# University of Portsmouth

# Computer Games Technology

# CT5MEGA

# UP784120

# Word count: 100

**Background:**

The purpose of this project is to create an Orrery that features one star, nine planets and nine moons. The user should be able to see a visual representation of the orbits and see the name of the planet/moon when it is selected. Also, he should be able to scale the time and be able to place the camera on one of the planets or moons.

The orrery is a mechanical model of the Solar System. The device is driven by a clockwork mechanism that simulates the motion of the planets around the Sun. (Williams, 2016, para.2)

I decided to create the orrery by using the hierarchic relation between the star, planets and moons. Trigonometry was used in order to plot the orbits and to draw the ellipses while Quaternions were used to rotate the planets around their own axis and implement the axial tilt. I also managed to draw the name of the planets when they are selected with the mouse. I was not able to scale the time and place cameras on different moons in order to give the possibility to the user to see the orrery from different perspectives.

**Analysis:**

Requirement 1: Planets rotating around each other

The artefact is going to be based on hierarchy relation between planets. Each planet's orbit is going to be plot around an origin point that is going to be represented by the parent planet's x, y, z coordinates. The hierarchy relationship between planets is important because each planet position is calculated using the position of another planet.

Requirement 2: Elliptical Orbits

The Elliptical Orbits are built based on the Kepler's Laws (Kepler's Laws, n.d), laws which describe the motion of planets across the solar systems. Ellipses are plot using the Law of Orbits that uses Trigonometry in order to calculate the position of each planet. The semi-major and semi-minor axis are used to define the ellipse's longest and shortest diameter. The x and y coordinates are going to be stored in a Vector that is going to have its position updated. (Ellipse, n.d)

```
if (increment <= 360)
{
    //plot the orbit
    increment = increment + speed;
    xPos = eccentricity+motherPlanetX + majorAxis * Mathf.Cos(increment);
    zPos = motherPlanetZ + minorAxis * Mathf.Sin(increment);
}
else
{
    increment = 0.0f;
    xPos = 0.0f;
    yPos = 0.0f;
}
//transform the position of the planet
transform.position = new Vector3(xPos,yPos,zPos);
```

[Primary Source]

Requirement 3: Planet rotation around its axis and axial tilt

For this requirement, the Euler Angles and Quaternations can be used in order to rotate the planet. One of the problem is that multiple angle combinations are going to be used and this can lead to ambiguity when using the Euler Angles. The Quaternations seem to be the best choice because the rotations are done in order, one after another. We must rotate the object along the Z axis to create the axial tilt and then along the Y axis in to rotate the planet around its axis.

Requirement 4: Orbital Inclination

I believe that the point slope form of the line equation or rotations can be used to implement this feature. I am going to calculate the inclination by indicating a slope because we know the coordinates of the Sun and the x coordinate of each planet.

The Sun's plane will be the one used as reference and the orbital inclination of the other planets will be calculated by defining a slope. Using the Sun's x and y coordinates, planet's x coordinate and a defined slope we can compute the y coordinate of the planet.

```
public float inclination(float x,float y, float x1,float y1, float slope)
{
    y = y1 + slope * (x - x1);
    return y;
}
```

```
yPos= inclination(xPos, yPos, motherPlanetX, motherPlanetY, angle); //calculate the Y position using the slope equation
```

[Primary Source]

Requirement 5: Graphical display of the orbits

To display the orbits, I used the same principles that were used to plot elliptical orbits but this time I line will be drawn between every two points so I can get the graphical representation of the orbit.

Requirement 6: Implement the possibility of selecting each planet or moon with the mouse

For this requirement, a geometric intersection test should be implemented and also there should be made transformations from World coordinates to View coordinates. I decided to do a circle-point intersection test, using the Pythagorean theorem, to determine if the mouse is in range of one planet or not.

```
//calculate the distance between two points using the Pythagorean Theorem
public float distance(float x, float y, float z, float w)
{
    float result, e1, e2 = 0;
    e1 = (z - x) * (z - x);
    e2 = (w - y) * (w - y);
    result = Mathf.Sqrt(e1 + e2);
    return result;
}
```

```
public void inRange()
{
    //calculate the distance between the mouse point and the object
    if (distance(mousePos.x, mousePos.y, twoD.x, twoD.y) <= 10)
    {
        intersection = true;
    }
    else
        intersection = false;
}
```

[Primary Source]

Another possibility would be to perform an 3D intersection test transforming the mouse coordinates from View Coordinates to World coordinates into a 3D point and creating a ray-plane collision test.

**Discussion and Conclusion:**

Moving the orbits:

I had some problems drawing the orbits because the line was drawn while the trajectory was plot so I was receiving an instantaneous representation of the orbit. It worked for the planets because their orbits are plot around the Sun and they are not moving, but, because the moons are rotating around the planets, their orbit was also changing the position. After doing some research I did find out a solution. The solution was to create an array of 360 elements that contains each line's coordinates. (Draw circle around game object to indicate radius, 2016) The line's coordinates are calculated using the coordinates of the parent planet.

Also, I could have implemented a function to turn on and off the orbits display.

Rotations:

I believe that storing the position of the planets and moons in matrices would have been a better idea because so I could also have implemented the axial tilt and the rotation around own axis using matrices. I think that my approach of using Quaternions is not showing a mathematical principle because I was just indicating the rotation axis and the angle of rotation.

World View-Screen View transformation:

I didn't manage to implement a transformation matrix to transform the objects coordinates from world space to view space so I admit that I used a built-in feature of Unity that does that.

I believe that I managed to use some mathematical concepts such as Trigonometry, Quaternion, the Pythagorean theorem, and slope equations but in the same time I think that I could have chosen alternative solution, such as using matrices. Because I didn't use them from the beginning, it was hard to implement some features such as world view-screen view transformations and rotations. I think I should

have done more research before choosing my solutions to be sure that my approach can be used to implement all the features.

**Reference:**


http://hyperphysics.phy-astr.gsu.edu/hbase/kepler.html

Williams, M. (2016). *What is an orrery?*. Retrieved from: https://www.universetoday.com/44671/what-is-an-orrery/

Ellipse. (n.d). Retrieved from: http://astronomy.swin.edu.au/cosmos/E/Ellipse

Kepler's Laws. (n.d). Retrieved from: http://hyperphysics.phy-astr.gsu.edu/hbase/kepler.html

Draw circle around game object to indicate radius. (2016) .Retrieved from: https://gamedev.stackexchange.com/questions/126427/draw-circle-around-gameobject-to-indicate-radius