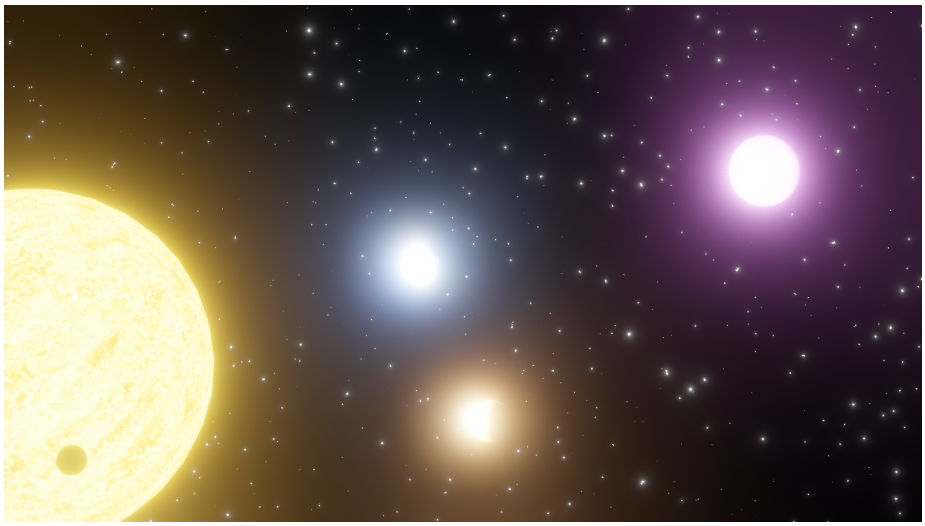

Solar System Simulator

Ludwig Boge - ludbo064
Nikita Sidarovich - niksi312
Jonatan Ebenholm - joneb274
Berkay Orhan - beror658

May 28, 2026

LINKÖPING UNIVERSITY

Abstract

This report goes trough the process of creating a solar system simulator in Blender using Python script, as well as creating an UI which helps the user of the add-on customize their experience. The report first introduces the subject and gives a background to the subject matter. Afterwards, the physics involved in simulating a solar system is discussed, as well as the physical model that was created. Thereafter Euler's approximation method is explained, how it is implemented with the physical model to simulate the system is then discussed. Next the graphical implementation is explained in detail, which delves into the UI of the Blender add-on, the animation method, the materials and textures of the planets that are implemented. The results are then shown and explained, after which there is a guide explaining how to use the add-on in Blender. Afterwards improvements and additions that are possible to the add-on are discussed. Lastly there is a short and concise conclusion to the project.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

People of all age have always been fascinated by space and planets, there has always been a huge enjoyment in looking up at the sky and admiring the stars at night. That fascination has led to many astronomical discoveries, as well as many astrophysical discoveries. Among these discoveries is human understanding of the universe and the solar system. The solar system has 8 planets in total, and they are all unique and different from each other, with different forces and physical properties distinguishing them from one another. How the solar system works as well as fascination with space is the backbone from which the project of building a Solar System Simulator has arisen.

1.2 Purpose

This report aims to show and describe the process of creating a Solar System Simulator. Different important aspects that will be discussed are how the physical system is described with the help of different mathematical models and numerical methods, how the system is implemented from a graphical point of view, as well as different results obtained from the project. The report also aims to present facts and methods, rather than comparing approaches and presenting opinions.

Chapter 2

Implementation

2.1 Physical System

The physical system has been simplified in a few ways to save on computation time and for simplicity in creating a mathematical model. Firstly, the group decided early that the project would not delve into Einstein's equations on the theory of general relativity and instead the scope of the project was focused upon Newton's equations on the law of universal gravitation. This was agreed upon because while Einstein's equations do describe reality more accurately than Newton's equations, they are far more complex and do not change much in the ways of our solar system and its' behavior in a meaningful way.

Second, the planets were considered to be set perfectly inside a single xy-plane and the third dimension was not considered in the equations. The reason for this is that the planets, with the exception of Pluto - which was excluded from the system - have a maximum orbital incline of seven degrees which was considered to be insignificant.

The third simplification involves restricting the moons within the system to be solely influenced by their host planet and the Sun, while also ensuring that the moons do not exert any effects beyond their respective host planets. This was done to save on computation time since every new planet exponentially increases the amount of computations that have to be made every time-step in the Euler approximation process. The first simplified model only calculated the force between the sun and the planets, but the system was expanded to include gravitational force between each of the planets as well as the sun, hence the computationally heavy equations.

2.2 Mathematical Model

To simulate the simplified system, where the planets did not affect one another but the sun did, a force equation was made. The equation, as discussed before, utilized Newton's equation on the law of universal gravitation

$$F(t) = \frac{Gm_1m_2}{r_{21}(t)^2} \tag{2.1}$$

where *F (N)* is the force enacting on the planet, *G* ($\frac{m^2}{kg*s^2}$) is the gravitational constant, $m_1(kg)$ is the mass of the planet that is having a force enacted on, $m_2(kg)$ is the mass

of the planet which is enacting a force upon the planet, *r (m)* is the distance between the two planets, and lastly *t (s)* is time in seconds [1]. Another equation from Newton's is his second law of motion in which we derived the force equation

$$F(t) = ma(t) \tag{2.2}$$

where *a* ($\frac{m}{s^2}$) is the acceleration [2]. Rewriting this relationship but in vector form gives

$$\vec{F}(t) = \frac{Gm_1m_2}{|\vec{r}_{21}(t)|^2}\hat{r}_{21}(t) \tag{2.3}$$

where previous reoccurring variables are the same, but $\vec{r}_{21}$ is a vector, in this case two-dimensional, which represents the difference between the two planets position in the xy-plane. This of course means that since the right hand side is a vector, so is the left, that is to say $\vec{F}(t)$ is now also a two dimensional vector, allowing a way to express the acceleration in vector form

$$\vec{F}(t) = m\vec{a}(t). \tag{2.4}$$

This equation is important because it gives us a relationship between known attributes among the planets in our solar system and their acceleration. The reason this is important is because the acceleration is a double derivative of the position, which can be written as

$$\vec{F}(t) = m\frac{d^2\vec{r}}{dt^2}(t). \tag{2.5}$$

In section 2.3 the significance of this relationship will be made clear. In the simplified model only the sun enacted a force upon other planets. This model was afterwards improved upon when all planets were made to affect one another as well. The equation for the net force

$$F_{net}(t) = \sum_{i=0}^{n} F_i(t) \tag{2.6}$$

which then can be rewritten as

$$a(t) = \frac{1}{m_1}\sum_{i=0}^{n} F_i(t) \tag{2.7}$$

which if 2.1 substitutes the force part of 2.7, it gives

$$a(t) = \frac{1}{m_1}\sum_{i=0}^{n} \frac{Gm_1m_i}{r_i(t)^2} \tag{2.8}$$

which further simplifies into

$$a(t) = \sum_{i=0}^{n} \frac{Gm_i}{r_i(t)^2}. \tag{2.9}$$

Written in vector space, which is done to simulate the planets in a two-dimensional plane, the equation becomes

$$\vec{a}_{net}(t) = \sum_{i=0}^{n} \frac{Gm_i}{|\vec{r}_i(t)|^2}\hat{r}_i(t). \tag{2.10}$$

Equation 2.10 is now discretized and rewritten as

$$\vec{a}_{net}(k) = \sum_{i=0}^{n} \frac{Gm_i}{|\vec{r}_i(k)|^2} \hat{r}_i(k), \tag{2.11}$$

where $k = 0, 1, 2, 3....$ This is the final equation that was used in the simulation part of the project to estimate the acceleration.

2.3 Numerical Methods

The simulation uses Euler's method to numerically compute the *Ordinary Differential Equations* (ODE) to assess the trajectories of celestial bodies within a solar system model. This approach is based on the mathematical models presented in *chapter 2.2* by discretizing time and iteratively updating the positions and velocities of respective planets and their moons whilst under the influence of gravitational forces as shown in equation 2.11 and figure 2.1.

According to Merriam-Webster [3], the definition of *Celestial body* is any natural physical entity that exists in the universe. This term encompasses a wide variety of objects, including stars, planets and moons. In our simulation, celestial bodies are represented as objects with attributes corresponding to their physical characteristics - namely; mass, position, and velocity vectors. The initial conditions of each object are set according to empirical astronomical data acquired from NASA at [4] and [5]. Whilst the definition of celestial bodies includes stars such as the sun, in this simulation the sun isn't taken account to as a *Celestial body*.

2.3.1 Euler's Method

The simulation progresses in discrete time-steps of fixed size, denoted by *h*, which is a parameter controlling the resolution and stability of the integration. At each time-step, the acceleration vector for each celestial body is computed based on the gravitational forces exerted by other bodies. The velocities and positions are updated using the numerically calculated accelerations. The acceleration is based on the *ODE* function presented in chapter 2.2. The velocity $\vec{v}$ at the next time-step is found by:

$$\vec{v}_{new} = \vec{v}_{old} + h * \vec{a} \tag{2.12}$$

Subsequently, the position $\vec{p}$ is updated according to:

$$\vec{p}_{new} = \vec{p}_{old} + h * \vec{v}_{old} \tag{2.13}$$

Algorithm 1 presents an overview of the process used to simulate the orbits of celestial bodies using Euler's method. The algorithm is structured in two primary functions: SIMULATE_ORBITS and STOREPOSITIONS.

The function SIMULATE_ORBITS takes as input the list of the time-step h, the number of time-steps $num\_timesteps$, the gravitational constant G, the mass of the Sun m_sun, and the class *Celestial Body*. Which is the parent class of *Planet* and *Moon* in our hierarchical structure. SIMULATE_ORBITS outputs the position data for all bodies over the course of the simulation. Internally, this function initializes a *dictionary* - data structure that stores items in key-value pairs - to store the positions of each body. For each time-step, it iterates over all bodies to reset their acceleration and compute the gravitational forces acting upon them. It then updates the velocities and positions of each body using Euler's method. At specified intervals, determined by $store\_every$, it stores the current positions of all bodies using the helper function STOREPOSITIONS.

The function STOREPOSITIONS is called to append the current position of each celestial body to the *dictionary* created in SIMULATE_ORBITS. It is responsible for storing the positions at regular intervals, thus creating a dataset *positions* available to use for the visualisation inside of Blender.

The following code provides the implementation details for Algorithm 1:

Algorithm 1 Euler's Method for Orbital Simulation

Require: List of celestial bodies $planets$, timestep h, number of steps $num\_timesteps$, gravitational constant G, solar mass m_sun

Ensure: Position data for all bodies over time

function SIMULATE_ORBITS($planets, h, num\_timesteps, G, m\_sun$)
    $positions \leftarrow$ dictionary to store body positions
    for $step \leftarrow 1$ **to** $num\_timesteps$ **do**
        for all $body \in planets$ **and** $body.moons$ **do**
            $body$.reset_acceleration()
            $body$.apply_gravitational_forces($G, m\_sun, planets$)
        end for
        for all $body \in planets$ **and** $body.moons$ **do**
            $body$.update_velocity(h)
            $body$.update_position(h)
        end for
        if $step \mod store\_every = 0$ **then**
            STOREPOSITIONS($positions, planets$)
        end if
    end for
    return $positions$
end function

function STOREPOSITIONS($positions, planets$)
    for all $body \in planets$ **and** $body.moons$ **do**
        $positions[body.name]$.append($body.position$.copy())
    end for
end function

Note that the APPLY_GRAVITATIONAL_FORCES procedure mentioned in the pseudocode is an abstraction that represents the calculation of gravitational effects between the celestial bodies. This computation includes both the effects between the sun, planet and its moon(s) respectively. This abstraction would also includes the inter-planetary calculations, which would theoretically take a higher toll on the computations, but will be an available option for the user.

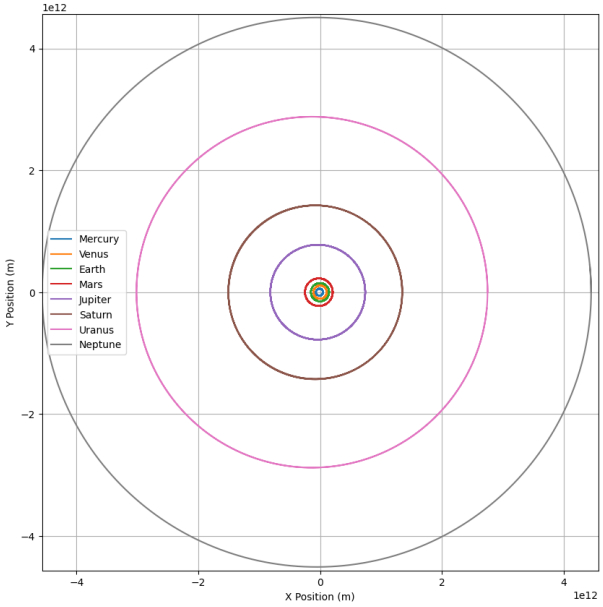

Figure 2.1: Plot of the 2D orbital system.

Before implementing the graphics within Blender, the position, speed, and acceleration data was plotted in both Python and MATLAB. Figure 2.1 represents the plot created in Python. It shows how the data gained from the ODE presented in *Algorithm 1* was used. The data gathered, constrained by the initial limits, provided the plot with sufficient realism. The legend in figure 2.1 shows a list of planets which were plotted within the system.

2.4 Graphical Implementation

The visualisation and graphical implementation was done inside the 3D creation suite Blender. This was done by creating an add-on for Blender which was written using the Blender python API (Bpy).

2.4.1 UI

The UI was created and drawn inside of a Panel class in the Python script within Blender. In order to create different buttons, unique operator classes were created. The most important part of the UI was the list of planets that would dynamically change when a user added or removed planets in the system. The list of planets was created in the form of a *UIList*, which allowed each planet to be created with the necessary attributes. As can be seen in figure 2.2, the user can select the different planets and the values in the sliders will be updated based on the selected planet. The *New* button is used to create a custom planet with default values and the *Add planet preset* button provides a list of planets in the solar system that the user can chose between adding. The *Remove* button will as the name implies remove the selected planet from the *UIList*.

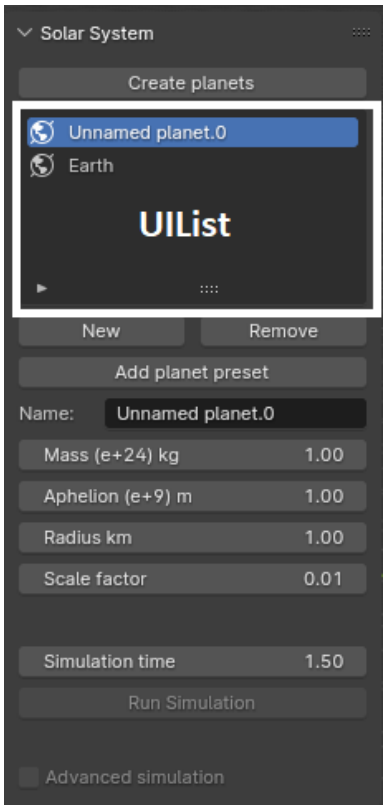

Figure 2.2: UI of the addon.

In order to create the objects for each planet, a *Create Planets* button was implemented. When pressed it creates a UV Sphere corresponding to each planet and places them in the scene. Only when the *Create planets* button has been pressed is the *Run Simulation* button enabled since it is dependant on the planets having been created. The *Advanced simulation* checkbox will also only be enabled once the planets have been created. The checkbox decides whether the simulation will calculate the two-body or the N-body problem. Above the *Run Simulation* button there is a slider that allows the user to decide how many years the simulation is going to calculate. Once the *Run Simulation* button is pressed, the simulation is run and returns a Python dictionary containing the simulated positions for each planet.

2.4.2 Animation and Material

As aforementioned, the software that was used for the visualisation of the project was Blender. The animation was implemented with the help of already simulated values, as well as *keyframes*. Here is an example to clarify how the animation process starts: A user interacts with the aforementioned UI, by doing so he/she creates an x-amount of planets and thereafter simulates the location of the selected planets. When the simulation is done the system uses the simulated values and adds a *keyframe* that stores the location for each simulated value for each planet. When the system is done with the whole simulation (including the *keyframing*), a user can manually start the simulation by starting the *Timeline*.

Chapter 3

Results and Discussion

3.1 Results

Different results throughout the project will be discussed in the following subsections.

3.1.1 Results from MATLAB

The MATLAB results can be seen in figure 3.1 and figure 3.2, where figure 3.1 shows Earth's orbital positions, while figure 3.2 shows Earth's different velocities.

3.1.2 Results from Python

The Python results are the code used for the planetary simulations as well as a plot of all planet positions, which can be seen in figure 2.1. The figure 2.1 plot depicts the simulation, which took into account forces that planets have on each other, the simulations step size was 5 minutes, and the simulation time was 170 years.

3.1.3 Results from Blender

The Blender results are that a person can download the add-on, start Blender, enable the add-on and then interact with the UI to create different planets. The user can change specific aspects like mass, position and so on, as well as create own imaginary planets. A user can also simulate the system to get a key-framed animation where the planets will move according to different physical properties and forces, an example scene can be seen in figure 3.3.

3.2 Software and Hardware Requirements

The only software requirement to run the visualization is having the software Blender installed on the PC. There are no specific hardware requirements to run the visualization, however, a better CPU and/or GPU will be able to perform all necessary calculations faster.

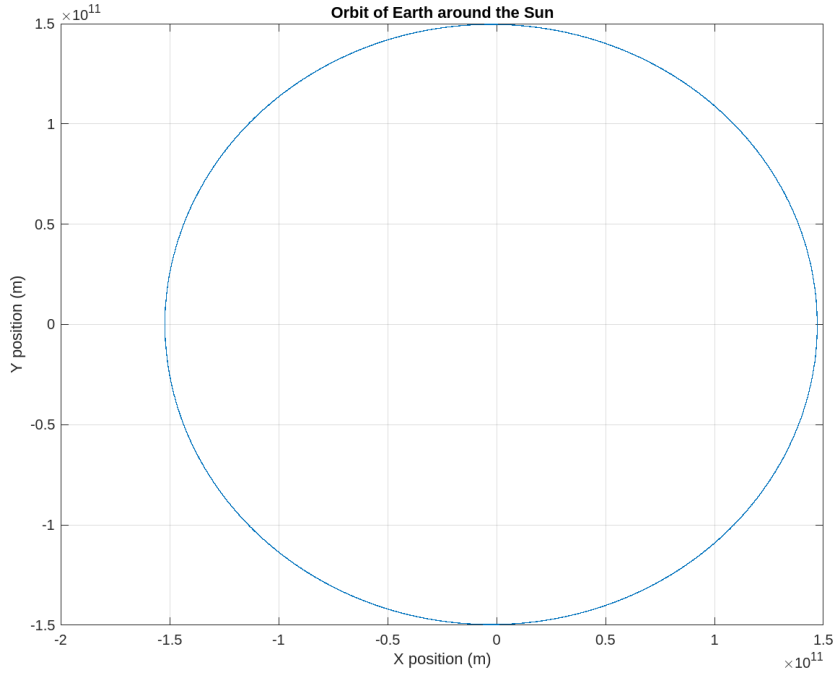

Figure 3.1: A MATLAB plot showing Earth's orbital positions.

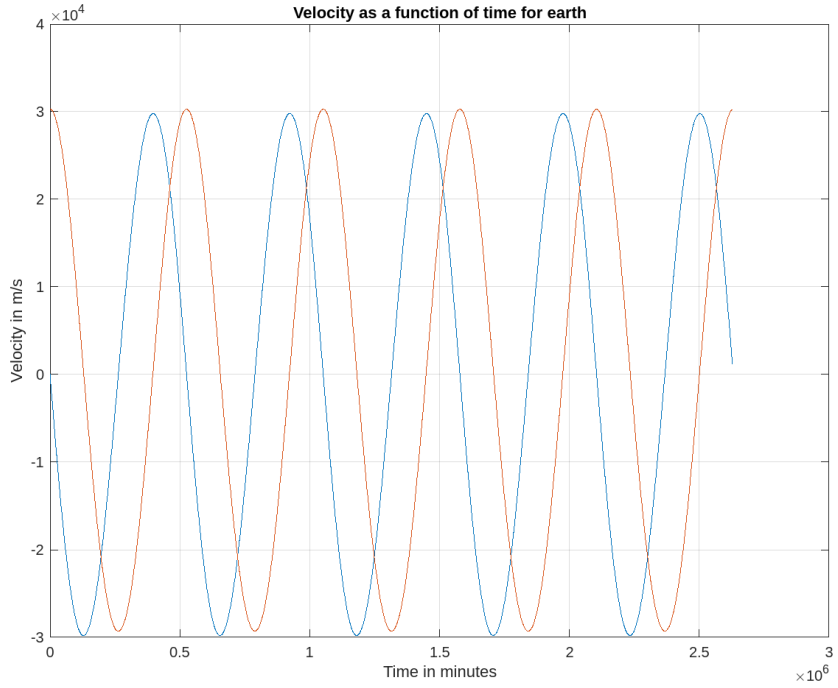

Figure 3.2: A MATLAB plot showing Earth's velocity in m/s along x and along y, with respect to time in seconds.

3.3 How to Use

For a user to be able to use the system, the user has to first download the zip file containing necessary textures and the Blender add-on, i.e. the code that will run in Blender. Once a user has installed the add-on, and started Blender, the Python script will be run automatically. When the script has been run, the user will see a UI menu in the *3D Viewport* editor type, see figure 2.2. A user can then interact with the UI to create different planets and change different parameters such as the *Scale factor* which is a factor that scales the system so that a user can for example see all the planets at once. A user can also interact with the *Simulation time* which is a parameter that determines how many years should be simulated. The *Advanced simulation* toggle lets a user choose if the simulation should take into account force that planets have on each other or not. If a person clicks on the *Add planet preset* button, a drop-down menu will pop up, which can be seen in figure 3.4. The planet presets are as the name suggests, planetary presets for existing planets in our solar system. A user can add an x-amount of planets from the planet preset and/or created an x-amount of imaginary planets through the *New* button seen in figure 2.2. Once the user has added the planets, the user can click on a planet in the list and if a user does so, the user will be able to change specific parameters for the planet, see in figure 2.2. Once the user is satisfied with all the planet parameters the user can click on the button *Create planets*, which will create the planets in the *3D Viewport*, an example can be seen in figure 3.3. When the planets have been created, a user needs to run the simulation by clicking on the *Run simulation* button in the UI. Once the system has been simulated, the user will be able to see the animation by starting the *Timeline*, which is also where *keyframes* that are generated for the simulation can be seen (if a user selects a planet). The *Timeline* and the *keyframes* can be seen on the bottom part of figure 3.3.

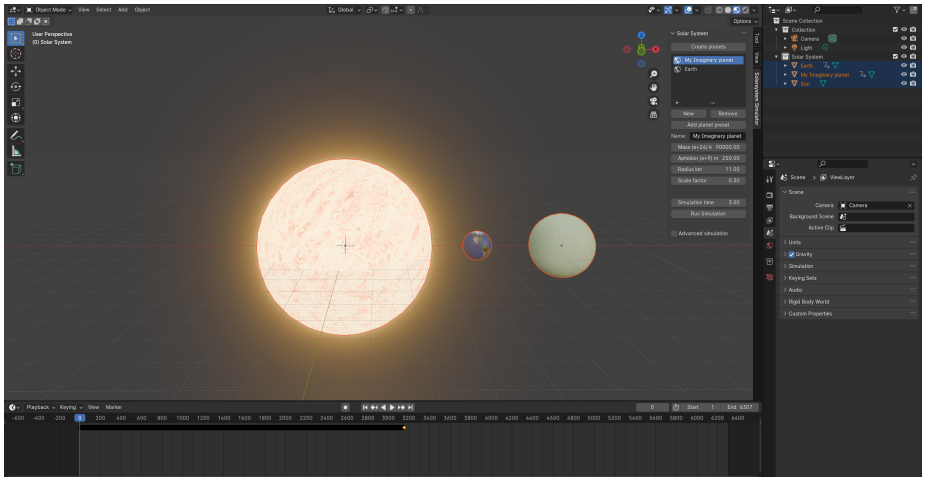

Figure 3.3: An image depicting an example scene after simulation, with Earth and an imaginary planet.

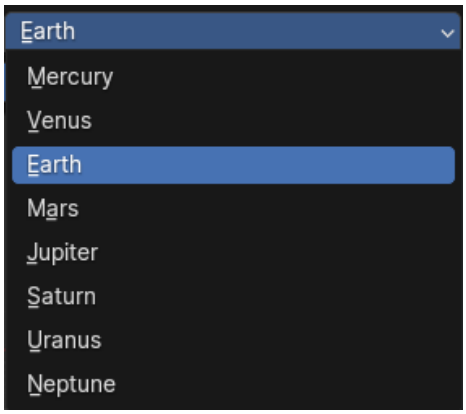

Figure 3.4: An image showing drop-down menu that appears if a user clicks 'Add planet preset'.

3.4 Further improvements / additions

There are multiple changes that could be made to the program and features to be added. Currently, all planets have a initial position along the same axis which is not realistic. To make the positions realistic, startdates with known positions could be implemented. Doing so would lead to future orbits matching their real-life equivalents.

A change that could be made to the simulation code was making the N-body simulation more performant. For each planet, the algorithm currently calculates one force per planet, meaning that some calculations are performed twice. The alternative would be to store the calculations to be reused instead of recalculated.

One feature that could be added to the visualization was the rotation of planets around their own axis. This could be a fixed rotational speed to save on performance, alternatively it could be simulated based on the planet's attributes.

3.5 Conclusion

In conclusion the project can be called a success. The Solar System Simulator was implemented and all the desired functionality was achieved. The end result is a robust Blender add-on that can be downloaded and used by anyone with an aspiration to create a solar system.

Bibliography

[1] Stern DDP. NASA; 2006. Accessed on March 15, 2024. Available from: https://pwg.gsfc.nasa.gov/stargaze/Sgravity.html.

[2] Hall N. NASA; 2023. Accessed on March 15, 2024. Available from: https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/newtons-laws-of-motion/.

[3] Merriam-Webster. Celestial body; 2024. https://www.merriamwebster.com/dictionary/celestial%20body. Website.

[4] Hall N. NASA; 2024. Accessed on March 15, 2024. Available from: https://nssdc.gsfc.nasa.gov/planetary/factsheet/.

[5] Williams DDR. NASA; 2022. Accessed on March 15, 2024. Available from: https://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html.