

CSCI 550 (Fall 2011)

Assignment #3

Handout: Tuesday, Oct. 25, 2011

Due: 11:49pm, Tuesday, Nov. 22, 2011

All assignments will be submitted through oncourse.

The non-programming part can be in any format such as Microsoft Word, PDF, plain text or scanned hand-writing.

The programming part should include: (1) all source code files; (2) the executable file (.exe); and (3) a README file containing necessary instructions, known bugs, and any other notes you would like me to read.

Part 1: non-programming problems (100 pts)

1. The parametric form of a typical helix is $(r \cos t, r \sin t, t)$. Write the parametric equations for two helix curves with radius r , one winding around the y -axis, moving towards $+y$ direction, and the other winding around the x -axis, moving towards $-x$ direction. The two curves will start at the same point (i.e. when the parameter $t = 0$).
2. A ruled surface is a surface formed by lines connecting two 3D curves at the corresponding parametric values.
 - (a) Write the parametric representation of the ruled surface, $S(s, t)$, formed by the following two curves: $(\cos s, \sin s, 0)$ and $(3 \cos t, 3 \sin t, 10)$.
 - (b) Derive the expression of the normal vector of this ruled surface at an arbitrary surface point $S(s, t)$.
3. How do you insert a polygon to a BSP tree? Write a pseudocode of your algorithm.
4. A certain camera's eye position is at the origin. If the camera is looking at $P = (0, -1, 0)$, with a view-up-vector $v = (-1, 0, 0)$, what is the resulting viewing matrix?
5. Suppose you are designing an integer z -buffer for flight simulation where all of the objects are at least one meter thick, are never closer to the viewer than 4 meters, and may be as far away as 100 km. How many bits are needed in the z -buffer to ensure there are no visibility errors (Show the process)?

Part 2: Programming problems (120 pts)

In this project, an OpenGL animation program will be implemented to simulate the solar system involving the Sun and the four nearest planets: Mercury, Venus, Earth and Mars, along with the moon of Earth and the two moons of Mars. The Sun only has self-rotation, and is located at the origin of your WCS. All objects are modeled as spheres. Each object rotates around its own axis with a certain *rotation period*, and at the same time rotates around the Sun (or the host planet in the case of moons). For example, the Earth rotates around the Sun once a year and around its own axis once a day. Each object rotates on a circular orbit that has a tilt angle, which is the angle between the normal of the orbit plane and the self-rotation axis of the central object of the orbit. The normal of an orbit plane can be generated by rotating the self-rotation axis of the central object by a given tilt angle about either X or Y axis. The tilt angle of the orbit of Earth is assumed to be zero. We also assume that each object's self-rotation axis is perpendicular to its orbit plane (i.e. the Z-axis of the orbit plane).

Since it is not practical to display the actual relative scales of the system, proper scalings need to be applied to the sizes of the objects and their orbits, as well as the rotation periods. Other specific requirements include:

- The system by default is in an animation mode. A pause/resume function should be provided to pause and resume the animation. During animation, the rotation angles for each frame need to be calculated carefully to reflect the relative rotation speeds of different objects.
- Shading and lighting are required for rendering. A frontal camera light will need to be provided that moves with the camera. Another light source is the Sun light at the origin of the solar system. Each of the two light sources can be turned on or off with menu or keyboard.
- Camera sliding, zooming, rolling, pitching and yawing need to be implemented using proper combinations of mouse movements, keys, and/or menus.
- Perspective projection (**gluPerspective**) will be used. Window resizing should be allowed with aspect ration preserved.

Bonus points (20 pts):

- Texture mapping: map the appropriate image to each planet or moon. The most convenient way to implement texture mapping for spheres is to use glQuadric to draw the spheres
- Allow the camera to focus on individual planet.