Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

COMP 342 (Computer Graphics)

Mini Project Report

Submitted By:

Mani Dumaru

Roll no.: 15

CE 3rd year/ 2nd semester

Submitted to:

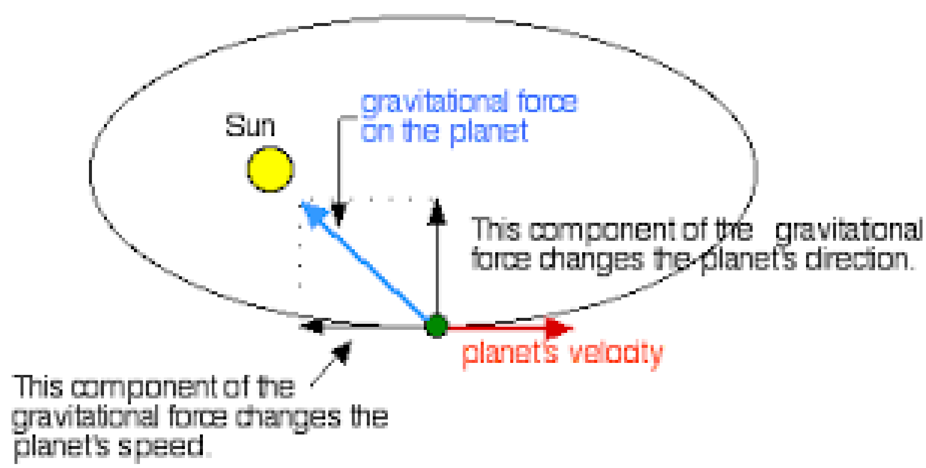
Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Submission Date: 3rd June, 2023

Mini Project on: Orbital Simulation

Gravitational Force attracts a lesser massive body towards the star while planets velocity tends to move it away in a straight line. They relatively form an orbit around the star in the shape of an ellipse as shown in the figure below.



Planets Orbit around a star.

Source Code: orbit.py, data.py

(Instruction: put data.py and orbit.py in the same folder location and run orbit.py)

Demo Video Link: simulation

Project Description

I've simulated a very simple 2D view of orbital motion of the planets around the Sun. For the project, I've used the real data for the distance between the Sun and the planets so, planets beyond the Jupiter could not be included. Their distance with the sun is beyond the viewing window that I've considered. Here is a snapshot of how I've represented the data of planets and the sun.

```
G = 6.67 * 10**-11
AU = 1.496 * 10**11
timestep = 86400
earth_route = []

sun = {
        "mass": 1.989 * 10**30,
        "radius": 20,
        "x": 0,
        "y": 0
}
earth = {
        "mass": 5.97219 * 10**24,
        "radius": 3.5,
        "distance": 149.6,
        "x": AU / 10**9,
        "y": 0,
        "vy": 29.783 * 1000, # m/s
        "vx": 0
}
venus = {
        "mass": 4.867 * 10**24,
        "radius": 3.4,
        "distance": 104.7,
        "x": 0.7*AU / 10**9,
        "y": 0,
        "vy": 35.02 * 1000,
        "vx": 0
}
```

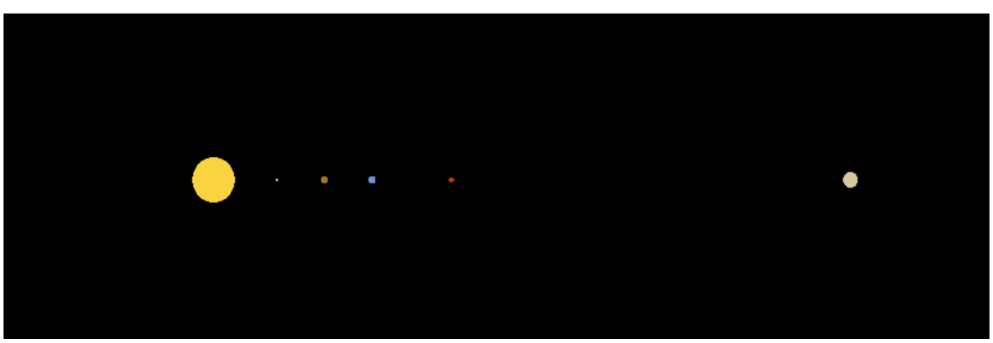
```
mercury = {
        "mass": 3.285 * 10**23,
        "radius": 1.5,
        "distance": 59.84,
        "x": 0.4*AU / 10**9,
        "y": 0,
        "vy": 47.36 * 1000,
        "vx": 0
}
mars = {
        "mass": 6.39 * 10**23,
        "radius": 2.5,
        "distance": 224.4,
        "x": 1.5*AU / 10**9,
        "y": 0,
        "vy": 24.08 * 1000,
        "vx": 0
}
jupiter = {
        "mass": 1.898 * 10**27,
        "radius": 7,
        "distance": 600,
        "x": 4.02*AU / 10**9,
        "y": 0,
        "vy": 13.06 * 1000,
        "vx": 0
}
```

Rendering Planets:

For the radius of the planets, if I had used the original radius and reduced them by a constant scale for all planets, the radius of the Sun would be very large compared to other planets. In that case, any planets won't be visible at all in the viewing window scale. So, for the radius, I've used manual data so as to make every object visible. Code for rendering planets.

```python
def display(planet, x, y, z):
    glColor3f(x,y,z)
    glBegin(GL_TRIANGLE_FAN)
    r = planet["radius"]
    for i in range(0,360):
        theta = 3.1415926 * float(i) / float(180)
        x = r * math.cos(theta)
        y = r * math.sin(theta)
        position_x = x + planet["x"]
        position_y = y + planet["y"]
        glVertex2f(position_x , position_y)
    glEnd()
    glFlush()
```

This function takes as arguments, a planet and colour (RGB value) of the planet. The planet variable already has the initial coordinates of the planet in the viewing window with respect to Astronomical Units (*Distance from the Earth to the Sun. i.e., for* <u>*Earth the distance is 1AU*</u> *while for* <u>*Venus it is 0.7AU*</u>) Triangle fan is used to draw a solid circle with the planet's colour at the calculated position. *Initial position of the planets up to the Jupiter.*



Displacement Calculation

Using the gravitational constant, the mass of individual planet, the mass of the sun, and distance between them, the force of attraction is calculated. This attractive force is separated as force in x direction and in y direction using trigonometry. Using the force of attraction in x and y direction, the velocity of planet in both x and y direction is calculated. This velocity is then used to calculate the displacement of the planet in both the directions using time step of 86400 seconds (1 day). The planets new position is updated and is rendered again and again after clearing the screen in fixed interval.

Function for displacement calculation:

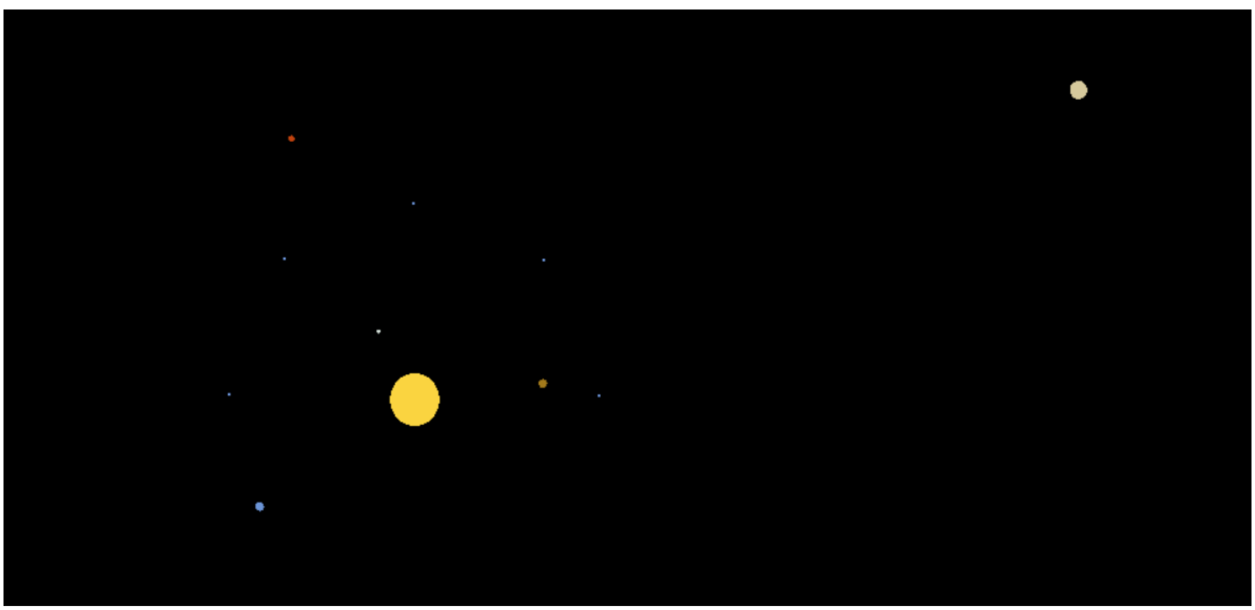
```python
def displacement_calculator(planet):
    force_e = (data.G * data.sun["mass"] * planet["mass"]) / (planet["distance"] * 10**9) **2
    theta = math.atan2(planet["y"], planet["x"])
    angle = theta * (180/math.pi)
    if (angle > 0):
        fx = -force_e * math.cos(theta)
    else:
        fx = force_e * math.cos(theta)
        fx *= -1
    fy = force_e * math.sin(-theta)
    velX = (fx * data.timestep) / planet["mass"]
    velY = (fy * data.timestep) / planet["mass"]
    planet["vx"] += velX
    planet["vy"] += velY
    dx = (planet["vx"] * data.timestep) / 10**9
    dy = (planet["vy"] * data.timestep) / 10**9

    return dx, dy
```

Tracing Earth’s Path:

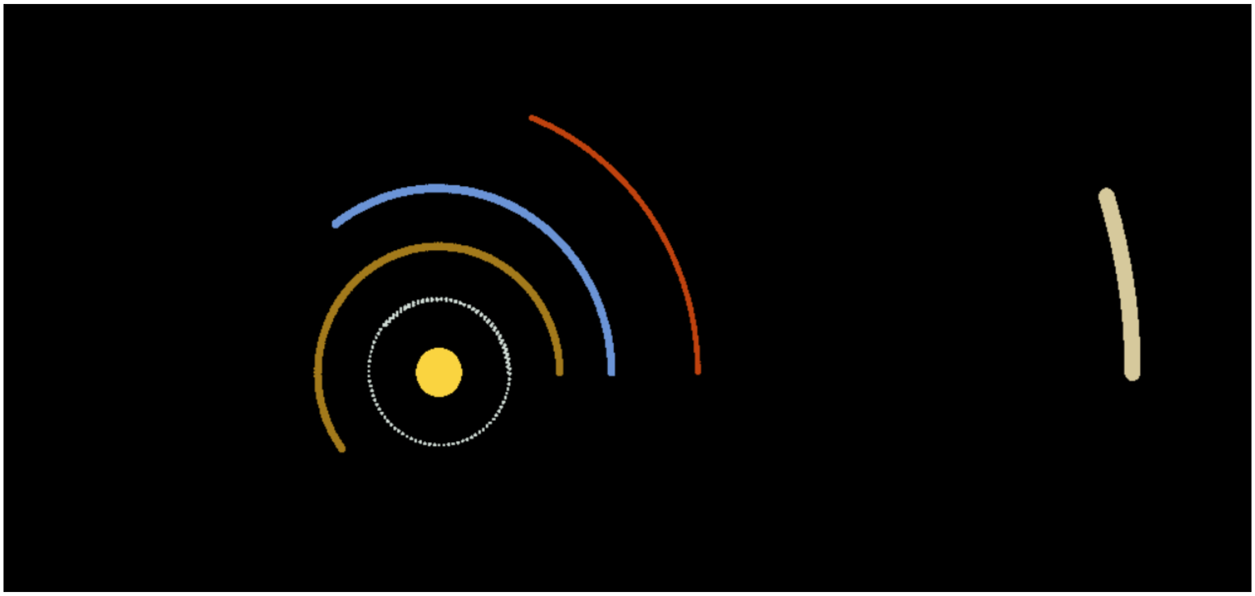
```
earth_position = (data.earth["x"], data.earth["y"])
if counter % 45 == 0:
    data.earth_route.append(earth_position)
counter += 1
if (len(data.earth_route) > 8):
    data.earth_route.pop()
for position in data.earth_route:
    glColor3f(0.419,0.576,0.839)
    glPointSize(2)
    glBegin(GL_POINTS)
    glVertex2f(position[0], position[1])
    glEnd()
    glFlush()
```

earth_position is the current position of the earth and the counter variable is set to 0 in the beginning of the code. Counter variable is incremented after the position of the earth is updated every time. Whenever its value is perfectly divisible by 45 i.e., at each 45-degree angle, earths position is appended to the list and at that position, a point is rendered. In this way, the path of earth is traced. A total of 9 such points are rendered.



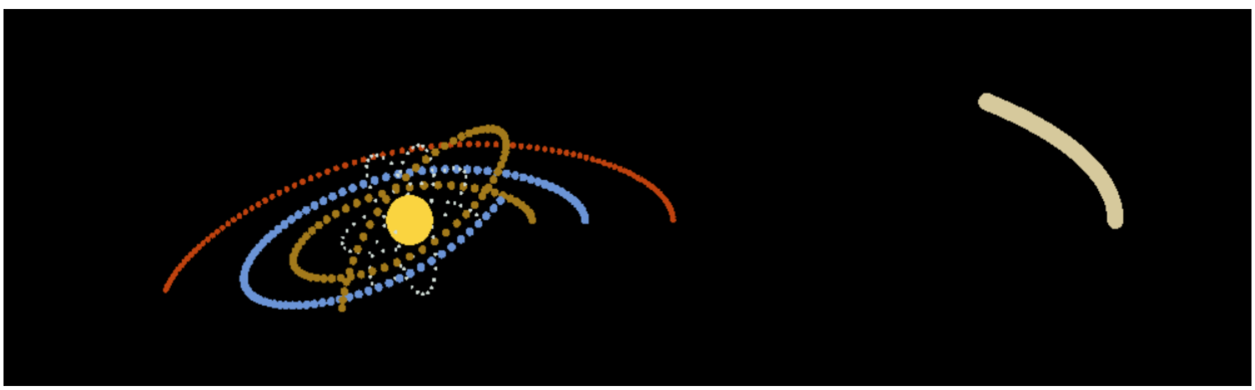
Earth’s path traced

Orbital path without clearing the screen:



When gravitational constant is increased:

Increased force of attraction: higher velocity around the Sun



Driver Code:

```
while(True):
    # glClear(GL_COLOR_BUFFER_BIT)
    display(data.sun, 0.98, 0.83, 0.25)
    display(data.mercury,0.81, 0.86, 0.83)
    display(data.venus, 0.64, 0.48, 0.105)
    display(data.earth, 0.42, 0.58, 0.84)
    display(data.mars, 0.75, 0.26, 0.055)
    display(data.jupiter, 0.84, 0.79, 0.61)

    mercury_displacement = displacement_calculator(data.mercury)
    data.mercury["x"] += mercury_displacement[0]
    data.mercury["y"] += mercury_displacement[1]

    venus_displacement = displacement_calculator(data.venus)
    data.venus["x"] += venus_displacement[0]
    data.venus["y"] += venus_displacement[1]

    earth_displacement = displacement_calculator(data.earth)
    data.earth["x"] += earth_displacement[0]
    data.earth["y"] += earth_displacement[1]

    mars_displacement = displacement_calculator(data.mars)
    data.mars["x"] += mars_displacement[0]
    data.mars["y"] += mars_displacement[1]

    jupiter_displacement = displacement_calculator(data.jupiter)
    data.jupiter["x"] += jupiter_displacement[0]
    data.jupiter["y"] += jupiter_displacement[1]
```

```
glutSwapBuffers()
time.sleep(0.01)
```

The planets are rendered, their positions are updated and the screen is cleared every 0.01 seconds which makes them appear to be revolving around the Sun.

Conclusion:

Hence, in this way the orbital motion of the planets around the Sun is simulated. Remaining 3 planets (Saturn, Uranus and Neptune) could not be rendered as this scale of distance with the sun was not enough for them to appear in the viewport. The screen was cleared constantly in a very small interval for the effect of motion of the planets.

References:

Simulating Planetary Orbits — 50 Examples 1.0 documentation. (n.d.).

https://fiftyexamples.readthedocs.io/en/latest/gravity.html