

## Computer Graphics 50:198:456/56:198:556 (Spring 2009)

<b>Homework:</b> 3	<b>Professor:</b> Suneeta Ramaswami
<b>Due Date:</b> 4/10/09	<b>E-mail:</b> rsuneeta@camden.rutgers.edu
<b>Office:</b> 321 BSB	<b>URL:</b> <a href="http://crab.rutgers.edu/~rsuneeta">http://crab.rutgers.edu/~rsuneeta</a>
	<b>Phone:</b> (856)-225-6439

---

### Programming Assignment #3

The goal of this programming assignment is to write a 3D graphics program of moderate complexity. The assignment involves geometric transformations, some simple viewing, and possibly some modeling.

**Problem Description:** Write a graphics program to model a planetary system consisting of the following objects: The Sun, and the planets Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, and Pluto in that order from the Sun (even though Pluto has been decreed not to be a planet, we'll continue to call it one in this assignment). The planets orbit the Sun and also rotate about their own axes. Several of the planets have moons. At a minimum, your program should include a moon for Earth and two moons for Jupiter that orbit the respective planets. The moons themselves do not rotate (hence "the dark side of the moon"). You will find more information than you need at <http://www.nineplanets.org>. Scroll down to "Solar System Data" for precise data.

**Note:** The next programming assignment will build on this one; *i.e.*, will require you to add features to this one. In particular, you will add lighting, material properties, and texture mapping to the planets. So design your program carefully - effort expended on this one will pay off on the next assignment as well.

More specific instructions are provided below in two parts. The first part relates to the modeling of the solar system itself. The second part describes a simple menu driven fly-through to be included in the program.

#### (A) Modeling the solar system.

- Depict the solar system with planets orbiting the Sun, moons orbiting their planets and planets rotating about their own axes. All of this should take place continuously, as in a real planetary system; that is, these movements should **not** depend on mouse clicks or any other input from the user. Take the Sun to be at the origin of your coordinate frame and assume that the orbits of the planets all lie on the  $xz$ -plane.
- The default position of the viewer is a point just above the positive  $z$ -axis and looking towards the origin, say  $x = 0, y = 2$  and  $z = d$ , where  $d$  is chosen sufficiently large so that all objects are in front of the viewer.
- The viewer's position is controlled by keyboard input, operating via a keyboard callback. Depressing the 'X' (respectively, 'x') key causes movement in the positive (respectively, negative)  $x$ - direction. Likewise for the 'Y' ('y') and 'Z' ('z') keys. The amount of movement per keystroke is left to you. Pick something that is neither too much nor too little (I don't want to miss any features while navigating your planetary system).
- Use perspective projection throughout, so that objects decrease in size as they move away from the viewer and increase in size as they near the viewer.

- Use display lists for all the objects in your solar system, as it will make a noticeable difference to the speed of the animation.

### Notes:

- You might find the program `planet.c` in Chapter 3 of the *Programming Guide* useful. I suggest typing up and running that program to see what it does. Of course, this program is quite crude and you should aim to do much better than that. Note that `glutSolidSphere` and `glutWireSphere` have the “north and south poles” on the  $z$ -axis.
- Include a “quit” option (as keyboard input, for instance).
- Indicate the orbits of planets by (faint) curves. You may assume that the orbits are circular (that is, you can ignore the eccentricity of the orbits).
- Use a depth buffer to do hidden-surface removal. In OpenGL this is done by including `GLUT_DEPTH` in the `glutInitDisplayMode` function, and by enabling the depth test through `glEnable(GL_DEPTH_TEST)`. See the *Programming Guide* for details.
- Include a simple feature on the planets (*e.g.*, dots) so that the rotation of a planet about its axis can be discerned. Another possibility is to render a planet as a solid sphere and surround it with a slightly larger wireframe sphere, both of which rotate. Also, choose colors appropriately for the planets (such as, blue for Earth and red for Mars). In the next assignment, you will use lighting and texture mapping to get more realistic effects.
- A planet doesn’t necessarily rotate about the vertical direction. Each planet rotates around its own axis which is at some angle to the perpendicular of the plane of the solar system. The Earth is only tilted 23 degrees from the vertical (which is why we have seasons) though other planets have other axial tilts. For example, Venus is basically “upside down” and rotates backwards, and Uranus has its axis in the plane of the solar system. Choose different tilts for the rotation axes of different planets.
- Use reasonably realistic proportions. The following table approximates some quantities of interest. All are normalized with respect to earth. For example, Jupiter’s diameter is roughly 11 times that of Earth, its distance from the Sun is five times that of Earth and it makes about 8.4% of its orbit around the Sun in the time that the Earth makes one full orbit. Keep in mind that using *completely* realistic proportions may make the solar system look very spread out (*i.e.* mostly black), making it difficult to see much. Also, the Sun and Jupiter may appear much too large. So tweak these numbers within reason to achieve the desired effect.

Planet	Diameter	Dist. from Sun	Orbital Velocity
Mercury	0.4	0.4	4.15
Venus	0.9	0.7	1.63
Earth	1.0	1.0	1.0
Mars	0.5	1.5	0.53
Jupiter	11.0	5.0	0.084
Saturn	9	9.5	0.034
Uranus	4	19	0.012
Neptune	4	30	0.006
Pluto	0.2	40	0.004
Sun	100.0		

- Feel free to add more features. **Note: This is required for graduate students.** For example, you could add Saturn (with its rings) to the set of planets, a comet with a highly elliptical orbit around the sun (with the sun near one of the foci, say), have a planet orbit in a plane other than the  $xz$ -plane (for example, Pluto’s orbit is noticeably off the  $xz$ -plane), or simulate an asteroid belt. You could also have a spacecraft (such as the space shuttle, a satellite, or something like the Voyager) orbiting a planet or traveling through the solar system - the possibilities are endless!

**(B) A menu driven fly-through.**

Implement a menu driven fly-through. Specifically, the left mouse button should allow the user to choose from a menu of options for the trajectory of the fly-through. Two types of trajectories should be incorporated, as described below:

- In the first type, the trajectory is a vector (an  $(x, y, z)$ -triple) that specifies a direction going through the center of the Sun. The idea is that the viewer starts *sufficiently far away* on the trajectory and flies in a straight line to the Sun. You should display what the viewer sees up until the time the viewer hits the surface of the Sun (s/he probably can’t see much by then, anyway). At this point, the viewer should return to the default viewing state. For example, a trajectory specified as  $(-1, 0, 0)$  means that the viewer starts off on the positive  $x$ -axis to the right of all the planets, and moves in the negative  $x$ -direction towards the Sun. To completely specify the fly-through, you will need to give the “up” vector. You may take this to be the positive  $y$ -direction. The speed of the fly-through should be reasonable: I won’t be able to see much if it’s too fast, but don’t make it unbearably slow either.
- In the second type of trajectory, the fly-through is along a curve; for example, along a circle, or an ellipse, or some other curve of your choosing. Choose the `at` and `up` arguments for `gluLookAt` so that you are guaranteed to see interesting sights. In other words, don’t design the fly-through in such a way that you see just black background most of the time.

The menu attached to the left mouse button should present a choice of trajectories:

- the first few menu entries should be some pre-defined directions of your choosing, given as  $(x, y, z)$ -triples,
- the next choice should be the curve trajectory (you may include more than one curve, if you wish), and
- the last choice should be labeled “other”, which will allow the user to input a straight line trajectory consisting of an arbitrary  $(x, y, z)$  direction and an arbitrary up vector. These values will be read in as standard input in the shell from which your program was run. Do not include this in the keyboard callback.

**Grading:** The assignment is worth 150 points. Your program will also be judged for “overall impression”. In other words, instead of simply implementing the requirements in a minimal way, aim for results that have visual appeal and are technically interesting. For graduate students, the extra features are worth 25% of the final grade.

**What to turn in:** Email me the **final version** of your program before midnight on April 10, 2009.