Goals

To create a 3D simulation of the solar system that can be easily manipulated and viewed from many different angles, with a balance of aesthetics and realisticness.

Main Algorithms

Math/Physics

The Orbits are calculated through a complicated procedure using Keplerian Orbital Elements. These elements are related to Euler Angles and describe the orbit and current location of each planet using 6 variables (+1 for the current "epoch"), by calculating the *mean anomaly (M)* in order to solve Kepler's Equation:

$$M = E - e^* \sin E$$

After calculating the *eccentric anomaly (E)* from this equation, there are three coordinate conversions to bring the coordinates to Equatorial Coordinates, a cartesian system that we use to translate the planets relative to the Sun (origin). A full description of the process we used is available here: https://ssd.jpl.nasa.gov/txt/aprx_pos_planets.pdf. We also implemented a Calendar/Date system to iterate through time and display the solar system at various points in time.

Why Keplerian Orbital Elements?

Starting point: Even with a gravity- based simulation, some precomputed parameters are required to obtain accurate starting positions of the planets.
Flexibility: Orbital elements are easier to move through time, since they allow for us to create a function that takes any arbitrary time and output a position
Non-trivial: Orbital elements involve more complicated math that is more relevant to some of the lectures in class (Euler angles, coordinate transformations) than a gravity based simulation, so we believed this method would be a better use for this class.

Why not Pluto?

The elephant in the room. Pluto unfortunately is now considered a dwarf planet by astronomers. Logistically, it is also difficult to include meaningfully because it is a very small planet that is very far from the Sun, meaning the user would not be able to see the planet very clearly anyhow.

Graphics:

The planets are all spheres texture mapped based on a UV projection in Tesselation shaders. The orbital trails of the planets are small objects that are instanced following the positions of each planet to make the orbit of each planet more apparent. They act as particles and "die" after their life span passes. The background is a texture of the Milky Way. It is not cube mapped, but it is just a static picture of the Milky Way.

Implementation Engineering Process

Our main class representing the Solar System is SolarSystem, in “solar_system.h/cc”. The class contains 8 PlanetaryObject’s that represent each planet, plus one additional that represents the Sun. The class provides an interface for main.cc to generate the initial solar system, as well as update the state of the solar system (by changing the date and updating the positions based on that date). The class does very little math but acts as a central system to process all useful changes to the SolarSystem through the classes that do implement the math. The class also contains code to read configuration data about the planets from our JSON files. Each configuration file contains the following data:

- Planet name, size, mass, texture image filepath, orbital elements data
- This data is parsed and added to the SolarSystem class through a function called at the start of the program.

The file containing most math and time code is “physics.h.” The Date class contains our representation of time. We implemented our own Date class because C++ does not have a proper Date class of its own, and third party alternatives were not satisfactory. It also allowed us to add convenient helper functions for our own purposes. The Date class is fairly simple, although it accounts for leap years, it does not include hour/minute/second increments (although partial days can be used, since the “day” variable is a double) since they were not necessary.

The OrbitalElements class contains the bulk of the actual orbital math needed to calculate the position of each planet at each time position. The class contains the necessary orbital elements for each planet in two categories: “start” and “diff.” The “start” elements are the elements from the J2000 (January 1st, 2000) time frame. The “diff” elements are how much each element is translated per century past (or before) J2000. Jupiter through Neptune contain 4 additional elements to calculate their orbits. The class also contains the 6 steps needed to turn the elements and a time variable into the position of the planet in equatorial coordinates, in 6 functions. These functions are all put together in a function in SolarSystem and called every frame to calculate the new coordinates.

The rendering code is basically contained in “main.cc”. The code pertaining to the particles is contained in “particle.h” and “particle.cc”. Particles are left at the planet’s last position in each pass. A particle trail is only updated to make new particles and decrease the life of old particles when the last position is far enough from the new position.

Bugs/Limitations

The textures of the planets currently move to face the user. We also do not have the background cubemapped, but it is just a static image, so the stars are static when looking around. In addition, we did not have time to get Saturn’s rings to appear. Finally, the orbital trails do not extend very far for planets that are further away because there are too many particles to render for the planets that are further away, and it would reduce the FPS too much. In addition, our program runs at approximately 20 FPS when all the particles are rendering, so it is not very fast right now.