Mixed Reality & Simulation - Solar System

Kranzl, Manuel
ai22m038@technikum-wien.at

October 9, 2023

1 Introduction

The goal of this exercise was to implement a simple physics simulation of our solar system. The simulation should include all eight planets as well as earths moon. The sun should be the center of the simulation and therefore not move. It is sufficient to have circular orbits. Orbital periods for all planets shall be calculated to check the functionality of the simulation.

2 Solution

2.1 General Structure

The exercise was solved using C++ in combination with SDL2 for the output of the planets. Two classes were implemented. One for the planets themselves and one for the Solar system. While I will not go through the implementation of the output (even if it is pretty interesting) I want to give a short description of how to use all the features as well as how the physics simulation works.

2.2 Features

The moon will not be generated by default. If one wants to generate the moon the flag --moon is to be set, when calling ./main. The other functions can be called as follows:

j	Speed Down
k	Speed Up
n	Zoom Out
m	Zoom In
h	Earth Zoom
Esc	End the Simulation

While most of this commands are self explaining, Earth Zoom Mode will zoom in on earth as the new center. This is especially useful when starting the program with the earths moon enabled.
Note, that (with and without Earth Zoom) the Size of the planets are correct relatively to each other, but do not match the size of the orbits. Also a minimum radius of one pixel is implemented. The sun is displayed even smaller than the planets size scale, so one can

identify all planets. The size of the orbits are also correct relative to each other. When using Earth Zoom the Planets will appear even smaller, as the moon would otherwise be inside the earth. I choose to only use this smaller planets when Earth Zoom is enabled, as the other planets would get too small to see them when looking at the solar system as a whole.

2.3 Physics

In the main loop the following command gets called.

```
void SolarSystem::simulate(const double inc_step) {
    #pragma omp parallel num_threads(4)
    #pragma omp for
    for (Planet& planet : planets) {
                if(planet.getName()!="sun") {
            planet.resetGravityForces();
                  for(Planet& other_planet : planets) {
                                 if(other_planet.getName() != planet.getName()) {
                           planet.addGravityForces(other_planet);
                                 }
                  }
                }
    }
    for (Planet& planet : planets) {
                planet.update(inc_step);
    }
}
```

Basically the program first resets all forces acting on the bodies. Then it iterates through all other planets and adds up all the forces acting on the body itself. The sun will not receive any forces, as it should stay the center of the simulation and therefore not move. The forces are calculated using the following formula:.

$$F = G * \frac{m_1 * m_2}{r^2} \tag{1}$$

This calculation is implemented as follows.

```
void Planet::addGravityForces(const Planet& other) {
    double distance = this->getDistance(other);
    double force_abs = GRAVITATIONAL_CONSTANT * mass * other.getMass()
                       / (distance*distance); //undirected force
    force += ((force_abs * (other.getPos() - pos)) / distance);
}
```

After calculating all the forces (it is important to calculate all the forces first before moving planets, because their position is used in the force calculation), the planets now will move in their new direction.

```
void Planet::update(const double inc_step) {
    double old_pos_x = pos.x;
    velocity += (force * inc_step) / mass;
    pos += velocity * inc_step;
    if(old_pos_x < 0 && pos.x >= 0){
                rotations++;
    }
}
```

Here, the velocity delta is calculated using the following formulas.

$$F = m * \frac{\delta v}{\delta d} \tag{2}$$

2.4 Time measuring

As seen in the update-step every time a planet gets a full orbit an integer will be incremented. At the end of the simulation this integer gets added to the remaining rest of the last orbit. By dividing the number of earth orbits by the number of every planets orbits the output will have the same format as the table on the exercise sheet. The table can be seen in section 3.1.

3 Results

3.1 Orbit Times

The following table shows the calculated orbital periods OP_{calc} as well as the real world orbital periods OP_{real} and the error in the calculated orbital periods of the simulation.

Planet	OP_{real}	OP_{calc}	Error
Mercury	0.241	0.273	11.72%
Venus	0.615	0.597	3.02%
Earth	1.000	1.000	0.00%
Mars	1.881	2.102	10.51%
Jupiter	11.862	11.127	6.61%
Saturn	29.457	28.314	4.04%
Uranus	84.021	85.821	2.10%
Neptune	164.800	160.074	2.95%

Besides the obvious reason for the error margin in simulation (that the simulation might be far from perfect) one very important thing to note here is, that in this solar system simulation we only simulated perfectly round orbits, while the real world orbits are elliptical.

3.2 Screenshots

The following figures show screenshots of the finished simulation.

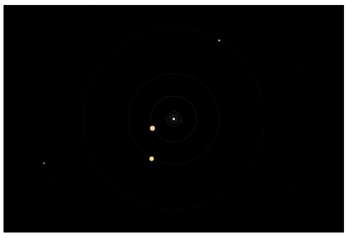
Figure 1: Solar System

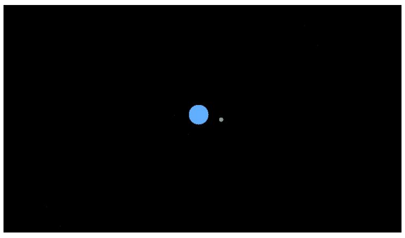
Figure 2: Earth Zoom

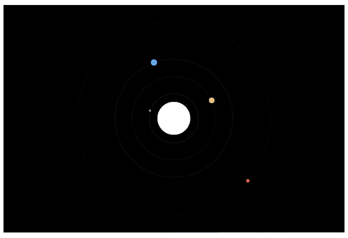
Figure 3: Inner Planets