Project 16 - Simulation of Solar System with 8 Planets

Simulation Link

Problem Statement:

Simulate a solar system with at least five planets. This may not necessarily be our solar system but use realistic, astronomical, values.

Solution:

The solar system is a complex system with many scientific laws in motion and many mysteries still to be solved. Simulating the solar system can be done using many different ways. In this project, we have used four primary equations:

1. Newton's Gravitational Equation

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

where,

F = force
G = gravitational constant
m_1 = mass of object 1
m_2 = mass of object 2
r = distance between the centers of the masses

2. Orbital Velocity Equation

$$v = \sqrt{\frac{GM}{r}}$$

where,

v = velocity
G = gravitational constant
r = distance between the centers of the masses

3. Momentum Equation

$$p = mv$$

where,

p = momentum
m = mass
v = velocity

and in continuation,

$$p = Ft$$

where,

p = momentum
f = force
t = time

4. Velocity/Distance/Time Equation

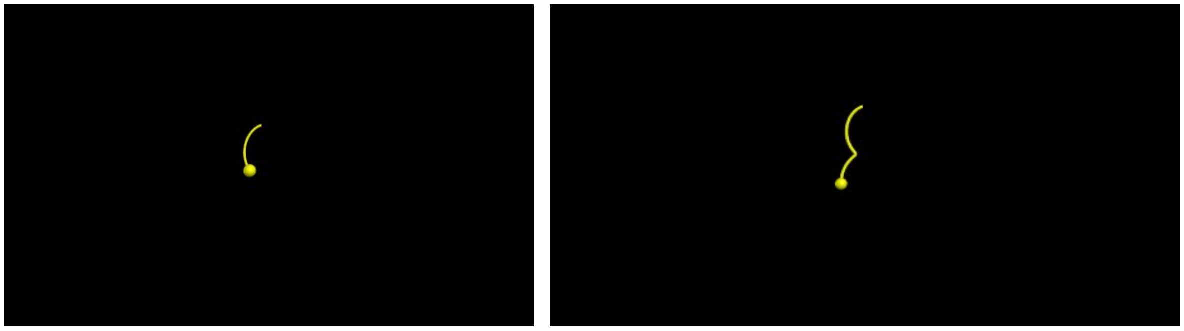
$$v = \frac{d}{t}$$

where,

v = velocity
d = distance
t = time

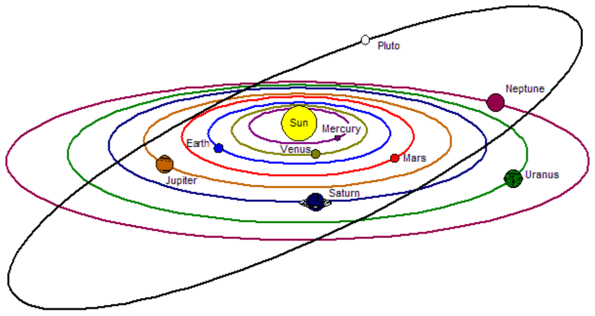
There are some assumptions that need to be considered before the simulation is implemented. These assumptions include:

1. Newton's Gravitational Law and Newton's Third law states that force acts in the form of a pair. This means that the Sun also experiences a small amount of force from all the planets. However, for our model it is safe to assume that this force is so small that it can be ignored. If we chose to include this combined force in our simulation, then the sun would start to slightly deviate from its path taking all the planets along with it which would be accurate enough but would mess up the orbits making trails all over the place. This is mainly because we have set the change in time to 2 days to observe the orbits of planets in the outer belt of the solar system. Based on this same assumption, the interplanetary forces between planets will also be ignored. A demonstration however is included in the figures below which gives a vague idea of how the simulation looks if the assumption was not made:

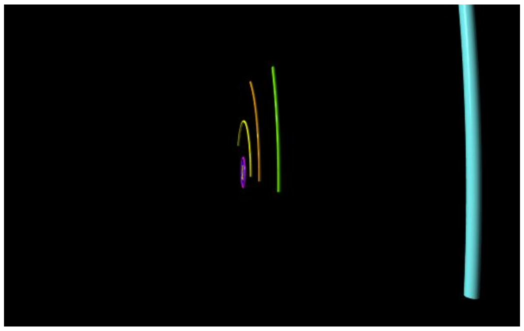


   It can be seen that the sun starts to move due to the force on it by all the planets even after it is not given an initial momentum

2. The planets orbit the Sun with an inclination. Earth is considered to have a base inclination of zero degrees and other planets draw their angles from this base value. Pluto has the highest inclination and eccentricity. This Three Dimensional inclination is ignored in our simulations. The figures below demonstrate how the orbits really are VS how they appear in the simulation due to our assumption:

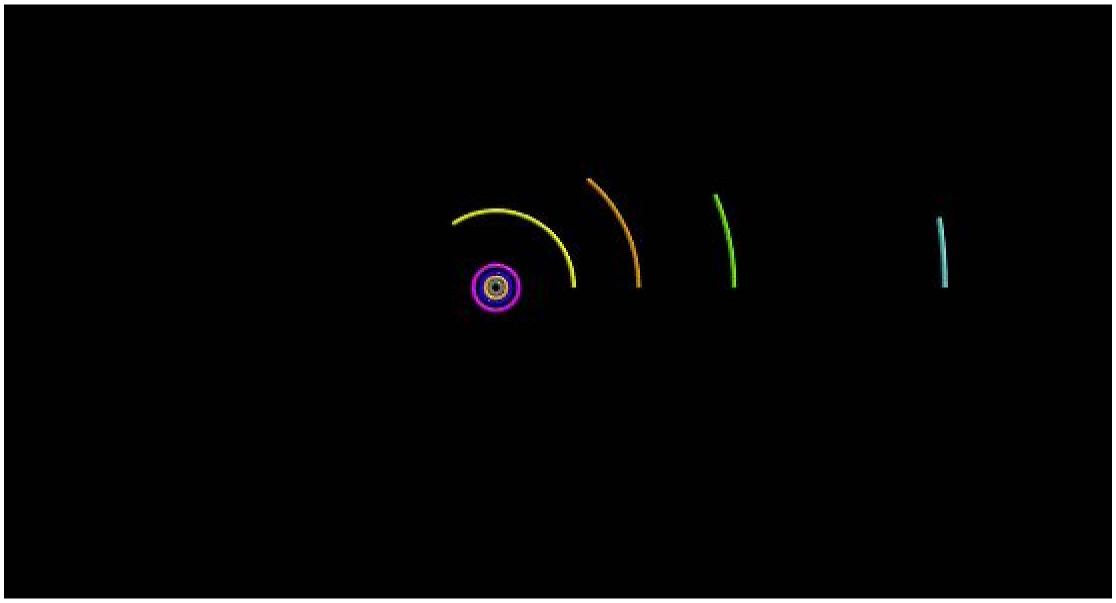


Real



Simulation

3. For orbits that have a lower value of eccentricity, the length of the orbit is similar to that of a circle. However, when one mass is nearly negligible when compared to the other mass we can use the orbital speed equation to approximate the speed of the orbiting body. This will result in a much more circular orbit in contrast to real orbits which are elliptical in nature. This can be observed in the figure below:



For cases when masses are not negligible, like in the case of binary star systems, then it is referred to as the gravitational two-body problem which is not the subject or area of study of this project. Note: the acceleration of Earth is also not constant because the Earth is not going in a straight line but is actually moving in a circle. The velocity has to change at some places. This causes the Earth to be fast and slow at different positions during its orbit.

With these three assumptions in place, we can simulate a functional solar system model with real astronomical values. The values in this project are 100% to scale and real values.

Procedure:

The online platform Trinket.io was used to create this simulation which used GlowScript Version 3.0, a modified form of VPython.

Animation:

The actual simulation is made possible through the Euler-Cromer method which is widely used for such animations. To sum up Euler-Cromer in simple steps,

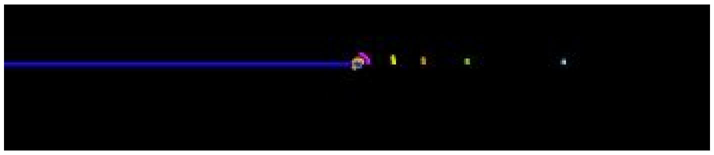
1. We turn the derivative into a slope formula
2. Multiply it by the change in x (Δx) also known as the step-size/rate
3. We then split the change in the function into its next value and current values
4. The current value is then added to the other side

5. This makes it possible for us to calculate the next value using the current value and adding it in the expression we have for the derivative.
6. The process is repeated multiple times and we get a smooth simulation

Objects in Simulation:

1. Spheres (https://www.glowscript.org/docs/VPythonDocs/sphere.html)

    Each of the planets and Sun itself is represented by a sphere that has a radius, and a position which is represented by a vector and a mass. A starting momentum is also provided in order to get a fluid motion. Otherwise, the planets will never move and would demonstrate a straight line motion as seen below:



Sphere is practically the only main object of this simulation.
The trail is made possible using the make trail function of VPython.

Variables:

1. There is one main function in the program named **equation** which calculates the force on two celestial bodies. The variables used in that function include **rad** which calculates the distance between the two objects, **magnitude** that converts it into a form that can be used with the Gravitational Force Equation, while **unitdistance, force_magnitude** and **force_vector** all contribute in returning the force in its vector form so that it can be used to update the momentum and position later in the program.
2. **dt** is the change in time which is kept at a step of **2 days** so orbits of planets in the outer belt can also be observed because they take a significant amount of time to complete their orbit around the sun. **t** is the initial time that is updated throughout the program.

Initial Values:

1. The distances are kept as their real astronomical values while the same goes for mass and radius of the bodies.
2. Gravitational Constant is also kept at its original value.
3. A starting velocity is required by the objects so they could start to make the orbital trail and for this purpose, velocity is calculated by the Orbital Velocity Equation mentioned above. This is kept in the **vstart** variable with respective planet names.
4. Momentum is then updated with the calculated velocity as an attribute of the sphere object. **vstart** is multiplied by mass because momentum = mass x velocity.

Final Values:

No final values are required by the problem and hence the simulation continues till time is less than 165 years, orbital time of Neptune; the planet farthest away from Sun.

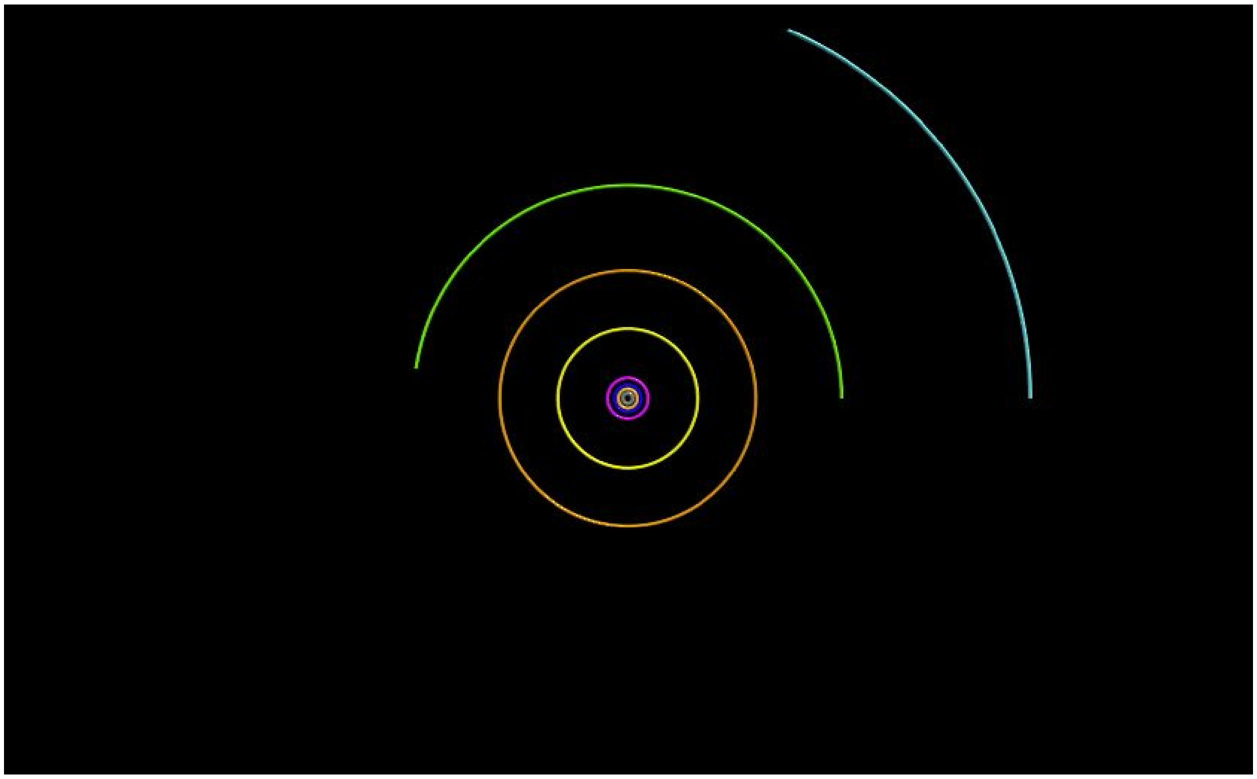
The next page features the Code Block for this simulation.

Code Block: (values for time step have been mentioned earlier)

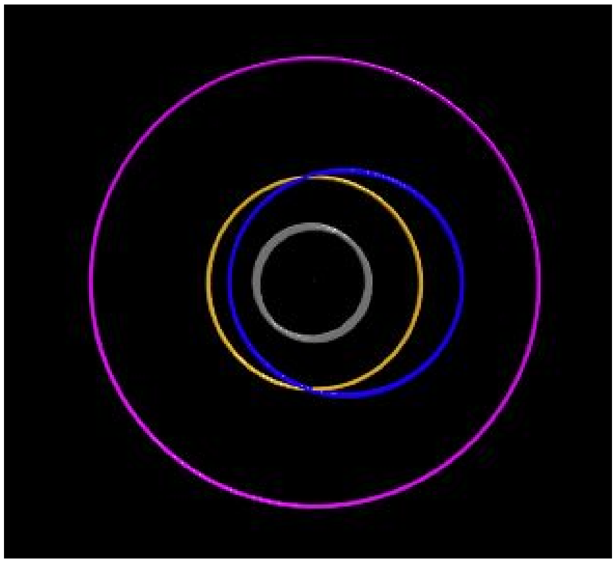
```
GlowScript 3.0 VPython
#Constants
G=6.67e-11
SunMercuryDistance = 5.7e10
SunVenusDistance  = 1.08e11
SunEarthDistance = 1.5e11
SunMarsDistance = 2.28e11
SunJupiterDistance = 7.79e11
SunSaturnDistance = 1.43e12
SunUranusDistance = 2.388e12
SunNeptuneDistance = 4.5e12

SunMass = 1.9891e30
SunRadius = 6.96e8
#Force Functions
def equation(x1,x2):
  G = 6.67e-11
  rad = x1.pos-x2.pos
  magnitude = mag(rad)
  unitdistance = rad/magnitude
  force_magnitude = G*x1.mass*x2.mass/magnitude**2
  force_vector = -force_magnitude*unitdistance
  return force_vector
#Starting Velocity
vstartMercury=sqrt(G*1.9891e30/SunMercuryDistance)*3.285e23
vstartVenus=sqrt(G*1.9891e30/SunVenusDistance)*4.867e24
vstartEarth=sqrt(G*1.9891e30/SunEarthDistance)*5.972e24
vstartMars=sqrt(G*1.9891e30/SunMarsDistance)*6.39e23
vstartJupiter=sqrt(G*1.9891e30/SunJupiterDistance)*1.898e27
vstartSaturn=sqrt(G*1.9891e30/SunSaturnDistance)*5.683e26
vstartUranus=sqrt(G*1.9891e30/SunUranusDistance)*8.681e25
vstartNeptune=sqrt(G*1.9891e30/SunNeptuneDistance)*1.024e26
#Making Objects
sun = sphere(pos=vector(0,0,0), radius=SunRadius, color=color.yellow, mass = SunMass, momentum=vector(0,0,0),
make_trail=True)
earth = sphere(pos=vector(SunEarthDistance,0,0), radius=6.378e6, color=color.blue, mass = 5.972e24,
make_trail=True, momentum=vector(0,vstartEarth,0))
mercury = sphere(pos=vector(SunMercuryDistance,0,0), radius=2440000, color=color.gray(0.5), mass = 3.285e23,
make_trail=True, momentum=vector(0,vstartMercury,0))
venus = sphere(pos=vector(SunVenusDistance,0,0), radius=6052000, color=vector(1,0.7,0.2), mass = 4.867e24,
make_trail=True, momentum=vector(0,vstartVenus,0))
mars = sphere(pos=vector(SunMarsDistance,0,0), radius=3390000, color=color.magenta, mass = 6.39e23,
make_trail=True, momentum=vector(0,vstartMars,0))
```

```
jupiter = sphere(pos=vector(SunJupiterDistance,0,0), radius=69911000, color=color.yellow, mass = 1.898e27,
make_trail=True, momentum=vector(0,vstartJupiter,0))
saturn = sphere(pos=vector(SunSaturnDistance,0,0), radius=58232000, color=color.orange, mass = 5.683e26,
make_trail=True, momentum=vector(0,vstartSaturn,0))
uranus = sphere(pos=vector(SunUranusDistance,0,0), radius=25362000, color=color.green, mass = 8.681e25,
make_trail=True, momentum=vector(0,vstartUranus,0))
neptune = sphere(pos=vector(SunNeptuneDistance,0,0), radius=24622000, color=color.cyan, mass = 1.024e26,
make_trail=True, momentum=vector(0,vstartNeptune,0))
#Main Loop
dt = 24*60*60*2
t = 0
while t<24*60*60*365.25*165:
  rate(100)
#Getting the Force
#   sun.force=equation(earth,sun) + equation(mercury,sun) + equation(venus,sun) + equation(mars,sun) +
equation(jupiter,sun) + equation(saturn,sun) + equation(uranus,sun) + equation(neptune,sun)
  earth.force=equation(earth,sun)
  mercury.force=equation(mercury,sun)
  venus.force=equation(venus,sun)
  mars.force=equation(mars,sun)
  jupiter.force=equation(jupiter,sun)
  saturn.force=equation(saturn,sun)
  uranus.force=equation(uranus,sun)
  neptune.force=equation(neptune,sun)
#Updating Momentum
#   sun.momentum = sun.momentum + sun.force*dt
  mercury.momentum = mercury.momentum + mercury.force*dt
  venus.momentum = venus.momentum + venus.force*dt
  earth.momentum = earth.momentum + earth.force*dt
  mars.momentum = mars.momentum + mars.force*dt
  jupiter.momentum=jupiter.momentum+jupiter.force*dt
  saturn.momentum=saturn.momentum+saturn.force*dt
  uranus.momentum=uranus.momentum+uranus.force*dt
  neptune.momentum = neptune.momentum + neptune.force*dt
#Updating Position
#   sun.pos = sun.pos+sun.momentum/sun.mass*dt
  mercury.pos = mercury.pos+mercury.momentum/mercury.mass*dt
  venus.pos = venus.pos + venus.momentum/venus.mass*dt
  earth.pos = earth.pos + earth.momentum/earth.mass*dt
  mars.pos = mars.pos + mars.momentum/mars.mass*dt
  jupiter.pos=jupiter.pos+jupiter.momentum/jupiter.mass*dt
  saturn.pos=saturn.pos+saturn.momentum/saturn.mass*dt
  uranus.pos=uranus.pos+uranus.momentum/uranus.mass*dt
  neptune.pos = neptune.pos + neptune.momentum/neptune.mass*dt
#Updating Time
  t = t + dt
```

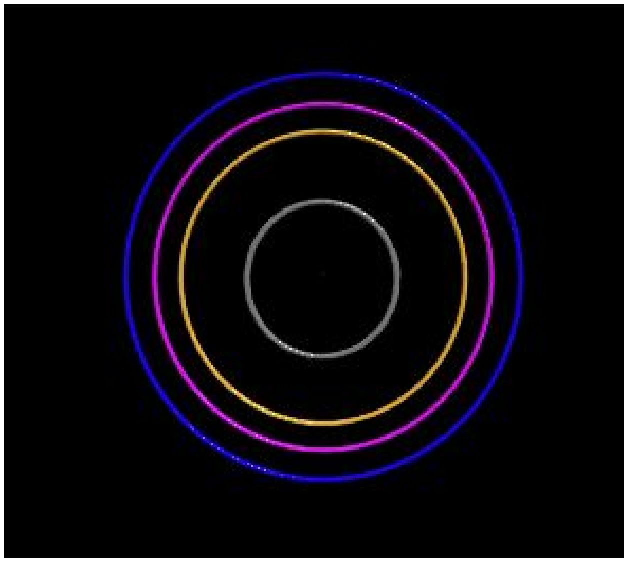
Results:



The figure above shows the final results of this simulation where all the planets are in their respective orbits. In order to test Newton's Gravitational Law, we can change the distance between planets or their masses to observe different orbits.

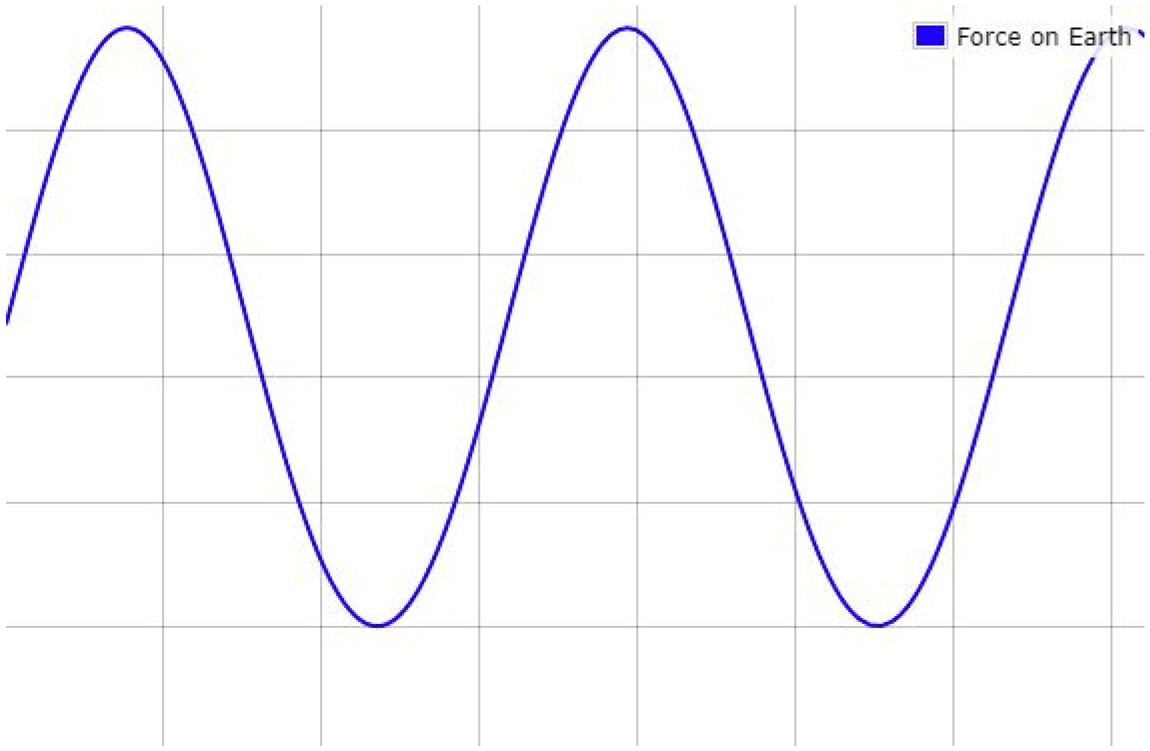


*Changing Earth's Mass



*Changing Mars's distance from Sun

The animation is accomplished by using the force function to calculate the force between Sun and the planets after which the momentum is updated by dividing the force vector with change in time. The position is then updated using the momentum and distance equation in pairs to calculate the change in position which is added to the current position. The time is then incremented skeeping the simulation alive. A graph is attached below to show the variation of force on the y-axis with change in time:



The graph is like this due to multiple revolutions around the sun.