black hole. This would create a long-lived planetary system – long enough for life to evolve. Note that, because the universe is spherical and thus wraps around itself, any non-orbital systems are expected to eventually collide and extinguish.

The N-Body Simulation

Black Hole Universe is a classic N-body simulation because both the black holes and planets (and any other objects that may get added to the system) are modeled as particles with a mass, velocity, radius, and location. Massive objects exert a force on each other known as the universal gravitational force[1]

$$F = G\frac{m_1 m_2}{r^2}, \tag{1}$$

Where F is the force, G is Newton's gravitational constant, m_1 and m_2 are the masses of the two objects and r is the distance between the two massive objects. Also, according to Newton's second law of motion, the change of motion of an object is proportional to the force acting upon that object:[2]

$$F = m\frac{dv}{dt} = ma. \tag{2}$$

Here, v is the velocity of the object, t is time, and $dv/dt \equiv a$, the acceleration of the object. In this context, it is often helpful to consider acceleration as a change in the velocity over time. To determine the acceleration of an object (the object's velocity change) due to the gravitational influence of other massive objects in the world, we can substitute equation 2 into equation 1 and solve for the acceleration to derive the gravitational field experienced by an object:

$$\vec{a} = G\frac{m}{r^2} \tag{3}$$

Here, m refers to the mass of the other object and a is a vector with a magnitude and direction. For multi-body systems, the a vector can be vector-summed over all gravitational masses, m, with distance r to produce a resultant vector describing the total acceleration felt by the object at each time slice.

Utilizing Newton's equations of motion,[3]

$$\vec{v} = \overrightarrow{v_0} + \vec{a}t \tag{4}$$

$$x = x_0 + \vec{v}t - \frac{1}{2}\vec{a}t^2 \tag{5}$$

The velocity vector and 2-dimensional position, x, can be updated at each rendering frame for each object after defining the initial conditions.

Project Vision

Black Hole Universe will contain two main types of objects: black holes and planets. The numbers of each object will be determined by user-input controls as well as initial velocities and mass distributions. The latter two cases will be governed by Gaussian statistics; however, the mean and standard deviation of the distribution will be user-controlled. Objects will initially spawn randomly throughout the universe.

The Bevy game engine will be used to display the model using a square map with programmed wrap-arounds (to enforce the spherical nature of the universe). Objects (both black holes and planets) will be modeled as circles using Bevy's built-in primitives and will be distinguished based upon mesh color[7]. The generalized relationships[4,5]

$$r_{black\ hole} = 3m_{black\ hole} \tag{6}$$

$$r_{planet} = m_{planet}^{0.27} \tag{7}$$

Will be used to dictate the physical sizes of the objects. These equations are given in literature as approximately representative of real-world relationships.

Collision detection will be built into the Bevy-rendering pipeline whereby the following interaction scheme will be enforced:

1. Collision of black hole + black hole makes a black hole with cumulative mass and radius defined by equation 6
2. Collision of black hole + planet creates a black hole with cumulative mass and radius defined by equation 6
3. Collision of planet + planet will cause the planets to fracture in-half, splitting the mass of each planet by 2, effectively creating 4 planets with sizes determined by equation 7.
4. A mass / radius cutoff will be experimentally determined in which a planet ceases to be habitable and is deemed "dust" and removed from the game

The simulation will track the duration that hospital planets existed in the universe and the duration that multiple black holes existed. It is expected that, at some point, all black holes will coalesce into a single super massive black hole and that will be the natural end of the simulation. In this manner, a game can be made to discover the longest-lived planetary system and non-singular black hole states, implying that certain initial conditions are more or less optimal to the formation of life!

Additional (Time Permitting) Features

With available time, the following list of topics would be interesting to incorporate into the system: A "User" planet that can be set a particular location and given an explicit velocity vector, dark matter incorporation, black hole evaporation rates, big bang simulation beginnings, spontaneous (theoretical) white hole formations which add planets into the universe, and spontaneous worm hole formations which will instantly transport objects across the map.

Concerns

At every frame update, the acceleration due to the gravity of other objects on the map will need to be calculated for every object. Despite the $1/r^2$ distance dependence of gravity according to Newton's equation of gravity, the most realistic simulation will account for the effect that all gravity sources on the map, regardless of distance. If the simulation has n total gravitational objects, then for each of the n objects, $n-1$ calculations must be made to produce that object's instantaneous acceleration vector. This requires $n*(n-1) = n^2 - n$ evaluations per frame update which is $O(n^2)$. For large n, this could be a limiting step which causes the frame rate to noticeably slow.

To address this concern, Rust threads will be deployed to parallelize these calculations across CPUs. The acceleration vector for each nth object can be calculated as the sum of all gravitational influences on individual threads and then after all threads join, the velocity and position of each object will be updated.

Because there is usually overhead in spinning up threads, the schema outlined above may also fail to be sufficiently efficient. In this event, more complex data structures like Barnes Hut trees or Fast Multipole Method[6] may need to be investigated. Distance cutoffs may also need to be implemented.

Sources

1. https://en.wikipedia.org/wiki/Newton%27s_law_of_universal_gravitation
2. https://en.wikipedia.org/wiki/Newton%27s_laws_of_motion
3. https://en.wikipedia.org/wiki/Equations_of_motion
4. https://blackholes.stardate.org/resources/article-structure-of-a-black-hole.html
5. https://www.aanda.org/articles/aa/full_html/2024/06/aa48690-23/aa48690-23.html#F1
6. https://patterns.eecs.berkeley.edu/?page_id=193
7. https://bevy.org/examples/2d-rendering/2d-shapes/