Galaxy Engine

> Recommended Team Size: 2-3 people
>
> Project Lead: Site Fan (**fans2021@mail.sustech.edu.cn**)

Project Overview

In astronomical simulations and research, high-performance computing is often used for astrophysical simulations and celestial data processing. Simulating celestial systems involves numerous interactions between astronomical objects and complex physical calculations, with data ranging from a few to millions of bodies. The field of astrophysical simulation has spawned many algorithms, from $O(N^2)$ exact physical simulations to $O(N)$ methods, such as the Fast Multipole Method (FMM) [1], which are widely applied in astronomical research.

In this project, you are tasked with developing a high-performance C/C++ library for multi-body simulation calculations to handle astrophysical simulations of varying scales. The library should be intuitive and elegant, showcasing examples of its usage. Through this project, you will gain experience in writing open-source libraries, using CPU/GPU parallel computing, simple GUI design, and understanding basic astronomy and simple simulation algorithms.

Project Requirements and Details

Galaxy Engine Library (40 points)

In this section, you are required to complete the core functionalities of the Galaxy Engine by implementing a flexible, high-performance C/C++ library for simulating multi-body interactions in celestial systems. The key requirements are as follows:

1. **Data Structures and Algorithms (10 points)**
    - Choose appropriate data structures based on different scenarios to store data during celestial simulations.
        - Use object-oriented programming to manage celestial data, ensuring class member access control and programming standards.
            - Possible implementations include structures like CelestialBody, Trajectory, Galaxy, etc.
        - astronomical object data should include, but not be limited to, time, spatial coordinates, velocity, acceleration, mass, radius, and others (additional parameters can be added depending on the desired computational accuracy).
        - For basic requirements, the celestial coordinates should be in two dimensions with a top-down view for observation.
2. **Gravity Calculations and Astronomical Object State Updates (10 points)**
    - Provide at least two simulation algorithm interfaces for different scales or scenarios:

    - Small-scale computation: classical Newtonian mechanics, $O(N^2)$ gravitational calculations.
    - Large-scale computation: implement high-performance simulation algorithms [1] [2] for larger datasets.
  - Required update interfaces: astronomical objects, trajectories, camera positions for rendering, etc.
3. **Parallel Computing Support (15 points)**
   - Implement parallel computing to enhance computation speed. If feasible, try using multi-machine, multi-CPU, or multi-GPU scheduling.
   - Parallelize interactions between astronomical objects to improve computational efficiency during large-scale simulations or renderings.
     - Use tools such as OpenMPI [3], CUDA [4], SIMD [5], etc.
   - Perform vertical comparisons of the library's performance under different data volumes, as well as horizontal comparisons of the library's performance with and without parallel computing or CPU/GPU acceleration, comparing it with existing astrophysical simulation frameworks for efficiency.
   - There are many open-source celestial simulation libraries on GitHub; compare the bottlenecks in these libraries and the Galaxy Engine and propose possible improvements with estimated performance gains.
4. **Modular Design, Interfaces, and Usability (5 points)**
   - Design clear APIs that allow easy configuration of celestial parameters, time steps, simulation steps, etc.
   - Modularize the library's components, separating different functionalities to simplify its use and improve the extensibility of data structures/algorithms.
   - Ensure portability and ease of use, allowing the library to

     be used as a static or dynamic library, ideally with plug-and-play capabilities.
     - Consider using tools like CMake, Docker [6], Kubernetes [7].

Use Case Demonstration (40 points)

Using the C/C++ library developed by your team, write one or more example use cases that elegantly showcase how to use the library.

1. **Basic-Scale Celestial Simulation (20 points)**
   - Implement a small system with a few astronomical objects (e.g., the Three-Body System, the Solar System) to demonstrate the interactions between multiple bodies.
   - Support input of celestial data from one or more files, log customizable events like collisions or eclipses and allow export of celestial system data to files (5 points).
   - Support visualization effects (15 points):
     - The input and output data should interface with the computation library.

        - Provide the following features in the GUI:
            - Movable camera to change the observation view.
            - Add/remove/modify/view information of astronomical objects.
            - Time controls: forward/backward simulation, adjustment of time steps, jumping to specific points in time, etc.
        - There's no strict constraint on GUI design, try OpenGL[8], Web front-end, or even ASCII rendering[9]. Creativity is all you need!

2. **Large-Scale Simulation Display (20 points)**
    - Implement several large-scale celestial system simulations to demonstrate the efficiency and accuracy of Galaxy Engine under high-performance computing.
        - Choose an appropriate large-scale data set based on your algorithm implementation. Larger is only sometimes better; balance precision and efficiency.
    - Support input from one or more files, log customizable celestial events and allow the export of celestial system data (5 points).
    - Support visualization effects (15 points):
        - The input and output data should interface with the computation library.
        - Provide the following features in the GUI:
            - Movable camera for observation.
            - Import/export astronomical object data.
            - Time controls: forward/backward simulation, adjust time steps, jump to specific points, etc.
        - The GUI format is flexible.

Bonus Section (Up to 40 points)

This section rewards innovative designs and advanced functionalities. Here are some possible bonus points:

1. High-Performance Computing (10 points each)
    - Use multiple machines for distributed task scheduling and result integration or simulate with Docker containers.
    - Provide performance stress-testing tools and test sets for the library, focusing on versatility and ideally making it easy to test other computational libraries via clear interfaces.
2. Real Celestial Data Import and Export (10 points)
    - Support automatic import and update of real celestial data and simulate celestial systems based on this data.
    - Use real data for simulations and showcase how real celestial systems move.
3. Advanced Visualization and Interaction (10 points each)

    - Add gravitational field and force visualizations to enhance understanding of the celestial system's structure and gravity distribution.
    - Support 3D celestial system rendering, with an additional Z-axis for the camera, allowing users to observe star maps, eclipses, and more.
    - Add non-solid textures to astronomical objects and support video export (even short, a few seconds will show dynamic changes).

4. Creative Ideas (Graded by efforts, up to 20 points)
    - Provide detailed explanations of these features in your report and clearly demonstrate their use cases. Do not just paste code.
    - Support 4D celestial system rendering[10]. The camera may need more directional controls; your imagination will be key!
    - Enhance physical simulations:
        - Support body collisions and mergers, where merged bodies inherit mass and momentum, forming new astronomical objects.
        - Include more astronomical object types and behaviors, such as black hole formation and comet melting.
        - Simulate phenomena that Newtonian mechanics cannot accurately simulate or support non-standard celestial shapes.
    - Use ray tracing to enhance rendering quality and effects. For example, simulate solar and lunar eclipses, Venus transits, and more!
    - Excellent coding standards, progress management, and teamwork at a commercial level!

Project Submission

Code Submission

It is recommended that the project be managed using Git. If the project is open-source, provide the Git link in the report. Also, consider packaging the project as a Docker image and uploading it to Docker Hub. For closed-source projects, submit the entire project as a .zip file.

Please ensure that a README file in the root directory includes detailed instructions for setting up the environment (library and software dependencies etc.) to ensure the project can be reproduced in the given environment.

Report Submission

This project will be graded based on an online presentation. Prepare slides that clearly and concisely highlights all the scoring points (including performance comparison reports), showcasing any extra highlights.

Along with the code, submit a PDF report that differs from the README and slides. The report should cover the project's structure, algorithm design, code implementation, performance analysis, testing results, and related work. This report is an auxiliary document and will not directly impact the grade but will serve as a reference for showcasing your development and research skills in future applications or interviews. There are no specific length or format requirements, but ensure it meets your needs.

---

1. **Fast multipole method - Wikipedia** ↩ ↩
2. **Barnes-Hut Simulation** ↩
3. **MPI Tutorial** ↩
4. **CUDA C++ Programming guide** ↩
5. **SIMD Basic Tutorial** ↩
6. **Docker Tutorial** ↩
7. **k8s Tutorial** ↩
8. **OpenGL Tutorial** ↩
9. **GitHub/ASCII-galaxy-simulation** ↩
10. **GitHub/4D Camera** ↩