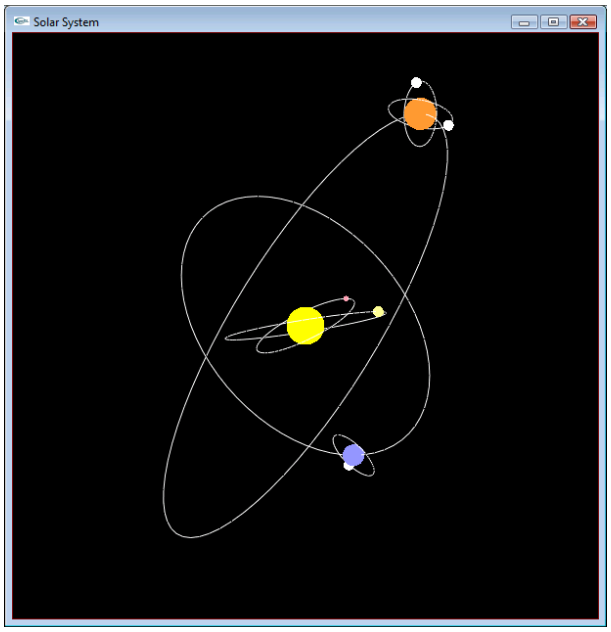
Assignment 2: Solar System in a Box

Out: 28th September 2016 **Due: 12th October 2016 at 11:59pm**



Objectives

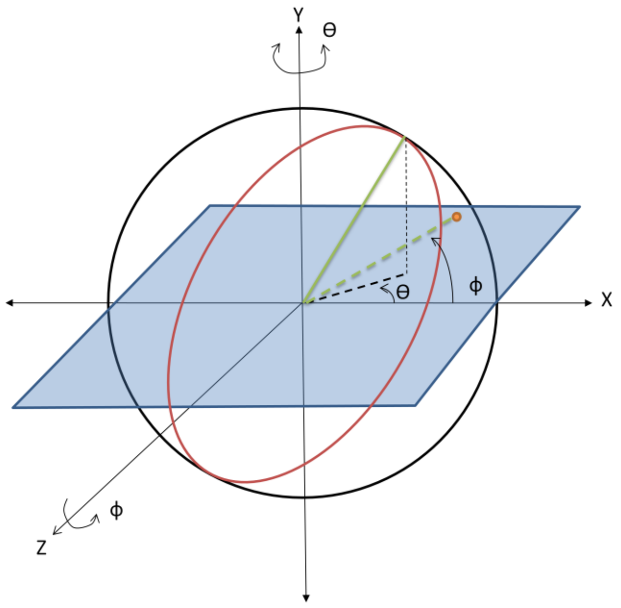
The first objective of this assignment is to practice using transformations to set up a 3D solar system scene and animate it. The second objective of this assignment is to implement the common "trackball interface" through an understanding of transformations.

Starting early is highly recommended, as this assignment make take significantly more time than the first assignment.

Part 1: Solar System (60 points)

In this part you will create a "mini solar system" consisting of four planets, two of which have moons. A static screen shot is shown above. You will create an animation of all the planets revolving around the sun, with the moons revolving around their respective planets.

All planets and moons are spherical. The orbit of every planet is a circle. The plane of this circle is given by two angles. Let the "default" orbit be centered at the origin and lie in the X-Z plane, with the default position of the planet at the point where this default orbit intersects the +X axis. Then ϕ is the angle (in degrees) that a given orbit makes with the X-Z plane, and Ө is the angle (in degrees) made by the X-Z projection of the line joining the planet to the origin with the X-axis. That is, the orbit is the circle obtained by rotating the default orbit by ϕ about the Z axis, followed by Ө about the Y axis.



You should use the following parameters for every planet:

Planet	Radius	Color	Orbit radius	Φ	ϴ	Speed
Mercury	10	Light pink	200	30	30	3
Venus	20	Light yellow	300	10	335	5
Earth	40	Light blue	550	-60	50	7
Jupiter	60	Orange	900	60	20	10

In addition, the following are the parameters for the moons (ϕ and ϴ for the moon's orbit are relative to the orbit of the planet they revolve around):

Belongs to	Radius	Color	Orbit radius	Φ	ϴ
Earth	20	White	100	20	10
Jupiter	20	White	120	-80	50
Jupiter	20	White	120	30	100

The radius of the sun is 70. All the orbits themselves should be drawn in white.

The whole solar system is centered at the origin of your world coordinate system. The entire solar system should be clearly visible in your screen window.

To produce the animation, you must use the orbital speeds mentioned above. Note that these are relative speeds, so you can use any scale factor to calculate the actual movement of every planet in your solar system, provided you use the same scale factor for all planets and moons. For example, you may increment the angle of rotation of Mercury by 3^o and Jupiter by 10^o respectively, and so on. Make sure that the speed is fast enough to observe multiple cycles of revolution, but slow enough to actually verify that each planet and moon is moving correctly.

Hint: Proceed in small steps and mentally work out how you should apply various transformations and the correct order. You may find it easy to start with the orbits, then the planets and finally their moons. Finally, animate the planets before animating the moons. You can use "sphere.obj" from the ObjViewer example to draw a sphere.

Part 2: Solar System in a Box (30 points)

In this part, you will pile on another set of transformations that will rotate the whole system according to mouse movements. Proceed as follows:

1. Draw a cube around the solar system using only lines. You can use the bounds of the solar system calculated in part 1 to determine the size of this cube. The solar system should be fully enclosed by the box at all times!
2. Scale the whole solar system and the cube so that you can see both clearly in the window.
3. Now implement a track-ball interface using the mouse. The trackball is an imaginary sphere surrounding your model. Imagine your model "pinned" to this sphere. As you rotate the sphere, you rotate the model accordingly. The trackball behaves as follows: if you press the left mouse and drag the mouse vertically downwards, the cube with the solar system rotates counter-clockwise around +X axis. If you drag it to the right, it will rotate counter-clockwise around the +Y axis, and so on. Dragging it diagonally will cause similar rotations about a "middle" axis between X and Y.

A user should be able to press the mouse button, drag the mouse nonlinearly before releasing with the model rotating correctly. That is, you may not assume that a mouse drag follows a straight line.

Hint: Every mouse movement will cause a rotation. Try out various mouse movements and try to determine how specifically to use rotation.

Part 3: Documentation (10 points)

Prepare a document detailing the features and details of this program. This document should include:

1. Screenshots of your program.
2. How to use your program once it is running.
3. An explanation of how you got the trackball to work. This description should be point-wise and algorithmic in nature, rather than paragraphs of text. Show Math and transformations wherever possible (matrix notation, not code snippets).

Program details: Your program should be robust to screen refreshing and resizing (the solar system should not be stretched or squeezed). Also the cube in Part 2 should always look like a cube irrespective of how you resize the window (i.e. the cube should not stretch/squeeze either). Under no circumstances should your program crash.

What to submit: Submit the entire JOGL/QT project as a single zipped file.