CPSC 314, Programming Project 2: RCSS Spaceship Camera Control

Out: Wed 10 Feb 2016. Due: Tue 1 Mar 2016, 11:59pm. 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 three different ways to control the motion of the ships: changing the position, lookat point, and up vector in absolute solar system coordinates; relative flying *through the lens* where incremental changes are made relative to the local camera coordinates; and zooming in and out from geosynchronous orbit around a moving planet by translating along a ray between the ship and the center of the chosen planet. The interaction will be through both key presses and mouse drags.

You are allowed to use the full functionality of `three.js` including scenes, cameras, and geometry, and matrix autoupdating. You may want to use the `Object3D` support for scene graph manipulation with parenting through `add, remove`, for positioning through `position, up, lookAt, applyMatrix`, and for matrix autoupdating through `updateMatrix, updateMatrixWorld, matrixWorld`. You may want to use the `Matrix4` class, including `makeTranslation, makeRotation, makeScale, multiply, makePerspective`. You may want to use the classes for `Camera` and `PerspectiveCamera`, including `matrixWorldInverse, projectionMatrix, lookAt, fov, aspect, near, far`.

Modelling: [30 points total] Model and animate 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.

- (1 pt) The template already contains the sun model. Add animation so that it spins around its own axis.
- (16 pts) Model the planets: Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune. Animate the planets so that each spins around its own axis 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, and have their axes point straight up. It's OK to have them spin at random rates instead of times proportional to their true rotational speed. However, the planets should orbit at different rates.
- (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.
- (2 pts) Draw circles showing the orbits of all the planets and moons. You may find this geometry helpful when debugging.
- (2 pts) Model Saturn's rings.
- (4 pts) Model the scoutship and mothership geometries as very simple shapes.
- (2 pts) Interaction: Implement *freeze* mode which should be toggled on and off with the 'space' key. When freeze is on, time stops: all the planets/suns/moon stop orbiting/spinning. You may find this mode useful for debugging.
- (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) and assign their true tilt. Give the sun some *glow*. Make the orbits elliptic. Add dwarf planets like Pluto and Eris. Or make additional interesting solar systems to explore!

Navigation: [70 points total]

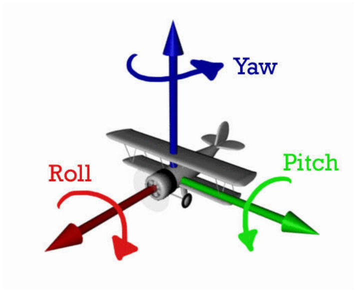
- **Camera target [5 pts]:** To toggle between navigating the mothership and the scoutship: use 'o' to control the mothership, and 'p' to control the scoutship. Use 'm' for a reset that resets the the location of the ships to be in a spot where both can see the entire solar system in their views, and the scoutship's geometry is visible in the mothership's view.
  - Control mothership with 'o'.
  - Control scoutship with 'p'.
  - Reset both cameras with 'm'.
- **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. Interaction details:

- Increase/Decrease camera x location with 'x'/'X'
- Increase/Decrease camera y location with 'y'/'Y'
- Increase/Decrease camera z location with 'z'/'Z'
- Increase/Decrease the x location the camera is looking at 'a'/'A'
- Increase/Decrease the y location the camera is looking at 'b'/'B'
- Increase/Decrease the z location the camera is looking at 'c'/'C'
- Increase/Decrease the value of the up vector in the x coordinate 'd'/'D'
- Increase/Decrease the value of the up vector in the y coordinate 'e'/'E'
- Increase/Decrease the value of the up vector in the z coordinate 'f'/'F'
- Increase/decrease step size (the increment moved by a keypress from above) with keys 'k/K'
- Enter lookat mode with 'l'

The cameras should act independant of each other. Changes to one camera should in no way effect the other camera.

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 or the mouse to control your roll, pitch, yaw, and forward/backward speed. Enter this mode with the 'r' key.



Yaw is rotating horizontally with respect to (wrt) 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://howthingsfly.si.edu/flight-dynamics/roll-pitch-and-yaw`

Interaction details:

- Change yaw wrt local Camera coordinates with increment/decrement keys 'q'/'Q' or with the horizontal component of the left mouse drag.
- Change pitch wrt local Camera coordinates with increment/decrement keys 's'/'S' or with the vertical component of the left mouse drag.
- Roll wrt local Camera up vector with increment/decrement keys 'a'/'A'
- Forward/backward motion wrt local Camera z with increment/decrement keys 'w'/'W', or with the vertical component of the mouse drag when the 't' key is held down. The vertical magnitude of the drag should control the translation distance.
- Increase/decrease speed (the increment moved by a keypress above) with increment/decrement keys 'k'/'K'.
- Enter relative flying mode with 'r'.

When moving with mouse drags, you should keep track of the start and end location of the cursor over the extent of the drag, so that changes to yaw and pitch only happen while the mouse button is held down: these changes stop when the button is released. Dragging upward increases pitch, dragging downwards decreases it. Similarly, dragging rightward increases yaw, dragging leftward decreases it.

Note that this motion is different than the three.js controls-fly example demo/code at `http://threejs.org/examples/#misc_controls_fly`, where the pitch and yaw motion is controlled by the distance between the current cursor location and the center of the window, and is always active whether or not the button is down. However, this code is a good example of how to access mouse status.

You may want to scale down the rotation computed from the changes in screen/display coordinates. So the camera does not rotate too fast.

- **Geosynchronous orbit [30 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 centre of the ship and the centre of the planet. Enter this mode with the 'g' key.

  Interaction details:

  - Move closer/further to planet with keys 'w'/'W' or with the vertical component of the mouse drag, as above.
  - Increase/decrease speed (increment moved by a keypress above) with key 'k'/'K'.
  - Change the planet to orbit around with the numbers '1' (for Mercury) through '8' (for Neptune). By default, the planet to orbit is Earth (number '3').
  - Enter geosync mode with 'g'.

- **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. Add virtual trackball mode as another type of navigation.

Suggested Strategy

- 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. (You could do this for extra credit.)

- For Saturn's rings, try THREE.RingGeometry.

- The relative *through the lens* flying 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.

  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 to draw the previous frame! If you have taken care not to clobber it, you can read it out from the Object3D scene graph with `matrixWorld`. This trick saves you a lot of work by using the three.js scene graph infrastructure as both a calculator and storage device! For more about three.js matrix transformations and automatic updating, see `http://threejs.org/docs/index.html#Manual/Introduction/Matrix_transformations`

- 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 camera. 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 sketching the full scene graph to help you understand which matrices you might need to invert.

- Remember that it is good practice to keep viewing transformations in the modelview matrix, not in the projection matrix. A good discussion of the reasons is in Steve Baker's projection abuse article at `http://www.opengl.org/archives/resources/faq/technical/projection_abuse.php`; although the syntax details of the old fixed-function OpenGL are different than modern shader-based WebGL, the fundamental ideas still hold.

- In order to get the freeze/pause working well you may need to implement *simulation time*. *Simultation time* is a seperate time that is relative to your simulation. This will help when you freeze and want to resume with planets in their current configuration. *Simulation time* only updates when your system is NOT frozen.

- 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 the Sun geometry, plus an axis to help you maintain your orientation. Download from `http://www.ugrad.cs.ubc.ca/~cs314/Vjan2016/P2.zip`, unzip, and open `P2/P2.html` in your web browser.
For further examples of working with multiple camera windows, see `http://threejs.org/examples/#webgl_multiple_views`

References

1. `http://howthingsfly.si.edu/flight-dynamics/roll-pitch-and-yaw`
2. `http://threejs.org/examples/#misc_controls_fly`
3. `http://threejs.org/docs/index.html#Manual/Introduction/Matrix_transformations`
4. `http://www.opengl.org/archives/resources/faq/technical/projection_abuse.php`
5. `http://threejs.org/examples/#webgl_multiple_views`
6. `http://threejs.org/docs/`