Solar System Sim

A VR solar system simulation.

Jacob Burtch, Max Mcnally, Chelaka Fernando, Marizza Ranasinghe, Braeden Hong
Dr. Miguel Garcia-Ruiz
Algoma University
Faculty of Computer Science and Technology
COSC4426
Sault Ste. Marie, Ontario, Canada
November 5, 2024
https://github.com/Entropite/solar-system-sim

1 The Simulation

This project features a VR simulation of the solar system. This simulation includes all of the main components of the solar system. Namely, the sun and the planets. The simulation also features the Earth's moon, but not the moons of all the other planets. Other objects such as asteroids are simulated as well. The application uses Kepler's laws of planetary motion to determine the orbits of celestial objects. Specifically, the simulation is built with Kepler's first and third laws in mind. The first states that every planet's orbit takes the shape of an ellipse with the sun at one of the focus points. This eccentricity inherent to orbits is very subtle and was virtually unknown until delicately calibrated scientific instruments were developed in the late 16th century. The third law describes the relationship between the time elapsed to complete an orbit and the mean distance from the sun. This relationship is illustrated well in the simulation as planets in the outer solar system orbit far slower than planets in the inner solar system. Figure 1 shows the first test of the VR system in the program showing the two controller rays that show where the VR controllers are in the world.

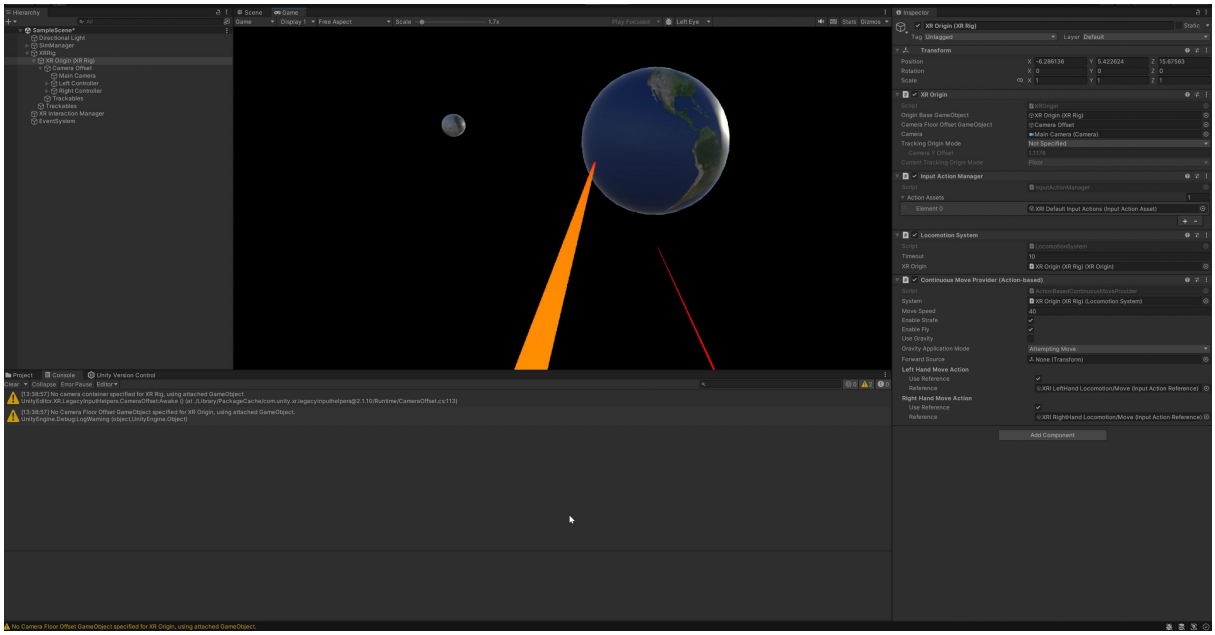

Figure 1: Initial VR test of the simulation.

2 Instructions

The camera can be moved by using the W, A, S, and D keys on the keyboard, and can be panned to look around by moving the mouse (see figure 2 and figure 3). In VR (on an Oculus touch controller), the left analog stick is used for continuous movement. Camera panning is of course done by looking around while wearing the HMD. Keyboard and mouse input will be disabled when playing the simulation in VR.

3 Considerations

A time scale slider was added to increase/decrease the speed of the simulation, along with a label showing what the time and date is in the simulated world. This allows the user to see the position and rotation of the planets as they would be in the real world at that date (figure 4).

To reduce the scope of this project, not all moons of every planet will be simulated. Some planets, for example Saturn, have more moons than would be feasible to show in this simulation. For planets with many moons, only a select few were simulated (see figure 5).

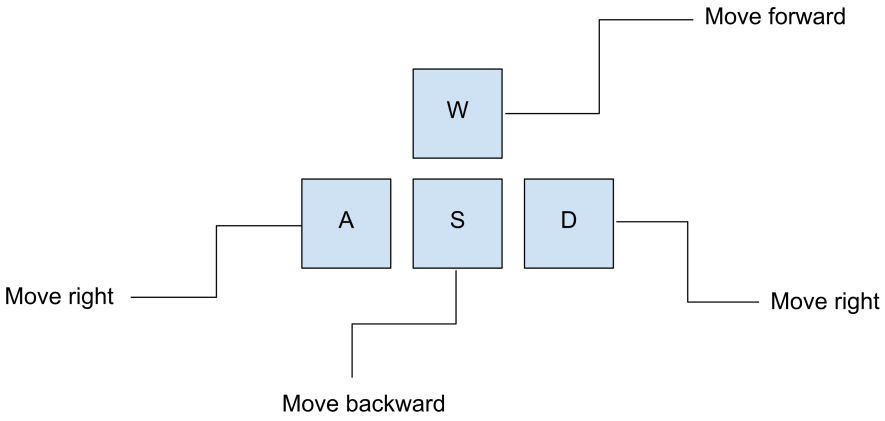

Figure 2: Keyboard controls.

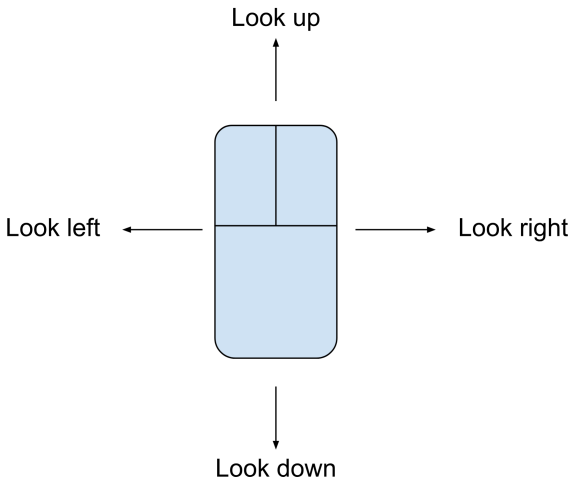

Figure 3: Mouse controls.

To enhance the visual appeal of the simulation, a random distribution of asteroids will be spawned to simulate the asteroid belt. These asteroids have no consequences on the rest of the simulation however as they will not interact with any of the planets in any meaningful way.

4 Challenges

The built in Unity physics engine was not very well suited for the simulation of planetary motion, and thus was replaced with a custom solution. The Unity physics engine is a good fit for games that usually take place in some world with Earth-like gravity, but it does not offer the appropriate functionality for simulating planetary motion. Since the motion of rigidbodies is dependent upon a built-in *timeScale* variable, if rigidbodies were relied upon, the simulation's speed would be bounded by the *timeScale* variable. Since this variable has a maximum value of 100, the simulation would progress slowly. Freed of this limitation, the simulation can simulate years in mere seconds.

This achievement also necessitated replacing Newton's law of universal gravitation with Kepler's laws of planetary motion. The former process was rather inefficient and involved making discrete updates to the planet's position every update event. This greatly reduced the maximum speed the simulation could run at and failed to appropriately model the motion of planetary moons. By adopting Kepler's laws, the position

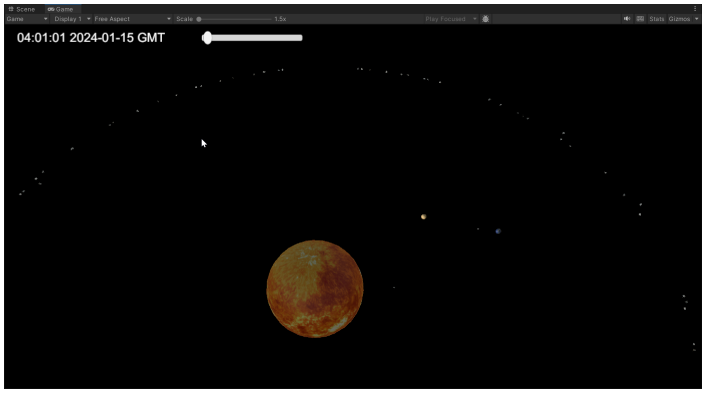

Figure 4: The simulation running showing the solar system on January 15, 2024.

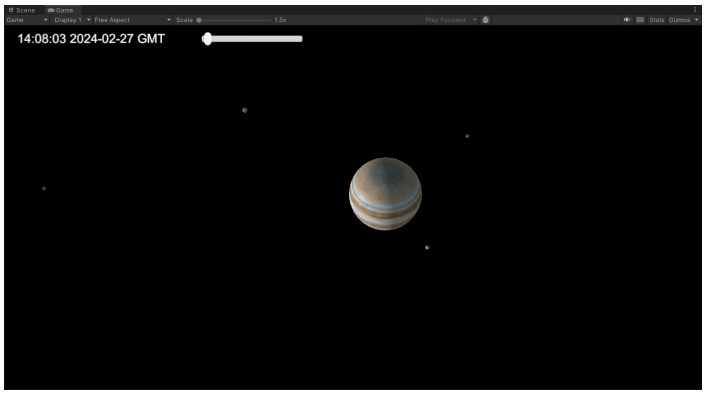

Figure 5: Some of the moons of Jupiter.

of celestial bodies depend not on discrete updates, but only the amount of time elapsed since the simulation began. This makes calculating planetary positions a very fast operation.

For project collaboration, Unity Cloud was considered, but it is only free for the first 3 seats. For this reason, we chose to use git with GitHub for source control instead.

Another point of issue during development was converting an existing Unity project to one that supports XR. While it should be very simple in theory, the Unity documentation for the version of Unity used in this project was slightly incorrect. This lead to an extended amount of time being spent trying to debug the wrong problem. The Unity documentation points to the documentation for version 3.0.3 of the "XR Interaction Toolkit". However, when attempting to use the Unity package manager, it claims that the latest version is 2.6.3. These two versions are completely incompatible and have different APIs. For some reason, the VR sample project uses version 3.0.3 (on the same version of Unity!), but the Unity package manager will not install it in an existing project.