CPSC 314, Programming Project 2: RCSS Spaceship Camera Control

Out: Mon 28 Jan 2013. Due: Fri 8 Feb 2013, 5pm. Value: 8% of final grade

In this project, you will create a solar system and fly around it in two RCSS (Royal Canadian Space Survey) spaceships: the mothership and the scoutship. You will have two camera windows, one showing the view from the mothership and one showing that of the scoutship. Each ship also has a geometric representation, which you may be able to see in the other ship's window. There are four different ways to control the motion of the ships: rotate/translate in absolute solar system coordinates, free flying "through the lens" where incremental changes are made relative to the local camera coordinates, and geosynchronous orbit around a moving planet where you zoom in and out to the surface along a ray between the ship and the center of the planet. In geosync mode, the target geosynchronous planet is selected by hitting a key.

Modelling: [30 points total] Model a simplified solar system using spheres. The size of the sun/planets/moons and the distances between planets should **not** be to scale: you should be able to see all the planets and moons when the spaceship is far enough back to see the whole solar system. The time should also **not** be to scale: it should be much faster than real-time motion, so you can see things moving.

- (2 pts) Model the sun. It should spin around its own axis.
- (18 pts) Model the planets: Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto. The planets should each spin around their own axes and orbit the sun. It's OK to have all the planets be the same size, and have their orbits be equally spaced away from the sun. It's OK to have them spin at random rates instead of times proportional to their true rotational speed.
- (3 pts) Model the Earth's moon. It does not spin around its own axis: one side always faces the Earth, and one side always faces away. It should orbit the Earth.
- (1 pt) Draw circles showing the orbits of all the planets and moons - very helpful when debugging!
- (2 pts) Model Saturn's rings.
- (4 pts) Model the scoutship and mothership as very simple shapes.
- (extra credit: up to 2 pts) Some ideas: Color the planets. Make more complex ships. Make a more realistic solar system: add the moons of Jupiter, or make the planets spin at their true rates (perhaps accelerated so one day takes one second). Or make additional interesting solar systems to explore!
- Interaction details: Hitting 'p' turns on "freeze" mode. When freeze is on, time stops: all the planets/suns/moon stop spinning. You may find this mode useful for debugging. Hit 'P' to leave freeze mode.

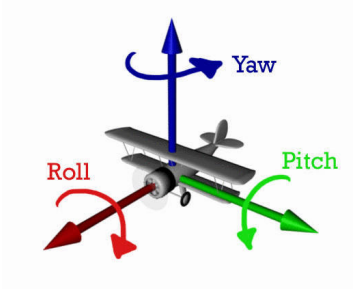
Navigation: [70 points total]
To toggle between navigating the mothership and the scoutship: use '>' to control the mothership, and '<' to control the scoutship. Use 'm' for a reset that resets the the location of the ships to be in a spot where both can see the entire solar system in their views, and the scoutship's geometry is visible in the mothership's view.

- **Absolute lookat [5 pts]:** Directly control the eye point, lookat point, and up vector in the global coordinate system of the entire solar system, using the keyboard. Enter this mode with the 'l' key. You'll increment a value with a lowercase key, and decrement it with an uppercase key, as follows:
  - eye point x with 'x'/'X'
  - eye point y with 'y'/'Y'
  - eye point z with 'z'/'Z'
  - lookat point x with 'a'/'A'
  - lookat point y with 'b'/'B'
  - lookat point z with 'c'/'C'
  - up vector x with 'd'/'D'
  - up vector y with 'e'/'E'
  - up vector z with 'f'/'F'

- increase/decrease speed (the increment moved by a keypress above) with '=/-' keys

This mode is easy to program, but hard to use if you want to follow a planet as it moves!

- **Relative flying [30 pts]:** The ship motion is calculated relative to the current camera coordinate system; that is, with respect to the view you see through the display window. So you can only move forward/backward in the direction that you're looking, like flying a plane. Use the keyboard to control your roll, pitch, yaw, and forward/backward speed. Enter this mode with the 'r' key.



Yaw is rotating horizontally with respect to your current position, like steering a car or shaking your head for no. Pitch is rotating vertically with respect to your current position, like nodding your head up and down for yes. Roll is tipping left or right by rotating around your current front-to-back axis, like tilting your head so your ear touches your shoulder. See also the animated illustration at http://www.nasm.si.edu/galleries/gal109/NEWHTF/PITCH.HTM

Interaction details:

- change yaw wrt local Camera coordinates with 'q'/'e' keys
- change pitch wrt local Camera coordinates with 'x'/'c' keys
- roll wrt local Camera up vector with 'a'/'d' keys
- forward/backward motion wrt local Camera z with 'w'/'s' keys
- increase/decrease speed (the increment moved by a keypress above) with '='/'-' keys

- **Geosynchronous orbit [35 pts]** The ship "locks on" to a target planet, and starts to orbit the planet at the same rate that the planet spins. The ship can zoom closer to and further from the planet's surface along a ray between the center of the ship and the center of the planet. Enter this mode with the 'g' key.

  Interaction details: The 'w' key moves the ship closer to the planet, the 's' key moves the ship farther away. Increase the speed at which you move forward/back with '=' and decrease with '-'. By default, the planet to orbit is Earth. Change the planet to orbit around with the numbers '1' (for Mercury) through '9' (for Pluto).

  This mode is hard to program, but makes it easy to track moving planets.

- **Extra Credit [Up to 2 pts]** Some ideas: show what planet a ship is orbiting by drawing a laser beam between from the ship to the planet. Support navigation via mouse drags in addition to just key presses. Add virtual trackball mode as another type of navigation. Add picking to geosynchronous mode: choose what planet to orbit by picking the planet that's under the cursor when the user hits a key.

Suggested Strategy

- For Saturn's rings, try squashing a sphere into a disk by scaling to a very small number or even zero.
- Remember that viewing transformations belong in the modelview matrix, not in the projection matrix. See Steve Baker's projection abuse article at http://sjbaker.org/steve/omniv/projection_abuse.html.
- You are allowed to reset the view to a default position when the user switches to a new camera mode: that is, to reset as if they had hit the 'm' key. It would be much more work to set the parameters of the new camera mode to match up with the old one to have the camera stay in the same spot, you are not required to do so.

- The freeflying mode requires incremental changes of roll/pitch/yaw angles and forward/backward motion with respect to the current camera coordinate system. You could imagine keeping track of cumulative roll/pitch/yaw values with respect to some global basis vectors (for example, kept in world coordinates), but that would require a **lot** of calculation. And transforming roll/pitch/yaw angles into the eye/lookat/up vector format required by gluLookat would be even more work.

  In contrast, if you assume that you know the current camera coordinate system (let's call it Current), it's easy to calculate the simple new incremental motion, where a drag means a simple motion with respect to the current x, y, or z axis of Current. This new incremental transformation (let's call it Incremental) needs to be applied with respect to the current transformation; that is, p' = Incremental * Current * p. The good news is that Current is exactly the modelview matrix used by OpenGL to draw the previous frame. If you don't wipe that out with glIdentity, that matrix is still intact and contains the information you need. However, OpenGL only allows you to postmultiply a matrix, which would result in the incorrect operation p' = Current * Incremental * p. The trick is to first explicitly store the Current matrix, which you can do with the `glGetDoublev` command that dumps out the contents of the top of the OpenGL matrix stack. Then you can get the desired effect by wiping the stack with glIdentity, first applying the incremental transformation, and finally multiply by the stored Current. This trick saves you a lot of work by using the OpenGL matrix stack as both a calculator and storage device!

- In geosynchronous mode, the spaceship should move with respect to the coordinate system of the locked-on planet. First try placing the spaceship geometry in that coordinate system, and debugging that motion by looking in that direction with the other ship's view. Then, to help you think about how to have a camera showing the view from the place where you have drawn the spaceship's geometry, try drawing the full scene graph to help you understand which matrices you might need to invert.

- Finish doing the entire required functionality before starting on any extra credit.

Handin/Grading/Documentation

The grading, required documentation, and handin will be the same as with project 1, except for two changes. First, use the command 'handin cs314 p2'. Second, there is no need to submit image files since there will not be a Hall of Fame for this project. Stay tuned for the ultimate Hall of Fame competition with Project 4!

Template Download

The template has two camera windows with a simple stand-in object, an axis. The code for all three platforms is combined within a single package. Download `http://www.ugrad.cs.ubc.ca/ cs314/Vjan2013/p2/P2_template.zip`, unzip, and follow the instructions in the README file for your platform (Linux, Windows, or Mac).