OpenGL Solar System Simulation Report

1. Introduction

This project focuses on simulating the formation and evolution of a solar system using OpenGL for visualization. The simulation begins with 100,000 objects distributed in space, each initialized with a velocity. These objects interact under gravitational forces, orbiting a central sun-like mass. Over time, collisions occur, leading to the merging of objects based on their mass and radius. This process gradually results in the emergence of larger celestial bodies, ultimately forming a solar system.

Initially, the project aimed to replicate our real solar system. However, after discussions with the professor, the approach was modified due to the challenges of accurately scaling physical laws and distances. Instead, a procedural evolution model was implemented, allowing a self-organizing system to emerge from simple initial conditions. The project combines real-time graphics rendering with fundamental astrophysical principles, offering insights into planetary formation dynamics.

This simulation not only serves as a visual representation of celestial mechanics but also explores computational challenges in handling large-scale object interactions. Future improvements may include performance optimizations on GPU, enhanced collision detection, and more realistic celestial integration.

2. Background & Theory

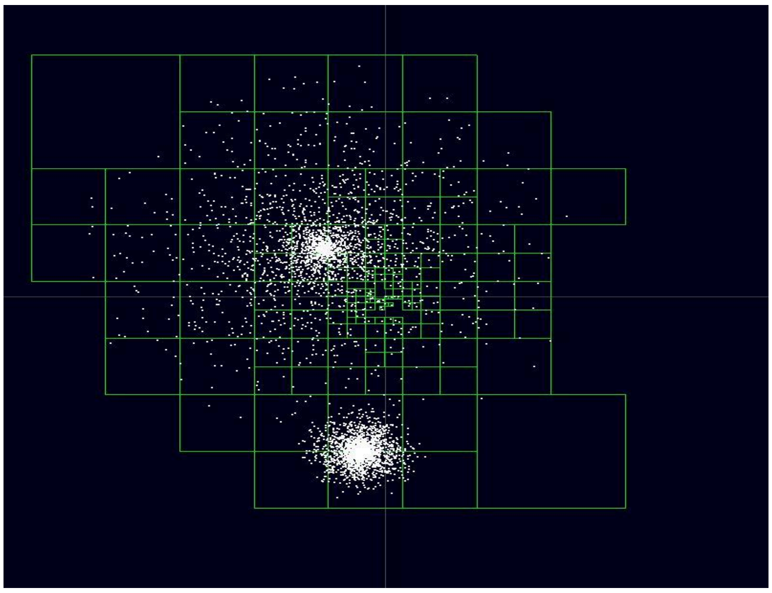
An N-body simulation models the motion of multiple interacting bodies under gravitational forces. Each object exerts a force on every other object, following Newton’s Law of Universal Gravitation:

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

where F is the gravitational force, G is the gravitational constant, m1 and m2 are the masses of two objects, and r is the distance between them. The system’s evolution is determined by solving these forces iteratively over time, updating velocities and positions accordingly.

However, this approach would be too naive when dealing with a large number of objects, as the complexity would reach O(N^2). To prevent this, we used the Barnes-Hut approximation for calculating interactions between objects.

Barnes-Hut utilizes the concept of quadtrees and relies on a center of mass, which is a representation of all the bodies present in the respective quadrant. This results in a reduction of time complexity from O(N^2) to O(NlogN) or even O(N).



Reference: Physics Stack Exchange - Barnes-Hut Algorithm

To handle collisions (merging of objects), we utilize spatial partitioning and Union-Find data structures, which have an average time complexity of O(N). We use a grid to cluster the objects together, and the grid size is adaptive.

3. Implementation Details

OpenGL for Visualization

- Utilized **instanced rendering** for efficient display of 100,000 objects.
- Window size: **1024 × 768**.

Initialization of Objects

- **Generated 100,000 particles** in a protoplanetary disk configuration.
- **Positions:**
    - X-coordinate: Bimodal distribution with means at ±15.0 AU, standard deviation 10.0 AU.
    - Y-coordinate: Normal distribution with mean 0.0 AU, standard deviation 10.0 AU.
- **Velocities:**
    - Calculated using the **circular orbital velocity formula** with a random multiplier (0.95 to 1.05) for slight perturbations.

Mass and Size

- **Radius:** Uniformly distributed between 0.005 and 0.02 AU.
- **Sun Mass:** Assumed to have **1000 Earth masses** (not to scale).
- **Density:** Uniformly distributed between 0.8 and 2.5 (arbitrary units using Earth's mass as a reference).
- **Mass Calculation:** Based on radius and density, creating a range of object sizes.

Evolution Process

- **Orbital motion and gravitational calculations.**
- **Collision detection and object merging logic.**

4. Results & Observations

In the simulation, N objects interact with one another. When two objects collide, they merge into a single entity and continue along their trajectory.

At the beginning, when there are only a few objects in the system, the sun remains stationary while the objects revolve around it. However, as the simulation progresses, objects begin to merge, forming larger bodies. These massive objects exert gravitational influence on the sun, causing it to shift its position within the system.

When the collision (merging) logic is disabled, a ring of objects forms around the sun at a certain distance. Additionally, in some cases, the simulation exhibits the formation of moons orbiting larger bodies.

5. Challenges & Improvements

Simulating a large number of interacting bodies presents several computational challenges:

- **Computational Complexity:** A naive approach requires computing forces between every pair of objects, leading to an O(N2)O(N^2)O(N2) complexity, which is inefficient for large NNN. To address this, we implemented the Barnes-Hut algorithm.
- **Precision and Stability:** Small numerical errors in integration accumulate over time, impacting long-term accuracy.
- **Collision Handling:** Efficient algorithms are necessary to ensure objects merge or interact correctly upon collision.
- **Rendering and Visualization:** Real-time visualization of 100,000 objects requires optimized graphics processing.

We faced challenges in integrating **GPU acceleration** due to platform dependencies, as we were using macOS. The simulation runs smoothly with **10,000 objects**, but performance degrades as the number of objects increases. With **GPU integration**, the simulation could achieve **30 FPS while handling 100,000 objects**.

6. Conclusion

This project successfully demonstrates the formation and evolution of a solar system using an **N-body simulation with OpenGL visualization**. By implementing **Barnes-Hut approximation** and **optimized rendering techniques**, we achieved a more efficient and visually appealing simulation.

Potential Future Work:

- Enhancing **realism** by incorporating more accurate astrophysical models.
- Implementing **procedural generation** for diverse planetary system formations.
- Integrating **GPU acceleration** to handle an even larger number of objects in real-time.