Project #2 - Modify the Solar System Demo by adding binary sun, moons, and an ice cap.
Due date: Wednesday, February 1, midnight.

Goals: Learn more about how to use OpenGL and interrupt-driven programming. Learn to use double buffering animation. Program some additions to an animated solar system. Use OpenGL commands to generate transformations that control the animation.

What to hand in: When you are done, place your C++ files, executable, and Visual Studio solution together in a separate folder, preferably named “Solar” or “Project2”, in your PC computer account in the APM basement labs. There should be nothing in this folder except your files for this homework assignment, and the creation/modification dates should be before the turn in deadline. The program must compile and run on these computers. Grading will be personalized and one-on-one with a TA or with Sam Buss. Your program must run on the PC lab, you must come into the PC lab and meet one of us. Alternately, ftp your folder to ieng6 and Prof. Buss can grade your program in his office hours. You will have to show your source code, run the program, make changes on the spot to your program and recompile as requested by the grader, and be able to explain how your program works and why it renders what it does. The grading must be completed no later than Wednesday one week after the due date.

FOR PROJECT #2, PLEASE DO THE FOLLOWING STEPS #1 - #10.

1. Download the Solar program from the zip file Solar_Winter2012.zip. Extract these into a directory named “Solar” or “Project2” in your home directory. This is the same as the program you downloaded for Project #0, but since you changed the Solar program in Project #0, you should download it again to make sure you have the right version. (Full URL for the zip file is: http://www.math.ucsd.edu/~sbuss/CourseWeb/Math155A_2012W/Project2/Solar_Winter2012.zip.)

2. Understand the code in the Solar program. Compile and run the program. Test out the keyboard controls. When the program first starts, aliasing causes the planet to appear to not be rotating, but instead always keeping the same face to the sun. Slow down the animation (up/down arrow keys) to see the "true" motion. Figure out how the animation code works. Understand what was meant by "aliasing" three sentences ago. (We will discuss this in class again, too.) See also the “Troubleshooting” section below.

By now, you should understand the callback-driven style of the program, and how the keyboard controls are handled by the program, and when drawScene is called.

3. Convert the sun to be a binary star. Replace the existing single sun by two smaller suns that rotate around each other in the center of the solar system. Have them revolve around each other approximately 20 times per earth year.

4. Give the Earth's moon a satellite of its own. This should be a small "moonlet" that orbits the Earth's moon four times every month.

5. Add a second, geostationary moon to the Earth. By “geostationary” is meant that the moon stays directly above the same point on the equator. That is, a person standing still at the right point on the equator, would always see the moon directly overhead.

6. As part of steps 3, 4, 5, adjust the orbital radii, the orbital rates, view distance, view angle, colors, etc. Make changes to the viewing distance, the viewing angle, and to the
sizes of the suns, planets and moons and to the radii of the orbits, so as to make viewing
the solar system convenient. This probably includes placing the view point further away
from the solar system to reduce the excess perspective. You may need to adjust the field
of view argument to \texttt{gluPerspective}. Choose colors that make all planets and moons
clearly visible and distinguishable.

7. \textbf{Add a 40 degree tilt.} Give the Earth and its moons and the moonlet a 40 degree
tilt. The kind of tilt is similar to the somewhat smaller tilt of our real earth that causes the
earth to have seasons. This means that the orbital path of the Earth should \textbf{not} be tilted;
instead the tilt just applies to the orientation of the Earth and to the path followed by the
moons. Thus the tilt should always be in the same direction (in the direction of the
negative \texttt{x} axis, for instance): it should \textbf{not} always be tilting at the same angle relative to
the sun. This means it will be “summer” in the northern hemisphere when the Earth is at
its rightmost position on the screen.

The \textbf{orbits} of the Earth's moon and its satellite should be tilted by the same amount so
that they are always above the equator of the planet. That is to say, the centers of the
Earth, the moon and the moonlet are co-planar and rotation axis for the Earth is
perpendicular to this plane.

The visual effect is that the Earth system is always leaning leftward 40 degrees. You
might have to use less perspective, or even try an orthographic viewpoint to see this well.
(The version of the program you hand in should use perspective view, however.)

8. \textbf{Add polar ice caps to the Earth.} There should be ice caps at both the north and south
poles, drawn as solid white (or light grey if you prefer). One suggestion is to use a
triangle fan to render this. The ice caps should rotate with the Earth.

9. \textbf{Implement a new keyboard control “w” to toggle showing the polar ice caps as filled-
in solid white, or as wireframes. This will let us see the rotation of the ice caps clearly.}

10. \textbf{Turn in the project as described above.}

\textbf{Program grading: Scale of 0 to 10. Personal grading session with TA Janine LoBue or
Professor Sam Buss.}

\textbf{Troubleshooting:}

- You may find the solar program runs way too fast, so that the earth and moon zip their
orbits many times per second.

There are two ways you can fix this. (a) The easiest way is to set the initial value for
the variable \texttt{AnimateIncrement} smaller. This causes the program to take smaller steps
in the animation of the solar system. Or, (b) on some computers you may be able to set
your display driver to force your program to slow down and show a single image per
screen refresh. For this, on my older computer with an ATI Radeon 9700, I open the
display driver properties (right-click on the root window), choose \textbf{Settings}, then
\textbf{Advanced}, then \textbf{OpenGL}, then set the \textbf{Wait For Vertical Sync} to be always on. This
causes a call to \texttt{glutSwapBuffers()} to wait until the screen is ready to redraw.

- The provided solar system code does not have any tilt on the Earth and moon system. It
might look like there is a little tilt due to the excessively high amount of perspective in
the version of the code provided. You can better view the tilt amount if you remove the
rotation around the x-axis that places the view point slightly above the orbital plane. You
will want to normally leave this rotation in your program, but might want to comment it
out for testing the tilt.